TIBCO iWay® Service Manager

HTTP Solutions Development Guide

Version 8.0 and Higher
March 2021
DN3502291.0321

Contents

Chapter 1

Introducing iWay HTTP Solutions

This section provides an introduction to iWay Service Manager (iSM) HTTP solutions. For more information on additional transport protocol adapters that are supported by iSM, see the *iWay Service Manager Protocol Guide*.

In this chapter:

HTTP Services Overview

iWay Service Manager (iSM) Hypertext Transfer Protocol (HTTP) services provide tools that enhance the development and implementation of applications where file transport services are required. The enhancements are accomplished by:

- reducing the requirement for custom programming when implementing a range of file transport services; and by
- providing access to packaged third-party adapter products.

In minimizing the amount of code required for their implementation, these HTTP services provide a solid foundation for flexible, service-oriented architecture.

What is HTTP?

Before you begin using the HTTP tools that are available in iSM, it is recommended that you have a good understanding of HTTP, and the types you plan to support.

Native HTTP

Hypertext Transfer Protocol (HTTP) is the foundation of data communication for Web browsers and servers. It is the default network request/response system used to transfer files from one host to another over the Internet.

HTTP is an application-layer network protocol built on top of Transmission Control Protocol - Internet Protocol (TCP/IP) which enables two hosts to establish a connection between them and exchange files or data streams. In addition:

- HTTP employs the client-server model, using separate control and data connections between a client (such as a Web browser) and a server.
- HTTP users can connect and authenticate themselves using a clear text sign-in, but can also connect anonymously (when the server is so configured).
- HTTP is a stateless protocol; one that processes requests without any knowledge of requests that came before. It processes one request/response pair at a time.
- The HTTP connection can be closed after a single request/response is processed. Optionally, the connection can remain open (or persistent) which can result in a decrease in system latency.
- Communication can be monitored through HTTP status codes which appear on the first line of the response file.
- HTTP communication is unencrypted, typically through port 80 or some related port.

HTTP first came into common use in 1996, prior to the creation of today's encryption mechanisms. Other aspects of the protocol that have come to be viewed as shortcomings are being addressed in technologies such as JavaScript and ActiveX. Nevertheless, HTTP remains a bedrock technology. One reason for this is encryption capability has been appended to HTTP as detailed in the following section.

Secure HTTPS

To implement secure transmissions that encrypt the user name, password, and content, HTTP is often protected with Secure Sockets Layer - Transport Layer Security (SSL/TLS). The designation HTTPS indicates HTTP that uses SSL/TLS.

In addition, HTTPS provides bidirectional encryption for counterparty authentication, designed to prevent packet capture and other man-in-the-middle attacks. Originally used for payment transactions on the World Wide Web, HTTPS grew to be used for e-mail and for any other private transactions in corporate information systems.

Due to encryption processing, it may be slower than native HTTP. HTTPS typically uses port 443.

HTTP versus HTTPS

The following table compares HTTP to HTTPS.

HTTP	HTTPS
Default Port - 80	Default Port - 443
Additional Protocol - No	Additional Protocol - Yes. Uses the Secure Sockets Layer (SSL) which transports secure data to and from the server.
Sender/Receiver Identification - No	Sender/Receiver Identification - Yes. SSL encrypts the data, preventing third party access.
Certificate - No	Certificate - Yes. The owner of the Web site purchases a certificate from a trusted authority.

Non-blocking nHTTP

Native HTTP is synchronous, processing one request/response pair at a time. It blocks the next request, preventing it from beginning until the active request has completed. However, this behavior is not always desirable.

Non-blocking HTTP (nHTTP) is asynchronous. It permits subsequent requests to begin processing before the current request has completed. In addition to improved performance, nHTTP provides an array of configurable parameters for security, connectivity, and header manipulation.

HTTP versus nHTTP

The following table compares HTTP to nHTTP.

HTTP	nHTTP
Synchronous - waits for a task to finish before processing the next task.	Asynchronous - Begins to processes other tasks before the current one completes.

HTTP	nHTTP
Blocking - processes one request/response pair at a time - the protocol handler is not required to match a request to its paired response.	Non-Blocking - can process a request and its related response independently from one another, therefore the protocol handler must match a request to its paired response.
Stateless - provides a response after a request, then requires no further attention. For high availability, a stateless service requires redundant, load-balanced instances.	Stateful - can process subsequent requests to the service depending on the results of a prior request. For high availability, a stateful service can use an active/passive or active/active configuration.

iWay Service Manager Suite of HTTP Tools

iWay Service Manager (iSM) offers a full complement of HTTP services that are designed specifically to be used in transactional situations. These services include client and server-side, RFC-compliant capabilities for HTTP, HTTPS, and nHTTP.

This section describes features in the iSM HTTP suite of tools for the client side and the server side of the HTTP suite.

Client Suite

The HTTP client suite allows iSM to connect to and interact with an HTTP, HTTPS, or nHTTP server. The client is capable of detecting the existence of a file on the remote server and reacting to this existence (through an iSM HTTP, HTTPS, or nHTTP listener).

The reaction of the client based on the existence of a file could be as simple as retrieving the file. It can also involve a complex iSM process flow that includes a series of steps that can manipulate that file or a group of files on the remote system or other systems that iSM is capable of accessing.

Sample Applications

The HTTP client suite of tools can be used to handle a variety of application requirements, such as:

- The HTTP/nHTTP listener can be configured to monitor a directory on the server.
  - When a specific file name (and/or extension) is written to that directory, that file will be retrieved and saved to a directory on the client.

- Alternately, the file that is retrieved can be presented to an iSM process flow and used as the input for additional processing by iSM (for example, updating a database table and initiating an SAP transaction).

- When used in conjunction with other iSM listeners, the HTTP/nHTTP clients can be used within an iSM process flow to transfer data from the client system to the server.
  - A messaging listener such as MSMQ or JMS can be used to take a message from a queue. The message itself or specific message contents can be transferred to the server.
  - An SAP IDoc that is obtained using the iWay Application Adapter for mySAP ERP (SAP JCo 3.x) could be sent to the server.
  - Sending an EDI file to a VAN.

Server Suite

The HTTP server suite of tools provides two listeners:

- HTTP/HTTPS Server Listener
- nHTTP Server Listener

Both listeners have the same characteristics. The listeners can be configured to interact with the client in the same way that any HTTP/nHTTP server would. For example, handling the request of the client (send, receive, rename, delete, and so on) as the server. The advantage of the HTTP server suite of tools is the ability to configure the listeners to use the file(s) that are received as messages that can be used to initiate a complex iSM process flow. Corresponding results are then returned from those process flows.

Security

The HTTP server suite of tools can be configured to handle login security using standard server authentication realms (LDAP, RDBMS, text based property file, Kerberos, and so on) as well as a complete directory authorization capability and user role tracking.

Sample Applications

The HTTP server suite of tools can be used to handle a variety of application requirements, such as:

- Receiving transaction messages (including EDI) from partners who need to use a standard protocol, and then passing these messages to a process flow. This a good solution for participating in standard EDI networks by EDI splitters and transformers.

- Receiving files and using a process flow to redistribute the incoming files to one or more internal recipients.
- Act as a relay for large files where not having the actual file materialized on a local disk is required. This can include the receipt of a file outside of a firewall and then relaying the file through the firewall.

Common Listener Functionality

All listeners offer the standard server startup failure flows to handle processing issues when the listener begins and when specific error conditions occur during the operation of the listener.

Table of HTTP Listeners

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

Listener Name
HTTP Listener Parameters on page 16
How to Configure an nHTTP Listener on page 23

Table of HTTP Emitters

The following table provides a quick reference to the iSM emitters that are defined in this documentation for HTTP services.

Emitter Name
HTTP Emitter Parameters on page 20
HTTP Nonblocking Emitter Configuration Parameters on page 39

Table of HTTP Services

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

Service Name
Add Attachment From File Service (com.ibi.agents.XDAddAttachmentFromFileAgent) on page 92
Add Attachment Service (com.ibi.agents.XDAddAttachmentAgent) on page 93
Attachment Operations Service (com.ibi.agents.XDAttachOps) on page 94.
Attachment to Document Service (com.ibi.agents.XDAttachmentToDocAgent) on page 95
Cross-Origin Resource Sharing Service (com.ibi.agents.XDCorsAgent) on page 96
Document to Attachment Service (com.ibi.agents.XDAttachmentFromDocAgent) on page 102
HTTP Cookie Agent Service (com.ibi.agents.XDCookieAgent) on page 103
HTTP Emit Service (com.ibi.agents.XDHTTPEmitAgent) on page 104
HTTP Nonblocking Emit Service (com.ibi.agents.XDNHttpEmitAgent) on page 110
HTTP Read Agent (com.ibi.agents.XDHTTPReadAgent) on page 114
HTTP ReST Routing Service (XDReSTRouteAgent and XDReSTRouteReviewer) on page 117
HTTP Session Invalidator Service (com.ibi.agents.XDHttpSessionInvalidator) on page 38
OAuth 1.0 Authentication Service on page 126
OAuth 2.0 Authentication Service on page 131
WS HTTP Client Agent (com.ibi.agents.XDWSHttpClientAgent) on page 139

Table of HTTP Preparsers

The following table provides a quick reference to the iSM preparsers that are defined in this documentation for HTTP services.

Preparser Name
HTTP Preparser (com.ibi.preparsers.XDHTTPpreParser) on page 141
Multipart for nHTTP Preparser (com.ibi.preparsers.XDMultiPartForNHTTP) on page 142

Chapter 2

Configuring HTTP Components

The protocols for the iWay transport utility protocol adapters support email exchanges and file transfers between Internet users. The protocols also support exchanges and file transfers between those connected through TCP/IP and other networks.

iWay transport utility protocol adapters provide tools that simplify the implementation of service-oriented architectures. The simplification is accomplished by reducing the requirement for custom programming when implementing a range of distributed messaging services and file transport services and by providing access to packaged third-party adapter products. By minimizing the amount of code required for their implementation, these adapters provide a solid foundation for flexible service-oriented architecture.

In this chapter:

HTTP

The Hypertext Transfer Protocol (HTTP) is an application-level protocol with the lightness and speed required for distributed, collaborative, hyper-media information systems. HTTP enables easy management of agent resources through management applications, such as Firefox, Internet Explorer, and so on, which are readily available in all systems.

The iWay Adapter for HTTP/S provides the iWay Service Manager Server Integration platform with the ability to send and receive messages over secure HTTP. It enables you to take advantage of the Internet by providing secure, reliable, and efficient communication with external business systems, whether within or outside of your enterprise.

The iWay Adapter for HTTP/S:

- Provides a bidirectional adapter.
- Enables HTTP clients to invoke business processes within iWay Service Manager through a URL.

- Enables a business process to request data from an HTTP server through a URL, thus acting as an HTTP client.
- Includes a built-in multi-threaded HTTP listener, as part of the iWay Service Manager base configuration, which is used to serve the iWay Configuration pages to the user.
- Supports HTTP and HTTPS protocols, both synchronous and asynchronous bidirectional invocation, and includes a limited-use license for the XML Data Handler.

For inbound messages, the adapter receives incoming messages through the registered URL, with the message payload being either an XML message or an arbitrary proprietary data structure.

HTTP Listener Parameters

The following table lists and describes the HTTP listener parameters. For instructions on creating a listener, see *Configuring Listeners* on page 153.

Note: Parameters that are common to all components are omitted from the following table. For a list of these properties refrer to *Configuring Common Parameters* on page 149 in the appendix of this manual.

Parameter	Description
Port (required)	The TCP port for receipt of HTTP requests.
Local bind address	The local bind address for multi-homed hosts. This is usually left empty.
Document Root (required)	The base directory from which all HTTP pages are served.
Timeout	The timeout interval for the TCP socket.
Set TCP No Delay	If set to *true*, this disables Nagle's Algorithm on the client socket. This results in faster line turnaround at the expense of an increased number of packets.*False* is the default.
Defer Close of Socket	If set to *true*, (default) the close of the client socket is deferred for one second after the response is written. This compensates for an issue seen on some older versions of z/OS.

Parameter	Description
Default Page	The default page if none is identified in the incoming HTTP or HTTPS request.
Response content type	This overrides content type of response. Select one of the following options from the drop-down list: ❏ *application/EDI-X12* ❏ *application/EDIFACT* ❏ *application/XML* ❏ *text/html* ❏ *text/plain*
Default Text	The default text sent with *200 OK*. The default text takes the configured Content Type.
Keystore	The full path to the keystore file, which provides certificate material to be used for a secure connection.
Keystore Password	The password to access the keystore file.
Keystore Type	The type of keystore file. *JKS* is the default value.
Truststore	The file that provides trust certificates used to authenticate clients. Leave this blank to use the default JVM truststore. For more information on security, see the *iWay Adapter for EDIINT User's Guide*.
Truststore Password	The password to access the truststore file, if required.
Truststore Type	The type of truststore. For more information on security, see the *iWay Adapter for EDIINT User's Guide*.
Security Provider Class	This will override the default Sun security provider, which is: `com.sun.net.ssl.internal.ssl.Provider`

Parameter	Description
Security Protocol	The protocol to enable security. Security protocol values include: ❑ *SSL* - Supports some version of SSL. May support other versions. ❑ *SSLv2* - Supports SSL version 2 or higher. ❑ *SSLv3* - Supports SSL version 3. May support other versions. ❑ *TLS* - (default) Supports some version of TLS. ❑ *TLSv1* - Supports TLS version 1. May support other versions.
Security Algorithm	This overrides the default security algorithm, which is: `SunX509`
Client Authentication	If set to *true*, authentication is required from the client. *False* is the default.
Require Authorization	If set to *true*, the listener implements HTTP basic authentication using one or more authorization drivers. *False* is the default.
Always reply to listener default	If set to *true*, the default reply definition is used in addition to defined replies. *False* is the default.
Channel Failure Flow	Name of published process flow to run if this channel cannot start or fails during message use. The server will attempt to call this process flow during channel close down due to the error.
Channel Startup Flow	The name of the published process flow to run prior to starting the channel.
Channel Shutdown Flow	The name of the published process flow to run after the channel is shut down.
Startup Dependencies	A comma-separated list of channel names that must be started before this one is called.

Note: The HTTP listener supports streaming. Streaming is used for large documents or documents for which the application needs to split the input into sections under the same transaction. For more information on streaming and configuring streaming preparsers, see the *iWay Service Manager Component and Functional Language Reference Guide*.

Reference: HTTP Listener Special Registers

The following table lists and describes the special registers on the HTTP listener.

Name	Level	Type	Description
...	Header	String	Each header value from the message.
action	Document	String	The action field of the post.
docroot	Document	String	The defined docroot from configuration.
ip	Document	String	The IP address of the sender.
iwayconfig	System	String	The current active configuration name.
iwayhome	System	String	The base at which the server is loaded.
iwayworkdir	System	String	The path to the base of the current configuration.
msgsize	Document	Integer	The physical length of the message payload.
name	System	String	The assigned name of the master (listener).
protocol	System	String	The protocol on which message was received.
requestType	Header	String	The type of HTTP request (GET, POST, or HEAD).
source	Document	String	The host name of the sender.
url	Header	String	The type of full URL of the HTTP (GET, POST, or HEAD).
tid	Document	String	Unique transaction ID.

HTTP Emitter Parameters

The following table lists and describes the HTTP emitter parameters. For instructions on creating an HTTP emitter, see *Configuring Emitters*.

Parameter	Description
Configuration Parameters HTTP Emitter	
Destination (required)	The URL to which to post this information.
Action Method	Select *GET* (with data on the URL and URL encoded) or *POST* (with a content length header).
Request content type	This overrides request content type value. Choose from among the following values: ❑ *application/EDI-X12* ❑ *application/EDIFACT* ❑ *application/XML* ❑ *text/html* ❑ *text/plain*
User ID	A valid user ID for basic authentication challenges.
Password	A password associated with the user ID.
Response Timeout value in seconds	The time (in seconds) to wait for response before signaling an error as integer.
IP Interface Host	The local IP interface from which the outgoing IP socket originates.
IP Interface Port	The local IP port from which the outgoing IP socket originates.
Relay Inbound Content Type	If set to *true*, then the relay headers are received as content type.

Parameter	Description
Set TCP No Delay	If set to *true*, then this parameter disables the Nagle Algorithm on the client socket. This results in a faster line turnaround at the expense of an increased number of packets.
Proxy	
Proxy	If set to *true*, emit through proxy server. *False* is the default.
Proxy URL	The URL of the proxy server.
Proxy User ID	The user ID for proxy authentication challenges.
Proxy Password	The password to access proxy server.
HTTPS	
Secure Connection	If set to *true*, then the emitter uses a secure connection. You may be required to configure the keystore under the HTTPS section of the system properties if client authentication is required, or if a certificate is used that does not have a matching entry in the default truststore of the JVM. *False* is the default.
Use 128-bit Encryption	Select *true* to use 128-bit encryption. *False* is the default.
Security Protocol	Select from the drop-down list. *SSL* - Supports some version of SSL. May support other versions. *SSLv2* - Supports SSL version 2 or higher. *SSLv3* - Supports SSL version 3. May support other versions. *TLS* - (default) Supports some version of TLS. May support other versions. *TLSv1* - Supports TLS version 1. May support other versions.

nHTTP

The nHTTP adapter is a nonblocking HTTP with improvement in performance, connection management, and various other security features.

The nHTTP adapter provides extensive flexibility by exposing an array of configurable parameters for security, connectivity, and header manipulation. Below are descriptions of some features that have been added as part of the improvement to the nHTTP component.

nHTTP REST Support

Representational State Transfer (REST) is a simpler alternative to SOAP and Web Services Description Language (WSDL)-based web services. There are many advantages of using this simpler HEEP-based design approach to calling server-based services, and it has been widely adopted by many advanced web service providers including Amazon, Google, and Facebook. These entities have either avoided SOAP or offered REST as a simpler alternative.

A RESTful service:

- Uses explicit HTTP methods.
- Is stateless (no session management is allowed or required).
- Exposes the calling URI as a directory-like structure.
- Transfers the payload as XML, JavaScript Object Notation (JSON), or both.

iWay Service Manager (iSM) offers a group of facilities to enable the use of RESTful services when executed by process flows. The nHTTP listener for iSM complies with the HTTP 1.1 specification. This listener implements all of the available verbs used in REST-style communication, including the main verbs (GET, POST, PUT, and DELETE).

iWay Providers

Named SSL Context Provider

Since this provider uses configured keystore/truststore providers, it allows you to configure multiple SSL context providers and use them as named providers in the nHTTP configuration.

Features

- **Persistent Connection Support.** The nHTTP adapter supports persistent connections, which will allow for better connection handling and management.
- **Session Resumption.** Session resumption is one of the new features available for the SSL configuration.

- **Large File Limit.** The nHTTP adapter contains various internal improvements to handle large file sizes. As an addition, a new option has been exposed on the nHTTP inbound processing that allows the user to limit the message size accepted by the adapter.

Configuring nHTTP Listeners

A listener is a component that is responsible for receiving inbound messages through an assigned listener protocol. After a listener is created, it must be added to an inlet configuration. An inlet will become part of the final channel configuration that will consist of an inlet, route, and an outlet. For more information on configuring channels, see the *iWay Service Manager User's Guide*.

Procedure: How to Configure an nHTTP Listener

To configure an nHTTP listener:

1. Ensure that iWay Service Manager is running.

   On Windows, you can start iWay Service Manager by clicking *Start*, selecting *Programs*, *iWay 7.0 Service Manager*, and then *Start Service Manager* for the configuration you are currently using.

   For more information on starting and stopping iWay Service Manager, see the *iWay Service Manager User's Guide*.

2. Open a browser window and point to the following URL:

   `http://host:port/ism`

   where:

   `host`

   Is the host machine on which iWay Service Manager is installed.

   `port`

   Is the port on which iWay Service Manager is listening. The default port is 9999.

   On Windows, alternatively, you can click *Start*, select *Programs*, *iWay 7.0 Service Manager*, and then click *Console*.

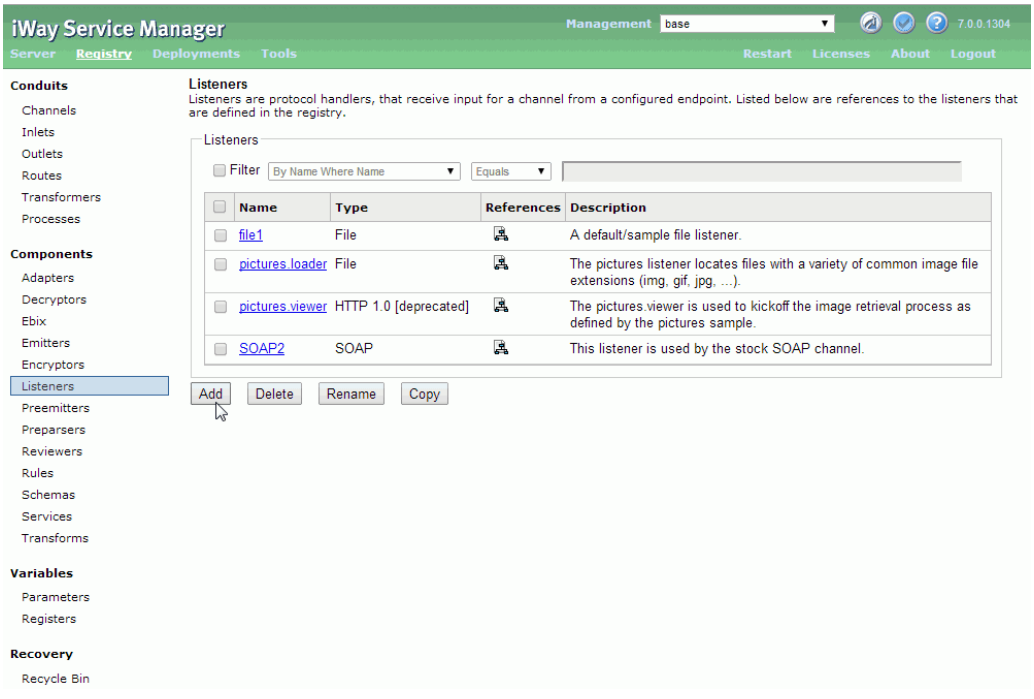
   A login dialog box opens.

3. Type a user name and password for the configuration you are using, and click *OK*.

   The iWay Service Manager Administration Console opens.

4. Click *Registry* in the top pane, and then click *Listeners* in the left pane.

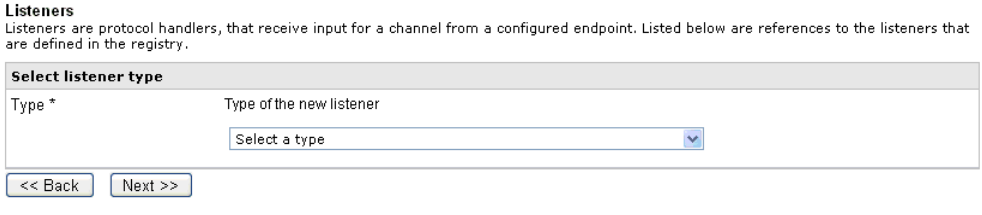
The Listeners pane opens.



The table that is provided lists all the previously configured listeners and a brief description for each.

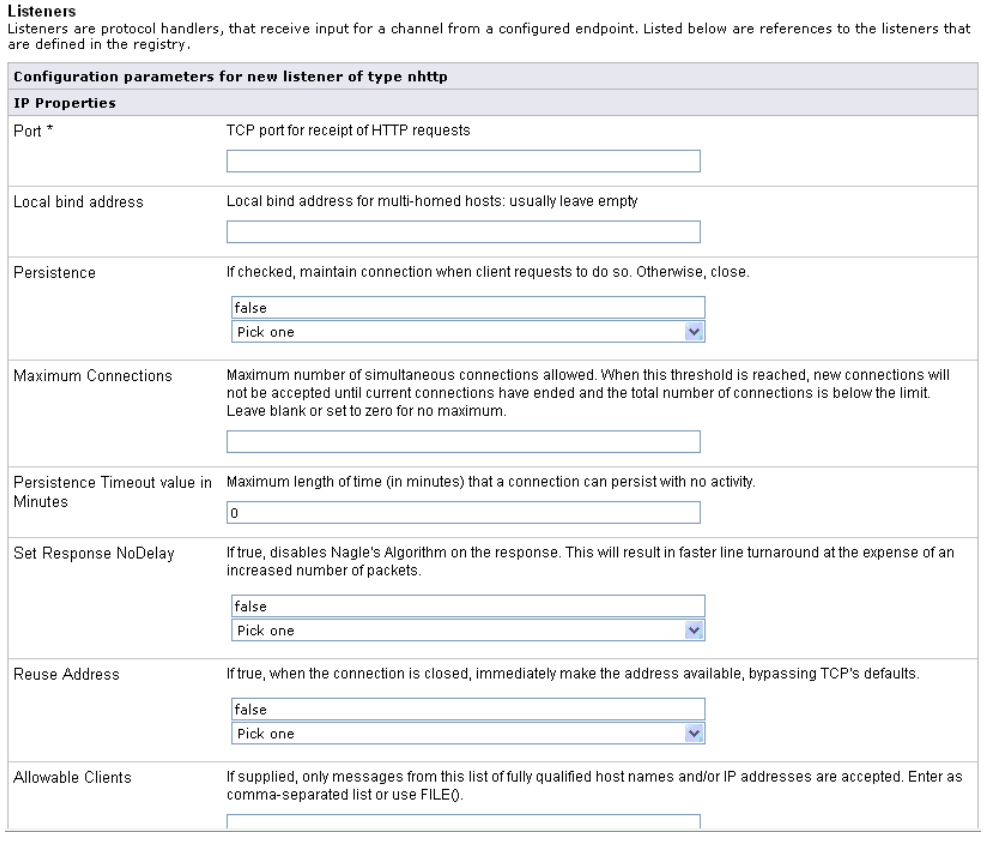
5. Click *Add*.

   The Select listener type pane opens.



6. Select *HTTP 1.1 [nonblocking] (nhttp)* from the Type drop-down list and click *Next*.

The Configuration parameters for the nHTTP listener pane opens.



Listeners
Listeners are protocol handlers, that receive input for a channel from a configured endpoint. Listed below are references to the listeners that are defined in the registry.

Configuration parameters for new listener of type nhttp	
IP Properties	
Port *	TCP port for receipt of HTTP requests
Local bind address	Local bind address for multi-homed hosts: usually leave empty
Persistence	If checked, maintain connection when client requests to do so. Otherwise, close. false Pick one
Maximum Connections	Maximum number of simultaneous connections allowed. When this threshold is reached, new connections will not be accepted until current connections have ended and the total number of connections is below the limit. Leave blank or set to zero for no maximum.
Persistence Timeout value in Minutes	Maximum length of time (in minutes) that a connection can persist with no activity. 0
Set Response NoDelay	If true, disables Nagle's Algorithm on the response. This will result in faster line turnaround at the expense of an increased number of packets. false Pick one
Reuse Address	If true, when the connection is closed, immediately make the address available, bypassing TCP's defaults. false Pick one
Allowable Clients	If supplied, only messages from this list of fully qualified host names and/or IP addresses are accepted. Enter as comma-separated list or use FILE().

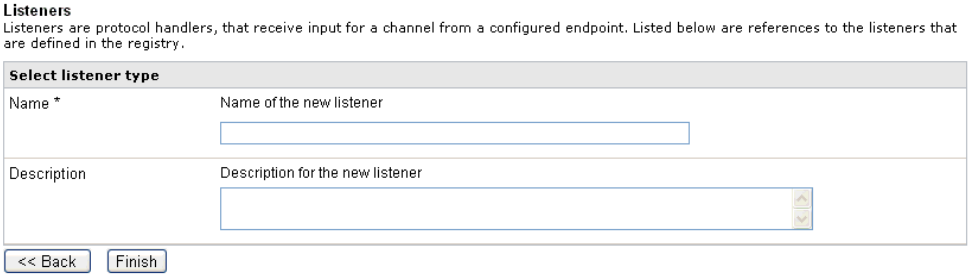
Note: The parameters prefixed with an asterisk (*) in the listener configuration pane are required.

7. Provide the appropriate values for the nHTTP listener parameters.

   For more information, see *Listener Configuration Parameters* on page 149.

8. Click *Next*.

You are returned to the Select listener type pane.



9. Enter a name for the nHTTP listener (required) and description (optional).
10. Click *Finish*.

    You can now use this listener as part of your channel configuration where the business logic will be applied to the received messages.

Reference: nHTTP Listener Configuration Parameters

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

Note: Properties that are common to all components are omitted from the following table. For a list of these properties refrer to *Configuring Common Parameters* on page 149 in the appendix of this manual.

Parameter	Description
IP Properties	
Port (required)	The TCP port for receipt of HTTP requests.
Local bind address	The local bind address for multi-homed hosts. This parameter value is usually not specified.
Persistence	If set to *true*, then the connection is maintained when the client requests to do so. If set to *false*, the connection is closed.

Parameter	Description
Maximum Connections	This defines the maximum number of simultaneous connections that are allowed. When this threshold is reached, new connections will not be accepted until current connections are closed and the total number of connections is below the limit. Leave this field blank (default) or set a value of zero to have no maximum limit of connections.
Persistence Timeout value in Minutes	The maximum length of time that a connection persists with no activity.
Set Response No Delay	If set to *true*, the Nagle Algorithm is disabled on the response. This will result in a faster line turnaround at the expense of an increased number of packets. *False* is the default.
Reuse Address	If set to *true*, then when a connection is closed, it immediately makes the address available, bypassing the TCP defaults. *False* is the default.
Allowable Clients	If supplied, then only messages from this list of fully qualified host names and/or IP addresses are accepted. Accepts comma-separated list or use the FILE() function.
Secure Connection	If set to *true*, then a connection using secure HTTP (HTTPS) is made. *False* is the default.
SSL Context Provider	The named iWay Security provider for SSL Context.
Redirect Insecure Requests	If set to *true*, a plain text request is redirected to the equivalent secure URL. Some HTTP clients can interpret the 301 response and retry the request over HTTPS. *False* is the default.
General Properties	

Parameter	Description
GET Handling	This determines how GET requests are handled. Options include: ❏ *docroot* - Attempts to serve a file from the document root directory. ❏ *error* - Returns an HTTP 405 Method Not Allowed. ❏ *event* - Generates an event message.
PUT and DELETE Handling	This determines how PUT and DELETE requests are handled. Options include: ❏ *unavailable* - Returns an HTTP 405 Method Not Allowed. ❏ *event* - Generates an event message.
OPTIONS Handling	his determines how OPTIONS requests are handled. Options include: ❏ *automatic* - Returns a response indicating which HTTP methods are handled. ❏ *event* - Generates an event message.
Document Root	The base directory from which all HTTP pages are served through GET if GET Handling is enabled for page access.
Default Page	The default page displayed if no page is identified in the incoming HTTP[s] request.
Default Text	The default text sent with 200 OK, which will take configured ContentType.

Parameter	Description
Response content type	This overrides the content type of the response. This overrides request content type value. Choose from among the following values: ❑ *application/EDI-X12* ❑ *application/EDIFACT* ❑ *application/XML* ❑ *text/html* ❑ *text/plain*

HTTP Response Code	Indicates the HTTP status code to send with the response. Usually this is left blank, which allows the channel to determine the appropriate response code.
	
	You can specify an iWay Functional Language (iFL) expression as a value for this parameter to be evaluated during the final emit process. If you specify an iFL expression, then ensure to include a leading backtick (`) character to prevent this expression from being evaluated until it is required. For example:
	
	`` `sreg(responsecode) ``
	
	If you decide to use a Special Register (SREG), then ensure to set it in the Message scope. The Message scope survives the process flow that elects to set the value. For example:
	
	`message:responsecode | 500 |`
	
	For more information on setting SREGs into scopes, see *SREG Service (com.ibi.agents.XDSREGAgent)* in the *iWay Service Manager Component Reference Guide*.
	
	The status code from nHTTP emits during the process flow are stored in a SREG called *httpstatus*, which is in the defined response SREG namespace of the emit service.

Parameter	Description
Authentication Scheme	The scheme to apply when authenticating HTTP requests. Select one of the following options from the drop-down list: ❑ *Digest Auth* ❑ *Basic Auth* ❑ *Negotiate* ❑ *None* (default)
Authentication Realm	If authentication is required, then this provides the name of the configured Realm provider to use.
Cookie Namespace	Special register namespace into which cookies from the incoming request will be saved. The default is to use the namespace *cookie*.
Request Header Namespace	The special register namespace to which HTTP headers from the incoming requests are saved. The Default option creates HDR type special registers without a namespace prefix.
Response Header Namespace	The special register namespace from which HTTP headers for the outgoing response are taken. The Default option sends HDR type registers with no namespace prefix. If none is selected, no special registers are sent as HTTP headers.
Response Main Part Header Namespace	The special register namespace from which MIME headers for the outgoing response are taken. Provide a prefix to control the response Main BodyPart headers in the presence of attachments. Selecting *none* means that no special registers are sent as MIME headers. An empty namespace prefix is treated as *none*.
Excluded Headers	A comma delimited list (case-insensitive) of headers that should not be sent with the response, even if they are found in the response header namespace.

Parameter	Description
Compress Response	If set, the HTTP request entity will be compressed using the selected encoding and the Content-Encoding header will be set accordingly. ❏ *deflate* ❏ *gzip* ❏ *none*
HTTP Session	
Session Support	If set to *true*, supports sessions by automatically creating a JSESSIONID cookie when absent, and providing a special register scope to hold session attributes. For example, a session attribute called a1 would be retrieved as special register called a1 in the register scope called "session". *False* is the default.
Maximum Sessions	Maximum number of active HTTP sessions. Beyond this threshold, the least recently used session will be deleted to make room for a newly created session. The value 0 means unlimited.
Session Max Inactive Interval	Maximum time interval that the listener will keep this session open between client accesses. The format is [xxh][xxm]xx[s], for example 1h30m is 90 minutes.
Cookie Path	Value of the Path attribute in the generated JSESSIONID cookie. The value / matches any path. This restricts the exchange of the cookie to the identified domain and optionally its subdomains. Most applications for iSM sessions will not need these settings
Cookie Domain	Value of the domain attribute in the generated JSESSIONID cookie. Leave empty to match the originating server.

Parameter	Description
Cookie Max Age	Value of the Max-Age attribute in the generated JSESSIONID cookie. This indicates the maximum lifetime of the cookie, represented as the number of seconds. Leave blank to omit the attribute which lets the user agent determine when the session is over.
Cookie Secure	Whether to add the Secure attribute in the generated JSESSIONID cookie. The attribute is automatically added when the listener is secure. This is meant to keep cookie communications limited to encrypted transmissions, directing browsers to use cookies only through secure and encrypted connections. Select one of the following options from the drop-down list: ❑ *automatic* (default) ❑ *false* ❑ *true*
Cookie HttpOnly	If set to *true*, directs browsers (or other clients) to use the JSESSIONID cookie through the HTTP protocol only (this includes HTTPS. HttpOnly is not the opposite of Secure). An HttpOnly cookie is not accessible through non-HTTP methods, such as calls using JavaScript (for example, referencing document.cookie), and therefore cannot be stolen easily through cross-site scripting. *False* is the default.
Denial of Service Protection	
Maximum Request Entry Size	On receipt of a request larger than the maximum, the listener will return HTTP 413 Request Entity Too Large; and close the connection. 0 means no maximum, empty defaults to 256KB.
Maximum Request Preamble Length	Maximum collective length for request preamble (in bytes). If the header length (key + value) plus the request exceeds this maximum, an error will be returned. Set the maximum to zero to avoid this test (no limit).

Parameter	Description
Events	
Channel Startup Flow	The name of the published process flow to run prior to starting the channel.
Channel Shutdown Flow	The name of the published process flow to run after the channel is shut down.

Note: The nHTTP listener supports streaming. Streaming is used for large documents or documents for which the application needs to split the input into sections under the same transaction. For more information on streaming and configuring streaming preparsers, see the *iWay Service Manager Component and Functional Language Reference Guide*.

Reference:

Special Registers for the nHTTP Listener

The following table lists and describes the special registers for the nHTTP listener.

Special Register	Level	Description
...	Header	Each header value from the message.
action	Document	The action field of the post.
docroot	Document	The defined docroot from configuration.
ip	Document	The IP of the sending system.
iwayconfig	System	The current active configuration name.
iwayhome	System	The base at which the server is loaded.
iwayworkdir	System	The path to base of the current configuration.
msgsize	Document	The physical length of the message payload.
name	System	The assigned name of the master (listener).
protocol	System	The protocol on which the message was received.
requestType	Header	The type of HTTP request (GET, POST, or HEAD).

Special Register	Level	Description
source	Document	The host name of the sending system.
url	Header	The full URL of the HTTP request (GET, POST, or HEAD).
tid	Document	Unique transaction ID.

Associating Session Information With an HTTP Interaction

In HTTP, a session is a sequence of network request-response transactions. A session may encompass a group of console screens or web interactions for a specific purpose. In transactional HTTP, such as REST or web services, a session may represent one or more request-response activities, such as sending a group of related shipping operations.

Applications can associate session information with an HTTP interaction. The session information is not actually carried between the client and the server. Rather, a token is assigned by iSM, which is carried between interactions. The token identifies the current session, much as a transaction ID represents the action of a single transaction within the session. By not carrying the session information between interactions, security is enhanced and network traffic is reduced.

Session information is carried in Special Registers (SREGs), which are created by the application and available in the later steps where they can be referenced and changed as required. The session SREGs are carried in a SREG scope called session. The session scope is not a namespace, although namespaces can be used within the session.

Registers in a specific scope can be set using the Special Register Setting Service (com.ibi.agents.XDSREGAgent). These registers can be referenced by the syntax *session:name*. For example, a database key carried in the dbkey SREG would be referenced as *session:dbkey*. The colon identifies the register as being in a named scope.

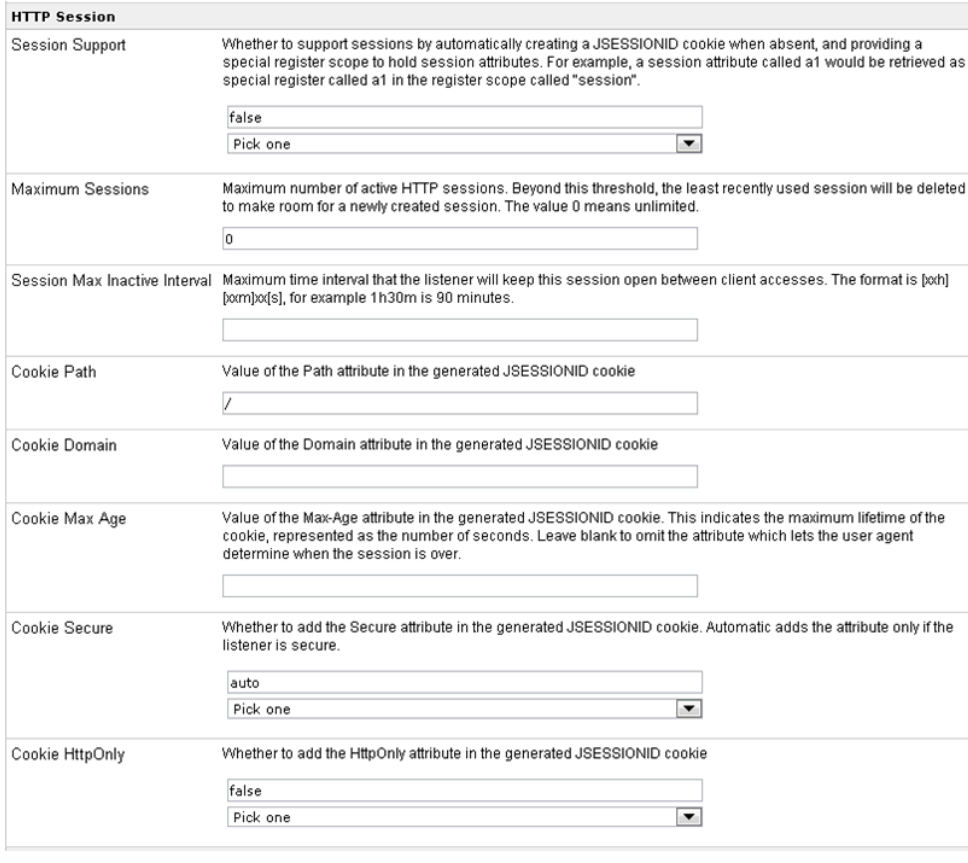
Note: Users are cautioned that scopes (denoted by the colon) are not namespaces. It is possible to use namespaces within the session scope (as it is in any scope), but usually in session scope namespaces do not add facility.

The session registers can be assigned to a type, such as USER, METRIC, or HDR. User registers can optionally be carried between channels within a transaction. All register attributes, including marshalling control and context recording can apply to session registers.

The session key is exchanged with the client by using the standard JSESSIONID cookie. As a result, management of the session involves dealing with the treatment of this cookie.

Configuring Sessions on an nHTTP Listener

Sessions are configured on the nHTTP listener in the HTTP Session section. Set the Session Support parameter to *true* to enable sessions.



HTTP Session	
Session Support	Whether to support sessions by automatically creating a JSESSIONID cookie when absent, and providing a special register scope to hold session attributes. For example, a session attribute called a1 would be retrieved as special register called a1 in the register scope called "session". false Pick one
Maximum Sessions	Maximum number of active HTTP sessions. Beyond this threshold, the least recently used session will be deleted to make room for a newly created session. The value 0 means unlimited. 0
Session Max Inactive Interval	Maximum time interval that the listener will keep this session open between client accesses. The format is [nxh][nxm]nx[s], for example 1h30m is 90 minutes.
Cookie Path	Value of the Path attribute in the generated JSESSIONID cookie /
Cookie Domain	Value of the Domain attribute in the generated JSESSIONID cookie
Cookie Max Age	Value of the Max-Age attribute in the generated JSESSIONID cookie. This indicates the maximum lifetime of the cookie, represented as the number of seconds. Leave blank to omit the attribute which lets the user agent determine when the session is over.
Cookie Secure	Whether to add the Secure attribute in the generated JSESSIONID cookie. Automatic adds the attribute only if the listener is secure. auto Pick one
Cookie HttpOnly	Whether to add the HttpOnly attribute in the generated JSESSIONID cookie false Pick one

The details of the session definition extend beyond the scope of this document. For more information, application designers should refer to other HTTP documentation.

The values specified for the Maximum Sessions and Session Max Inactive Interval parameters are important. Either of these settings can result in the loss of the session information for the session, even though a session token is received from the client. In such a case, the application must be designed to handle the loss of the session and session restart. For example, a shopping cart might be maintained in the session information. If the session expires, the shopping cart would be deleted and the application must reacquire the information that has been eliminated. While setting the listener to prevent expiration seems to avoid this situation, application designers must be aware that clients can be abandoned, resulting in the loss of the resources used to hold the session information in the server.

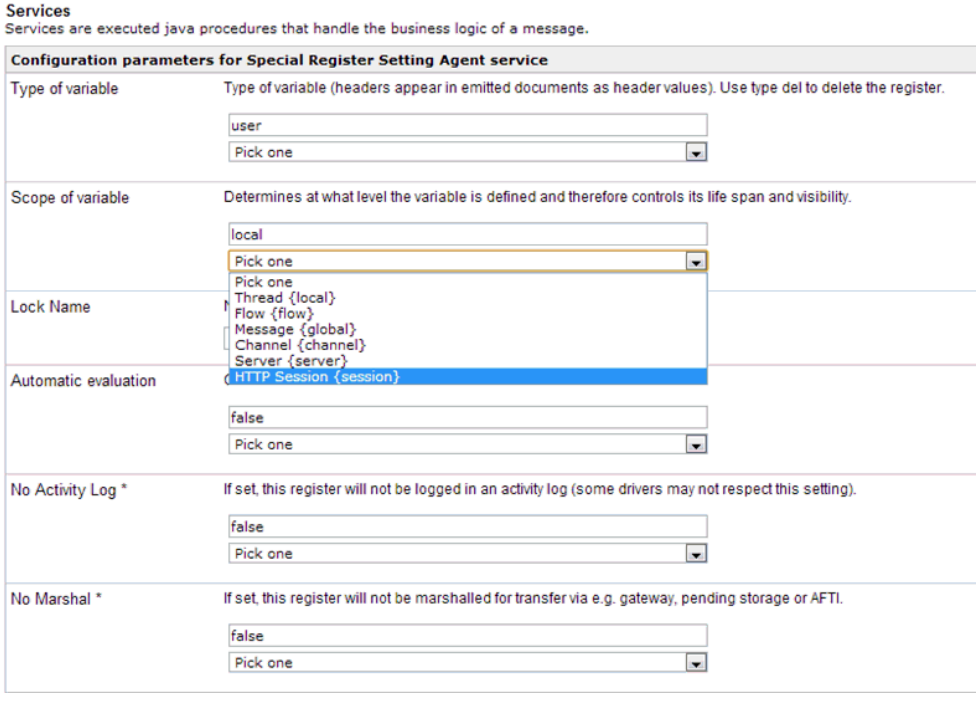
Because the session information is exchanged as a cookie (JSESSIONID), the cookie attributes can be applied. The cookie attributes are a cookie domain, a path, expiration time or maximum age, secure flag, and HttpOnly flag. Browsers will not return cookie attributes to the server. They will only send the name-value pair of the cookie. Cookie attributes are used by browsers to determine when to delete a cookie, block a cookie, or whether to send a cookie (name-value pair) to the server. The default entries for these attribute fields allow the cookie (session information) to be exchanged for all interactions.

Cookie parameters are defined in the HTTP Session section of the previous table.

Using Session Information in an Application

Setting the session information for an application is configured by using the Special Register Setting Service (com.ibi.agents.XDSREGAgent).

From the Scope of variable drop-down list, select *HTTP Session {session}*, as shown in the following image.



You must have the session support configured for the channel in order for the register to automatically appear in the next client interaction.

To reference the value of the register, you can use the following function from the iWay Functional Language (iFL):

```
_sreg('session:dbkey')
```

The HTTP Session Invalidator Service (com.ibi.agents.XDHttpSessionInvalidator) can be used in a process flow to invalidate the current session. This deletes all information in the session and prevents the session from being exchanged in subsequent client-server interactions. As a best practice, you can call for invalidating (deleting) the session once it is no longer required. An example would be a console logout or a determination of a catastrophic error situation that requires the user to restart an operation.

An application might keep an event count in the session, such that a count of 0 means this is the start of a session. Using 0 as the default value of an _sreg() lookup in a process flow test, the process flow can take whatever action is needed to begin the application session. Following the test, a Special Register Setting Service (com.ibi.agents.XDSREGAgent) might set the event count to the command shown in the following table:

eventcount	`_sreg('session:eventcount','0')+1`

HTTP Session Invalidator Service (com.ibi.agents.XDHttpSessionInvalidator)

This service is used to terminate the session. The following table lists and describes its parameter.

Parameter	Description
Expire Cookie	Determines how the termination is to be effected. If set to *false*, it removes the session information from the server. No further action is taken. If set to *true*, in addition to removing the session information, it sends an instruction to the client to delete the JSESSIONID cookie itself.

The service returns the input document on the success edge.

As a good practice, you can use this service when the application has completed the session, so as to reduce server resource requirements.

Configuring Emit Services

You can configure outbound processing of HTTP messages as a service that can be used within a process flow, which will become part of the route configuration or directly as a service assigned to a route. In this case, a business process can continue after an HTTP message has been sent out to the client. The following section describes how to configure an HTTP nonblocking emit service. For more information on configuring outlets and routes, see the *iWay Service Manager User's Guide*.

Procedure: How to Configure an HTTP Nonblocking Emit Service

To configure an HTTP nonblocking emit service:

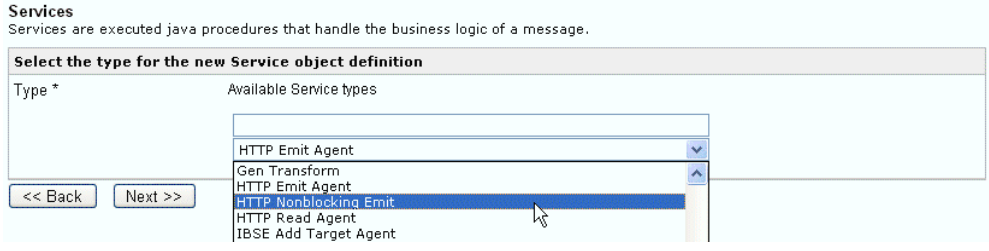
1. Click *Registry* in the top pane, and then click *Services* in the left pane.

   The Services pane opens.

   The table that is provided lists all the previously configured services and a brief description for each.

2. Click *Add*.

   The Select Service type pane opens.



3. Select *HTTP Nonblocking Emit* from the Type drop-down list.
4. Click *Next*.

   The configuration parameters pane for the HTTP nonblocking emit service opens.

5. Provide the appropriate values for the HTTP nonblocking emit service parameters.

   For more information, see *HTTP Nonblocking Emit Service (com.ibi.agents.XDNHttpEmitAgent)* on page 110.

6. Click *Next*.

   The name and description pane opens.

7. Enter a name for the service and description (optional).
8. Click *Finish*.

Reference: HTTP Nonblocking Emitter Configuration Parameters

The following table lists and describes parameters for the HTTP nonblocking emitter.

Parameter	Description
Configuration Parameters	

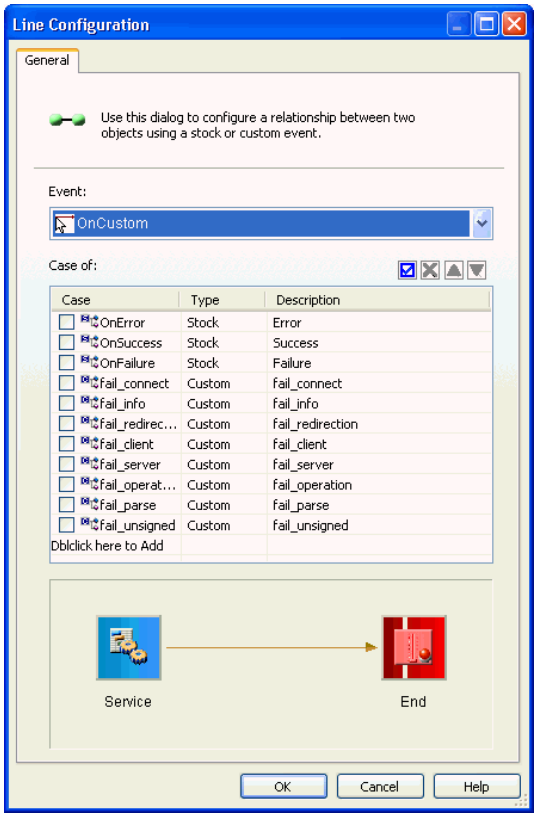
Parameter	Description
Destination (required)	The destination URL to post information that uses the following format: `http[s]://host[:port]/action`
HTTP Client Provider (required)	The HTTP client Provider that is used to manage connections for this emitter.
Cookie Store Name	Allows thread-specific management of cookies. If a name is not specified, a cookie store global to the HTTP Client provider will be used.
Action Method	Select one of the following supported methods from the drop-down list: ❑ *GET* - with encoded data on the URL. ❑ *DELETE* ❑ *HEAD* ❑ *POST* method with a Content-Length header. This value is selected by default. ❑ *PUT*
Request Content Type	The content type for the HTTP request sent by this emitter. Select a value from the drop-down list or provide your own. Available values from the drop-down list include: ❑ *application/EDI-X12* ❑ *application/EDIFACT* ❑ *application/XML* ❑ *text/html* ❑ *text/plain*
User ID	The user ID for Basic Authentication challenges.
Password	The password for Basic Authentication challenges.

Parameter	Description
Domain	The domain for NTLM authentication challenges. Note that to use NTLM, you must enable connection persistence.
Request Header Namespace	The special register namespace from which HTTP headers for the outgoing request will be taken. Choose *Default Namespace* to send HDR type registers with no namespace prefix, or supply a namespace prefix here. *None* means that no special registers will be sent as HTTP headers.
Request Main Part Header Namespace	The special register namespace from which MIME headers for the outgoing request will be taken. Provide a prefix to control the request Main BodyPart headers in the presence of attachments. Selecting *none* means that no special registers will be sent as MIME headers.
Response Header Namespace	The special register namespace into which HTTP headers from the incoming response will be saved. Choose *Default Namespace* to create special registers with no namespace prefix, or supply a namespace prefix here. *None* means that no special registers will be created. An empty namespace prefix will be treated as the default.
Excluded Headers	A comma delimited list (case-insensitive) of headers that should not be sent with the request, even if they are found in the request header namespace.
Ask for Compressed Response	If set to *true*, the request will set the accept-encoding to indicate that the client can accept a compressed response. If the response has a compressed content encoding, the client will automatically inflate the response.
Compress Request	If set to *true*, the request entities will be compressed using the selected encoding and the content-encoding header will be set accordingly.

Parameter	Description
Release Connection Immediately	If set to *true*, then the connection will be released to the connection pool immediately. This is the default. If set to *false*, then the connection identifier is stored in the *httpclient-key* special register and the HTTP Client Manager agent must be called later to release the connection explicitly. The main reason to retain the connection is to reuse it for streaming of other services.
Maximum HTTP Client Manager Delay	The maximum time for the HTTP Client Manager to handle a particular connection before it is automatically aborted. The format is [xxh][xxm]xx[s]. The default is 60 seconds.
Try Expect/Continue Handshake?	If set to *true*, the client will send the HTTP Expect: 100-continue header and await HTTP 100 response before sending the request body. By default, *false* is selected.
Chunk Encoded Request?	If set to *true*, the request entity will be sent with chunk encoding. By default, *false* is selected.
Maximum Request Size	The maximum size (after compression) of a request entity that can be sent with this emitter. A value of zero (0) means there is no maximum size limit and if no value is specified, the default value of 256KB is applied.
Maximum Response Size	The maximum size of a response entity that can be received by this emitter. A value of zero (0) means there is no maximum size limit and if no value is specified, the default value of 256KB is applied.
TCP	
Persistence	If set to *true*, the server is requested to maintain the connection.
Response Timeout value in seconds	The value in seconds to wait for a response before generating an error. The default value is 60 seconds.

Response Edges for nHTTPEmitAgent

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



The following line edges are provided:

- OnError
- OnSuccess
- OnFailure
- fail_connect
- fail_info
- fail_redirection
- fail_client

- fail_server
- fail_operation
- fail_parse
- fail_unsigned

nHTTP Samples

This section provides additional information about the nHTTP listener and includes samples you can use as a reference.

nHTTP Listener Event Schema

The nHTTP listener allows you to configure the handling of incoming HTTP requests. For example, the available options for the GET Handling parameter include docroot, error, and event. If you select event, an event document is created for the incoming request. This document can then be used in your process to determine an action for the request. The event document corresponds to the following structure:

```
<http user="auto" type="GET">
  <parms>
    <parm name="ibse-port">9000</parm>
    <parm name="Host">clientbox:10000</parm>
    <parm name="Connection">Keep-Alive</parm>
    <parm name="pdm">0</parm>
    <parm name="version">1.1</parm>
    <parm name="Accept-Language">en-AU</parm>
    <parm name="action">user.req?
user=999999999&account=1234567890123456&tranid=tid1234</parm>
    <parm name="Accept">text/html, application/xhtml+xml, */*</parm>
    <parm name="User-Agent">Mozilla/5.0 (compatible; MSIE 9.0; Windows NT
6.1; WOW64; Trident/5.0)</parm>
    <parm name="url">/user.req?
user=999999999&account=1234567890123456&tranid=tid1234</parm>
    <parm name="ip">127.0.0.1</parm>
    <parm name="source">hostname unknown</parm>
    <parm name="Accept-Encoding">gzip, deflate</parm>
    <parm name="reqType">GET</parm>
  </parms>
  <body />
  <url secure="false">
    <host>clientbox</host>
    <port>10000</port>
    <path>/user.req</path>
    <query>user=999999999&account=1234567890123456&tranid=tid1234</query>
    <queryparms>
      <queryparm name="user">99999999</queryparm>
      <queryparm name="account">1234567890123456</queryparm>
      <queryparm name="tranid">tid1234</queryparm>
    </queryparms>
    <incomingurl>    http:clientbox:1000/
user=999999999&account=1234567890123456&tranid=tid1234
   </incomingurl>
  </url>
  <version>1.1</version>
</http>
```

The following syntax is a sample document for the GET event:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<http user="unknown" type="GET">
<parms>
   <parm name="version">1.1</parm>
   <parm name="source">beck-xp.ibi.com</parm>
   <parm name="ua-cpu">x86</parm>
   <parm name="reqType">GET</parm>
   <parm name="accept-encoding">gzip, deflate</parm>
   <parm name="accept-language">en-us</parm>
   <parm name="connection">keep-alive</parm>
   <parm name="url">/TEST?A=B&C=D</parm>
   <parm name="user-agent">Mozilla/4.0 (compatible; Win 5.1</parm>
   <parm name="host">theservercom:7777</parm>
   <parm name="ip">172.19.22.60</parm>
</parms>
<body />
<url secure="true">
   <host>theserver.com</host>
   <port>7777</port>
   <path>/TEST</path>
   <query>A=B&C=D</query>
</url>
<version>1.1</version>
</http>
```

Supported nHTTP Requests

The following table lists the supported HTTP requests that can be processed by the nHTTP listener. Flow refers to the generation of the event signal document that can be processed within a process flow. A Reject action causes the client request to be rejected with an HTTP 405 Method Not Allowed response.

Request Type	Available Actions
DELETE	Flow or Reject
GET	From File, Flow, or Reject
HEAD	Like GET
OPTION	Reject
POST	Flow
PUT	Flow or Reject
TRACE	Echoes request as per RFC

Maximum Allowed Connections

The nHTTP listener has a parameter that can be configured to limit the number of simultaneous connections. However, this is not related to pool sizes or persistent connections. This parameter simply limits the number of clients that can connect to the server at once, persistent or not.

The listener tracks the current number of connections. When a new connection is accepted, the count is raised. When a connection closes, the count is lowered. Before accepting a new connection, the listener checks the current number of connections against the max connections parameter. If the current number is at the threshold, the new connection is not accepted and the following error message is written to the log:

```
ERROR (nh2) max connection threshold exceeded
```

Please note that the client has no knowledge of this back-end functionality. From its point of view, the connection might just be slow. The client will continue making connection attempts until it times out. As a result, it is normal to see multiple instances of this error message when simultaneous connections are over the limit. If the client does not time out and another connection closes, the new connection will be accepted and normal processing is continued

A blank value or 0 specified for the parameter indicates no limit.

SSL Host Verification

When SSL Host Verification is enabled, the client verifies that the certificate the server is presenting in the handshake matches the server hostname.

So, in the keystore of the server SSL context, there needs to be a key pair with CN == server hostname. If there is more than one private key in this keystore, you need to specify the server key alias to point to this key.

The client needs to add the <certificate of the CA that signed the server certificate> to its truststore. In the case of a self-signed certificate, this is the server certificate itself. The server never verifies the client hostname, even if SSL client authentication is enabled.

The following shows some of the information in a self-signed certificate with the CN in the subject Distinguished Name set to the host and port as required by the host name verifier.

```
Owner CN=myMachine.ibi.com:7777, OU=iWay, O=IBI, L=Cranston, ST=Rhode
Island, C=US
Issuer CN=myMachine.ibi.com:7777, OU=iWay, O=IBI, L=Cranston, ST=Rhode
Island, C=US
Serial number 46141cb7
Valid from Wed Apr 04 17:46:31 EDT 2007 until: Mon Oct 01 17:46:31 EDT
2007
Certificate fingerprints
MD5 61:02:2E:F2:D6:C2:0B:A8:AF:1F:6F:86:64:23:C9:17
SHA1 5F:7B:6C:A5:0E:FC:0C:33:F6:4C:4D:48:1B:C9:07:A4:DD:EF:54:62
```

Sonic Message Queuing

Sonic Message Queuing delivers a high performance and high reliability messaging system and provides certified Java Message Service Queue (JMSQ) implementation that includes both Queue Point-to-Point messaging and Publish/Subscribe messaging. It has an extended client/server feature that enables hierarchical security management. Sonic Message Queuing guarantees message persistence over the Internet. It contains message security, encryption, and certificate management.

The Sonic patent-pending Dynamic Routing Architecture (DRA) enables enterprises to participate in global e-Business exchanges through a single broker. When trading domains come online, Sonic dynamically discovers destinations and delivers messages between the servers on an optimized routing path. The Sonic architecture sets a foundation for high-throughput e-Business integration by robustly providing the following e-Business essentials: scalability, availability, and reliability.

The iWay Adapter for Sonic MQ provides bidirectional capability to and from Sonic destinations.

This topic describes how to:

- Listen on Sonic queues.
- Redirect output from a non-Sonic destination to a Sonic destination.

To redirect output to a destination, see *iWay Adapter for Sonic Emitter Functionality* on page 80.

Queuing Messages With Sonic

The iWay Adapter for Sonic MQ enables Service Manager to read messages arriving on a named queue or topic and route the messages to either another queue or topic, or to a non-Sonic destination. Similarly, messages received through any of the other iWay protocol adapter listeners can be directed to a Sonic queue. During the message flow, Service Manager can apply standard document analysis, validation, transformation, and business logic capabilities to the document.

The iWay Adapter for Sonic MQ provides bidirectional support for Sonic queues (a listener and an emitter). The adapter provides support for topics for a TopicSession in the Publish and Subscribe domain, as well as queue support (for a QueueSession in the PTP domain). Support is provided for persistent and non-persistent messages. The persistent option prevents messages from being lost in the event of a network or system failure. The iWay Adapter for Sonic MQ can operate over TCP, SSL, or HTTP(S).

The iWay Adapter for Sonic MQ includes high availability features. The adapter monitors the connection for an exception and re-establishes the connection if it was dropped. The adapter also supports load balancing. With load balancing enabled, a connect request can be redirected to another broker with a Sonic cluster. Similarly, failover capability is implemented. The adapter can redirect the message to another broker if it is unable to connect to the original broker.

Registering Sonic Client JAR Files

The following tables list the required .jar files for registration.

For instructions on registering JAR files, see *iWay Service Manager Protocol Guide*.

For Listeners:

Type	JAR Files
SSL	❑ Sonic_Client.jar Found at %Sonic_Home%/lib/. Add the Sonic_Client to the Path Settings PreClassPath section. ❑ Sonic_Crypto.jar ❑ jsafe.jar ❑ sonic_SSL.jar ❑ certj.jar ❑ sslj.jar Add the files to the Path Settings PreClassPath section.

Type	JAR Files
SSL Client Certificate	❑ Sonic_Client.jar Found at %Sonic_Home%/lib/. Add the Sonic_Client to the Path Settings PreClassPath section. ❑ Sonic_Crypto.jar ❑ jsafe.jar ❑ sonic_SSL.jar ❑ certj.jar ❑ sslj.jar

For Emitters:

Type	JAR Files
TCP or HTTP	Sonic_Client.jar Found at %Sonic_Home%/lib/. Add the Sonic_Client to the Path Settings PreClassPath section.
SSL	❑ Sonic_Client.jar ❑ jsafe.jar ❑ sonic_SSL.jar ❑ certj.jar ❑ sslj.jar Found at %Sonic_Home%/lib. Add the JAR files to the Path Settings PreClassPath section.

For instructions on registering JAR files, see *iWay Service Manager Protocol Guide*.

iWay Adapter for Sonic MQ Listener Capability

Sonic supports standard Internet protocols including Secure Socket Layer (SSL), HTTP, encryption, TCP, and security. iWay Service Manager can listen on Sonic queues using TCP, SSL, HTTP, and HTTPS protocols. Each protocol requires setup specific to the unique protocol. This topic details configuration information for each individual protocol.

Procedure: How to Configure Reconnect Support

1. Click the *Server* link at the top left of the Service Manager console.
2. Under Settings in the left pane, click *General Settings*.
3. In the Retry Interval entry field, enter a value to enable the listeners to retry the connection if it fails for external causes.

   The default value is 120 seconds.

   The retry interval is a global property that controls the configured listeners. It marks the frequency in seconds in which Service Manager attempts to reconnect to a broker if the connection is lost or cannot be established.

Configuring a Sonic Listener Using TCP or HTTP

The Transport Control Protocol (TCP) is a connection-oriented protocol that establishes a connection with another host before it sends data. Before a connection is started between two hosts, control messages called a handshake are sent out to initiate the connection. TCP is the default protocol of a Sonic broker installation. Client applications that are Internet-enabled generally use TCP/IP protocols.

Hypertext Transfer Protocol (HTTP), the underlying protocol used by the World Wide Web, defines how messages are formatted and transmitted and the actions web servers and browsers must take in response to various commands.

HTTP operates over TCP connections. After a successful connection, the client transmits a request message to the server, which returns a reply message.

By server design or by company security policy, proxy servers and firewalls frequently allow only HTTP-based traffic to pass through. You can establish a direct connection to a Sonic broker using HTTP tunneling as the protocol. However, because the HTTP tunneling protocol is significantly slower than TCP or SSL, this option is recommended only when TCP and SSL protocols are not available.

You require JAR files supplied by Sonic to emit messages to a Sonic message queue.

Sonic Listener Properties for TCP or HTTP

The following table lists and describes the Sonic listener properties for TCP or HTTP. For instructions on creating a listener, see *Configuring Listeners* on page 153.

Property Name	Property Description

Form of Input	`TOPIC`
	Is for a TopicSession in the Publish and Subscribe domain.
	`QUEUE`
	Is for a QueueSession in the PTP domain.

Receiver Name (required)	The name of the queue on which input documents will be received for processing by Service Manager.
Default Output Form	The topic or queue target.
Standard Reply To	The queue or topic to which the system writes response messages. The property may be overwritten by the configuration properties.
Broker URL (required)	The URL (address) used by the listener to connect to the Sonic broker. The format of the URL is Protocol://host:port. If Service Manager is listening on a Sonic broker configured to listen on TCP, the format of the URL is tcp://host:port. The default TCP port on which Sonic listens is 2506. This value is configured in the Sonic broker.ini file. When Service Manager listens to Sonic, the URL is in the form of http://host:port where the HTTP port is defined in the Sonic broker.ini file. The Sonic listener supports failover if unable to reach a particular broker. You must provide a comma-separated list of broker URLs. The client attempts to connect to brokers in the list, for example, tcp://host:port, tcp://host:port.
Pending Queue	If system resources are down or temporarily unavailable, requests that cannot be completed can be placed into the pending system for later execution. When listening on a TOPIC, the iWay adapter uses a Sonic queue for pending.

Property Name	Property Description
Form of Acknowledgment	Acknowledgment mode support: the session acknowledgment mode is either Transactional (to send and receive a series of messages and then explicitly commit or roll back the group) or in one of the following acknowledgment modes: *Client Provides* - An explicit acknowledge on a message acknowledges the receipt of all messages that were produced and consumed by the session that gives the acknowledgment. When a session is forced to recover, it restarts with its first unacknowledged message. *Duplicates Permitted* - The session lazily acknowledges the delivery of messages to consumers, possibly allowing some duplicate messages after a system outage. *Auto Acknowledge* - The session automatically acknowledges the client receipt of a message by successfully returning from a call to receive (synchronous mode) or when the session message listener successfully returns (asynchronous mode). The last message can be delivered again.
Send Persistently	The support for persistent and non-persistent messages. In the event of a network or system failure, the persistent option prevents messages from being lost. In the event of a broker or Service Manager failure, non-persistent messages are volatile. Persistent messages are saved to disk.

Property Name	Property Description
Durable Topic Subscriber	The support for durable topic subscribers. In Publish and Subscribe messaging, messages are not stored for later delivery unless the client establishes a subscription name that is associated with its user identity on the message broker. *False (Non-Durable Subscribers)* - A client uses a topic subscriber to receive messages that were published to a topic. A regular topic subscription is not durable. Messages are delivered when active but never retained. *True (Durable Subscribers)* - If a client must receive all messages published on a topic including those published while the subscriber is inactive, it uses a durable TopicSubscriber. The message broker retains a record of this durable subscription and ensures that all messages from the topic publishers are retained until they are either acknowledged by this durable subscriber or expire. Sessions with durable subscribers must always provide the same client identifier. In addition, each client must specify a name that uniquely identifies (within the client identifier) each durable subscription it creates. Within a session, durable topic subscribers must be uniquely named.
Message Priority	The Sonic broker attempts to deliver messages with a higher priority ahead of messages with lower values. Priority is suggested to the JMSQ provider and can only order the delivery of undelivered messages. JMSQ defines ten levels of message priority with values 0 through 9, where 0 is the lowest, and 9 is the highest. Zero through four are considered normal settings and five through nine, expedited. The Message Priority field is the default priority value set in the JMSQ header.
Set Reply Correlation Id	If set to a value, that value is used as the correlation ID of the response.
Load Balance	If set to *true*, the load balancing option is enabled on the Service Manager listener. With load balancing enabled, a connect request can be redirected to another broker within a Sonic cluster, if load balancing was not disabled on the broker side.

Property Name	Property Description
Duplicates Detect	You can configure the broker to commit transactions to index a universally unique, 64-character identifier (UUID) supplied by the sender. The sender then uses a commit method that commits the transacted messages, unless a previously sent transaction identifier is still unexpired. Otherwise, the method forces a rollback of the transaction. For more information, see the INDEXED_TXN_ server properties in the installation section of the *Sonic MQ V5 Configuration and Administration Guide*.
User	A valid user ID defined to Sonic. You can use the Sonic Explorer Users option (found under message broker) to display users defined to Sonic. The user ID is required only when Sonic security is enabled.
Password	A valid user ID and password combination defined to Sonic. Use the Sonic Explorer Users option (found under message broker) to display users defined to Sonic.
Client ID	The broker retains messages for durable subscriber, using the userName and clientID of the connection plus the subscriptionName to index the subscription. The Client ID is a unique identifier that can prevent conflicts for durable subscriptions when many clients use the same user name and the same subscription name.
Connection ID	Identifies the connection. When combined with Client ID, must be unique at the broker.
Subscription Name	A subscription name always includes the name of the topic. To distinguish different message selectors used in subscriptions, you can include a string that helps identify the message selector. For example, you can use a subscription named Atlas_priority4 for a subscription to the Atlas topic with selector JMSPriority=4. This construct enables you to create many durable subscriptions that are easily understood and nonconflicting.

Property Name	Property Description
Message Selector	In PTP domains, all message selection takes place on the server. However, in Pub/Sub domains, all messages for a subscribed topic are delivered by default to the subscriber, and then, the filter is applied to decide what is consumed. Message selectors can consist of: ❏ Literals and indefinites ❏ Operators and expressions ❏ Comparison tests ❏ Parentheses ❏ White space
Output Message Type	Select *Bytes*, *Text*, or *Dynamic*.
Prefetch Count	Only applies to queue messages. Does not apply to topic messages. Prefetch count is the number of messages buffered locally. A worker (thread) prefetches messages to this limit. Injudicious use of this configuration can result in unbalanced thread use and appear to cause worker starvation. For example, if a prefetch count is set to 3 and two workers are defined and three messages are in the queue, the first worker prefetches all three messages. The second worker finds the queue empty and awaits the arrival of yet another message. Thus, it appears that the first worker processes three messages while the second does no work. To achieve balance, set the prefetch count to 1 and the prefetch threshold to 0. Use of prefetch count and threshold can result in improved performance through overall reduction in network traffic to the broker.
Prefetch Threshold	Fewer messages cause prefetch to be initiated. For additional information, see the description of the Prefetch Count property.
Duration	The maximum time a document can remain in the retry pending queue.
Retry	The interval between retrying pending requests.

Property Name	Property Description
Preserve Undelivered	The preserve messages that are not retrieved to a dead letter queue.
Notify Undelivered	Notify the broker administrator if undelivered messages are preserved.
Sequential	Determines the order in which the connection to the brokers is made when multiple brokers are listed. If this property is enabled, connection occurs sequentially. If disabled, connection occurs in a random manner.
Fault Tolerant	Determines whether the listener is connecting to the Sonic Broker as a fault tolerant client or not. The default value is false, that is, the listener is not connecting as a fault tolerant client. This feature is supported only when the Sonic Broker is installed with a Sonic Fault Tolerance license code.
Reconnect Fault Timeout	Sets the interval, in seconds, of how long to wait before attempting to reconnect to the broker if the session is lost.
Initial Connect Timeout	Sets the interval, in seconds, of how long to wait before attempting to connect to the broker, or to another broker on the list, if connection fails when the listener is first started.
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 *condense* to remove extra whitespace in pretty printed documents and for compatibility with earlier versions.
Accepts non-XML (flat) only	If set to *true*, then non-XML input is expected. If enabled, XML input still can be passed to the listener. Preparsers do not run.
Optimize Favoring	Use this option to customize listener performance. For smaller transactions, select *performance*. For large payloads that could monopolize the amount of memory used by Service Manager, select *memory*.

Property Name	Property Description
Multithreading	The number of worker threads. (Equivalent to the number of requests that 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 being processed. The total throughput of a system can depend on the number of threads operating. Default: 1 Max value: 99
Execution Time Limit	The maximum time a request can take to complete. A request that takes longer to complete terminates. Prevents runaway requests. **Note:** The kill interval property checks for runaway requests that have exceeded their maximum life. Default is 60 (seconds). Duration is xxhxxmxxs, for example, 1h2m3s = one hour, two minutes, and three seconds.
Polling Interval	Indicates frequency in seconds that the listener returns control to Service Manager to determine if a stop listener was requested. Listener is constantly connected to the queue to retrieve incoming messages. The default value is 2.0.
Default Java File Encoding	The default encoding if incoming message is not self-declaring (that is, XML).
Agent Precedence	Sets the order by which Service Manager selects agents. Service Manager selects the agent(s) 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>. <document> overrides <listener> is the default value.
Always reply to listener default	If set to *true*, the default reply definition is used in addition to defined destinations.

Property Name	Property Description
Error Documents treated normally	If set to *true*, error documents are processed by configured preemitters.
Listener is Transaction Manager	If set to *true*, agents run in a local transaction. Agents can roll back uncompleted transactions.
Record in Activity Log(s)	If set to *true*, activity on this channel will be recorded in the activity logs, else the activity will not be recorded.

Note: The Sonic listener supports streaming. Streaming is used for large documents or documents for which the application needs to split the input into sections under the same transaction. For more information on streaming and configuring streaming preparsers, see the *iWay Service Manager Component and Functional Language Reference Guide*.

Sonic TCP Listener Configuration Example

You would like to listen on a Sonic MQ queue named SampleQ2, hosted by a broker on machine iwxfoc, which listens on default Sonic MQ port 2056. You would also like to output messages to the queue SampleQ4 on the same broker. The broker does not have security enabled.

To configure the components needed to support this scenario, you need to configure the following in Service Manager:

Note: For information on the steps required to accomplish these tasks, see the *iWay Service Manager User's Guide*.

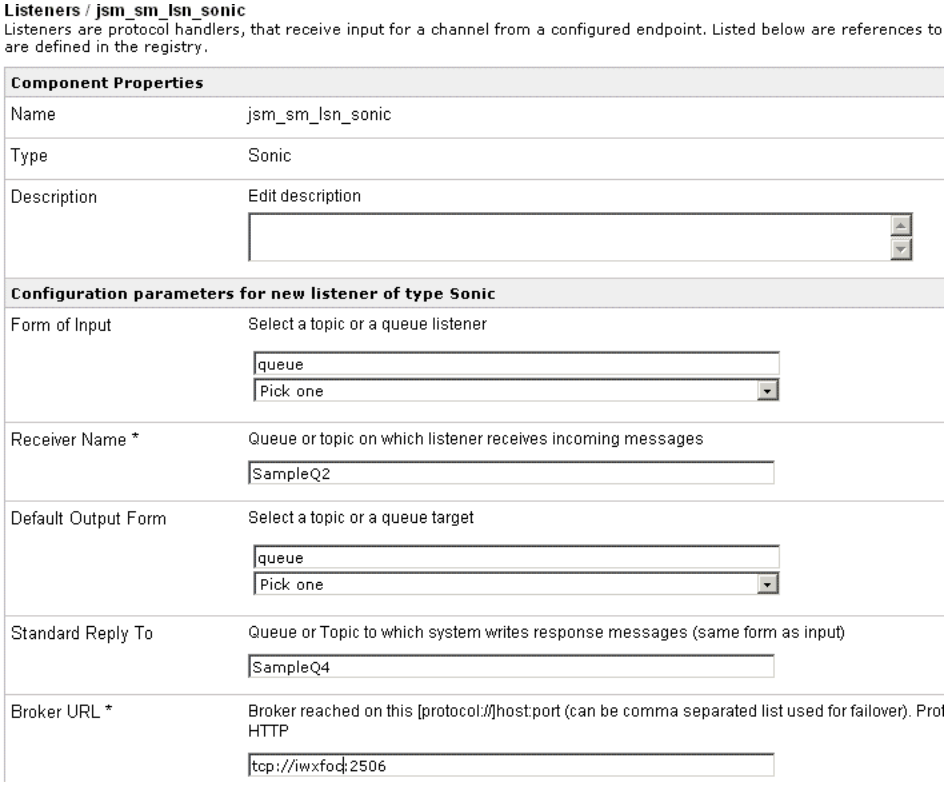
- ❑ Copy the Default Channel provided with SM installation and rename it as sonic_lsn_channel.
- ❑ Create a listener of type Sonic, and call it jsm_sm_lsn_sonic. The listener will connect to Sonic MQ on iwxfoc machine (tcp://iwxfoc:2506). It will get messages from SampleQ2 queue and output them into a queue named SampleQ4 at the same broker.
- ❑ Add the listener to an Inlet.
- ❑ Add the Inlet to the Channel.

The other parts of the channel will stay the same as default channel. After building and deploying the sonic_lsn_channel, you can see messages being transferred from SampleQ2 to SampleQ4 on the iwxfoc Sonic MQ broker.

Specify the listener values as follows. Leave default values for all other fields.

- **Type:** Sonic
- **Form of Input:** queue
- **Receiver Name:** SampleQ2
- **Standard Reply To:** SampleQ4
- **Broker URL:** tcp://iwxfoc :2506

The following image shows the Sonic listener configuration pane.



Configuring a Sonic Listener Using SSL

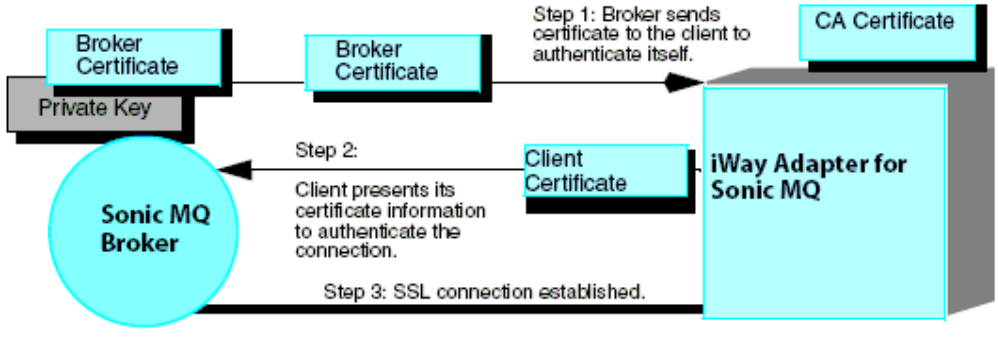
Secure Sockets Layer (SSL) provides authentication and encryption techniques that require the broker to set the appropriate properties and certificates on a security-enabled database. The client also must have the appropriate libraries and properties to call an SSL connection. A complete implementation sample is provided in the Protocols section of the *Sonic MQ V5 Configuration and Administration Guide*.

Note: You must have SSL libraries installed from RSA with the appropriate Sonic license or from supported third-parties such as Institute for Applied Information Processing and Communications (IAIK) or Java Secure Socket Extension (JSSE).

The manner in which you configure SSL for your application depends on whether you implement client authentication with a user name and password or with a client certificate.

When the Sonic broker is configured with the SSL property SSL_CLIENT_AUTHENTICATION=FALSE in the broker.ini configuration file, no client certificate is required, and the client is authenticated with a user name and password. The client reads the user name and password and then passes them to the broker.

The following diagram shows Step 1, in which the broker sends the certificate to the client to authenticate itself; Step 2, in which the client presents its certificate information to authenticate the connection; and Step 3, in which an SSL connection is established.



Set Broker Properties

```
SSL_CLIENT_AUTHENTICATION=TRUE
SSL_CERTIFICATE_CHAIN=certs/server.p7c
SSL_PRIVATE_KEY=certs/serverKey.pkcs8
SSL_PRIVATE_KEY_PASSWORD=password
SSL_CA_CERTIFICATE_DIR=certs/CA
```

JVM Options

```
SSL_CA_CERTIFICATES_DIR=install_dir/ce
rts/CA
SSL_CERTIFICATE_CHAIN=certs/client.p7c
SSL_PRIVATE_KEY=certs/clientKey.pkcs8
SSL_PRIVATE_KEY_PASSWORD=password
```

You must add JAR files, supplied by Sonic, to emit messages to a Sonic message queue using the SSL protocol. For more information, see *Registering Sonic Client JAR Files* on page 49.

Setting Java System Properties for Sonic SSL

For instructions on setting Java system properties, see the *iWay Service Manager Protocol Guide*.

In the Property field, type *SSL_CA_CERTIFICATES_DIR*. In the Value field, type the location of the CA certificate to be used by Service Manager.

The default certificate shipped with the Sonic product is located in:

```
%SONIC_HOME%\certs\ca
```

Sonic Listener Properties for SSL

The following table lists and describes the Sonic listener properties for SSL. For instructions on creating a listener, see *Configuring Listeners* on page 153.

Property Name	Property Description

Form of Input	`TOPIC`
	Is for a TopicSession in the Publish and Subscribe domain.
	`QUEUE`
	Is for a QueueSession in the PTP domain.

Receiver Name (required)	The name of the queue on which input documents will be received for processing by Service Manager.
Default Output Form	The topic or queue target.
Standard Reply To	The queue or topic to which the system writes response messages. The property may be overridden by the configuration properties.
Broker URL (required)	The URL (address) used by the listener to connect to the Sonic broker. The format of the URL is Protocol://host:port. If Service Manager is listening on a Sonic broker configured to listen on SSL, the URL would be in the format of ssl://host:port. This value is configured in the Sonic broker.ini file. The Sonic listener supports failover if unable to reach a particular broker. You must provide a comma-separated list of broker URLs. The client attempts to connect to brokers in this list, for example, ssl://host:port.
Pending Queue	Requests that cannot be completed because system resources are down or temporarily unavailable can be placed into the pending system for later execution. When listening on a TOPIC, the adapter uses a Sonic queue for pending.

Property Name	Property Description
Form of Acknowledgment	Acknowledgment mode support: The session acknowledgment mode is either Transactional (to send and receive a series of messages and then explicitly commit or rollback the group) or one of the available acknowledgment modes. *Client Provides* - An explicit acknowledge on a message acknowledges the receipt of all messages that are produced and consumed by the session that gives the acknowledgment. When a session is forced to recover, it restarts with its first unacknowledged message. *Duplicates Permitted* - The session lazily acknowledges the delivery of messages to consumers, possibly allowing some duplicate messages after a system outage. *Auto Acknowledge* - The session automatically acknowledges the client receipt of a message by successfully returning from a call to receive it (synchronous mode) or when the session message listener successfully returns the receipt (asynchronous mode). The last message can be delivered again.
Send Persistently	Support for persistent and non-persistent messages. In the event of a network or system failure, the persistent option prevents messages from being lost. In the event of a broker or iWay Service Manager failure, non-persistent messages are volatile. Persistent messages are saved to disk.

Property Name	Property Description
Durable Topic Subscriber	Support for durable topic subscribers. In Publish and Subscribe messaging, messages are not stored for later delivery unless the client establishes a subscription name that is associated with its user identity on the message broker. *False (Non-durable subscribers)* - A client uses a topic subscriber to receive messages that have been published to a topic. A regular topic subscription is not durable. Messages are delivered when active but never retained. *True (Durable subscribers)* - If a client must receive all the messages published on a topic including those published while the subscriber is inactive, it uses a durable TopicSubscriber. The message broker retains a record of this durable subscription and ensures that all messages from the topic publishers are retained until they are either acknowledged by this durable subscriber or expire. Sessions with durable subscribers must always provide the same client identifier. In addition, each client must specify a name that uniquely identifies (within the client identifier) each durable subscription it creates. Within a session, durable topic subscribers must be uniquely named.
Message Priority	The Sonic broker attempts to deliver messages with a higher priority ahead of messages with lower values. Priority is suggested to the JMSQ provider, and only orders delivery of undelivered messages. JMSQ defines ten levels of message priority with values 0 through 9, where 0 is the lowest and 9 is the highest. Zero through four are considered normal settings and five through nine, expedited. The Message Priority field is the default priority value set in the JMSQ header.
Set Reply Correlation Id	If set to a value, that value used as the correlation ID of the response.
Load Balance	Select *true* to enable the load balancing option on the Service Manager listener. With load balancing enabled, a connect request can be redirected to another broker within a Sonic cluster, if load balancing was not disabled on the broker side.

Property Name	Property Description
Duplicates Detect	You can configure the broker to commit transactions such that they index a universally unique, 64-character identifier (UUID) supplied by the sender. The sender then uses a commit method that commits the transacted messages, unless a previously sent transaction identifier is still unexpired. Otherwise, the method forces a rollback of the transaction. For more information, see the INDEXED_TXN_ server properties in the installation section of the *Sonic MQ V5 Configuration and Administration Guide*.
User	A valid user ID defined to Sonic. You can use the Sonic Explorer Users option (found under message broker) to display users defined to Sonic. The user ID is required when listening on a Sonic queue with client authentication.
Password	A valid user ID and password combination defined to Sonic. You can use the Sonic Explorer Users option (found under message broker) to display users defined to Sonic. The password is required when listening on a Sonic queue with client authentication.
Client ID	The broker retains messages for durable subscriber, using the userName and clientID of the connection, plus the subscriptionName to index the subscription. The Client ID is a unique identifier that can prevent conflicts for durable subscriptions when many clients are using the same user name and the same subscription name.
Connection ID	The identifies the connection. When combined with Client ID, must be unique at the broker.
Subscription Name	A subscription name always includes the name of the topic. To distinguish different message selectors used in subscriptions, you can include a string that helps identify the message selector. For example, you can use a subscription named Atlas_priority4 for a subscription to the Atlas topic with selector JMSPriority=4. This construct enables you to create many durable subscriptions that are easily understood and non-conflicting.

Property Name	Property Description
Message Selector	In PTP domains, all message selection takes place on the server. However, in Pub/Sub domains, all messages for a subscribed topic are by default delivered to the subscriber and then, the filter is applied to decide what is consumed. Message selectors can consist of: ❏ Literals and indefinites ❏ Operators and expressions ❏ Comparison tests ❏ Parentheses ❏ White space
Output Message Type	Select *Bytes*, *Text*, or *Dynamic*.
Prefetch Count	Only applies to queue messages; does not apply to topic messages. Prefetch count is the number of messages buffered locally. A worker (thread) prefetches messages to this limit. Injudicious use of this configuration can result in unbalanced thread use and appear to cause worker starvation. For example, if a prefetch count is set to 3 and two workers are defined and three messages are in the queue, the first worker prefetches all three messages. The second worker finds the queue empty and awaits the arrival of yet another message. Thus, it appears that the first worker processes three messages while the second does no work. To achieve balance, set the prefetch count to 1 and the prefetch threshold to 0. Use of prefetch count and threshold can result in improved performance through overall reduction in network traffic to the broker.
Prefetch Threshold	Fewer messages cause prefetch to be initiated. For additional information, see the description of the Prefetch Count property.
Duration	The maximum time that a document can remain in the retry pending queue.
Retry	The interval between retrying pending requests.

Property Name	Property Description
Preserve Undelivered	The preserve messages that are not retrieved to a dead letter queue.
Notify Undelivered	Notifies the broker administrator if undelivered are preserved.
Sequential	Determines the order in which the connection to the brokers is made when multiple brokers are listed. If this property is enabled, connection occurs sequentially. If disabled, connection occurs in a random manner.
Fault Tolerant	Determines whether the listener is connecting to the Sonic Broker as a fault tolerant client or not. The default value is false, that is, the listener is not connecting as a fault tolerant client. This feature is supported only when the Sonic Broker is installed with a Sonic Fault Tolerance license code.
Reconnect Fault Timeout	Sets the interval, in seconds, of how long to wait before attempting to reconnect to the broker if the session is lost.
Initial Connect Timeout	Sets the interval, in seconds, of how long to wait before attempting to connect to the broker, or to another broker on the list, if connection fails when the listener is first started.
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 *condense* to remove extra whitespace in pretty printed documents and for compatibility with earlier versions.
Accepts non-XML (flat) only	If set to *true*, non-XML input is expected. If enabled, XML input still can be passed to the listener. Preparsers do not run.
Optimize Favoring	Use this option to customize listener performance. For smaller transactions, select *performance*. For large payloads that could monopolize the amount of memory used by Service Manager, select *memory*.

Property Name	Property Description
Multithreading	The number of worker threads. (Equivalent to the number of requests that 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 being processed. The total throughput of a system is affected by the number of threads operating. Default: 1 Max value: 99
Execution Time Limit	The maximum time a request can take to complete. A request that takes longer to complete terminates. Prevents runaway requests. **Note:** The kill interval property checks for runaway requests that exceed their maximum life. Default is 60 (seconds). Duration is xxhxxmxxs, for example, 1h2m3s = one hour, two minutes, and three seconds.
Polling Interval	Indicates frequency in seconds that the listener returns control to Service Manager to determine if a stop listener was requested. The listener is constantly connected to the queue to retrieve incoming messages. The default value is 2.0.
Default Java File Encoding	The default encoding if incoming message is not self-declaring (that is, XML).
Agent Precedence	Sets the order by which Service Manager selects agents. Service Manager usually looks for a document entry in the configuration dictionary 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>. <document> overrides <listener> is the default value.
Always reply to listener default	If set to *true*, the default reply definition is used in addition to defined destinations.
Error Documents treated normally	If set to *true*, error documents are processed by configured preemitters.

Property Name	Property Description
Listener is Transaction Manager	If set to *true*, agents run in a local transaction. Agents can roll back uncompleted transactions.
Record in Activity Log(s)	If set to *true*, activity on this channel will be recorded in the activity logs, else the activity will not be recorded.

Note: The Sonic listener supports streaming. Streaming is used for large documents or documents for which the application needs to split the input into sections under the same transaction. For more information on streaming and configuring streaming preparsers, see the *iWay Service Manager Component and Functional Language Reference Guide*.

Reference: **Sonic Listener Special Registers**

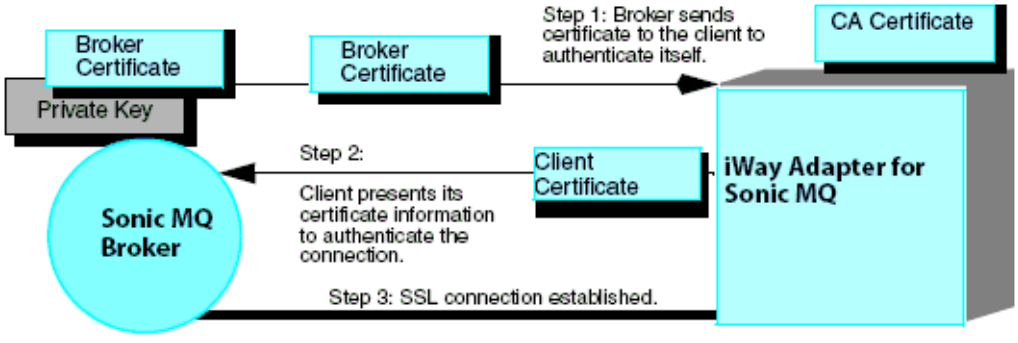
The following table lists and describes the special registers (SREGs) available on the Sonic listener.

Name	Level	Type	Description
correlid	Document	String	The correlation ID.
destination	Document	String	The default destination for reply (from message).
iwayconfig	System	String	The current active configuration name.
msgid	Document	String	The message ID.
msgsize	Document	Integer	The physical length of the message payload.
name	System	String	The assigned name of the master (listener).
priority	Document	String	The priority of this message.
protocol	System	String	The protocol on which this message was received.
source	Document	String	The queue or topic on which message was received.
tid	Document	String	Unique transaction ID.

Configuring a Sonic Listener Using SSL Client Certificate

When you configure the Sonic broker with the SSL property SSL_CLIENT_AUTHENTICATION=TRUE, you can achieve authentication by exchanging digital certificates between Service Manager and the Sonic broker. In addition to this mutual authentication, Service Manager can present a user name and password to the broker at run time (optional).

The following diagram shows Step 1, in which the broker sends the certificate to the client to authenticate itself; Step 2, in which the client presents its certificate information to authenticate the connection; and Step 3, in which an SSL connection is established.



Set Broker Properties

```
SSL_CLIENT_AUTHENTICATION=TRUE
SSL_CERTIFICATE_CHAIN=certs/server.p7c
SSL_PRIVATE_KEY=certs/serverKey.pkcs8
SSL_PRIVATE_KEY_PASSWORD=password
SSL_CA_CERTIFICATE_DIR=certs/CA
```

JVM Options

```
SSL_CA_CERTIFICATES_DIR=install_dir/certs/CA
SSL_CERTIFICATE_CHAIN=certs/client.p7c
SSL_PRIVATE_KEY=certs/clientKey.pkcs8
SSL_PRIVATE_KEY_PASSWORD=password
```

JAR files supplied by Sonic are required to emit messages to a Sonic message queue using the SSL protocol. For more information, see *Registering Sonic Client JAR Files* on page 49.

Reference: JVM Options for SSL With Certificates

The following table lists and describes the properties required to configure the appropriate environment for communication to Sonic using SSL with certificates.

For instructions on setting Java system properties, see the *iWay Service Manager Protocol Guide*. Complete the Property and Value fields as described here.

Property	Description
SSL_CA_CERTIFICATES_DIR SSL_CA_CERTIFICATES_DIR={certs/ca|path} SSL_CA_CERTIFICATES_DIR_n={certs/ca|path}	Specifies the location of the Certificate Authority (CA) certificate. The path must be fully qualified or relative to the Service Manager installation directory. The default directory is certs/ca.
SSL_CERTIFICATE_CHAIN SSL_CERTIFICATE_CHAIN={certs/server.p7c|path} SSL_CERTIFICATE_CHAIN_n={certs/server.p7c|path}	Specifies the location of the file containing the client keystore certificate chain for SSL. The path must be fully qualified or relative to the Service Manager installation directory. The default directory is certs/server.p7c.
SSL_CERTIFICATE_CHAIN_FORM SSL_CERTIFICATE_CHAIN_FORM={LIST|PKCS12|PKCS7} SSL_CERTIFICATE_CHAIN_FORM_n={LIST|PKCS12|PKCS7}	Specifies the format of the file containing the certificate chain. PKCS7 is the default value, a comma-delimited list of path names that point to files containing each individual certificate in the chain.
SSL_PRIVATE_KEY SSL_PRIVATE_KEY={serverKey.pkcs8|path} SSL_PRIVATE_KEY_n={serverKey.pkcs8|path}	Provides the location of the file containing the client encrypted private key for SSL. This path must be fully qualified or relative to the Service Manager installation directory. The default value is serverKey.pkcs8.

Property	Description

`SSL_PRIVATE_KEY_PASSWORD`
`SSL_PRIVATE_KEY_PASSWORD={password|password}`
`SSL_PRIVATE_KEY_PASSWORD_n={password|password}`
	Provides the password that encrypts the private key for SSL.
	The default value is password.

Sonic Listener Properties for SSL With Client Certificate

The following table lists and describes the Sonic listener properties for SSL with client certificate. For instructions on creating a listener, see *Configuring Listeners* on page 153.

Property Name	Property Description

Form of Input	`TOPIC`
	Is for a TopicSession in the Publish and Subscribe domain.
	`QUEUE`
	Is for a QueueSession in the PTP domain.

Receiver Name (required)	The name of the queue on which input documents will be received for processing by Service Manager.
Default Output Form	The topic or queue target.
Standard Reply To	The queue or topic to which the system writes response messages. The property may be overwritten by the configuration properties.

Property Name	Property Description
Broker URL (required)	The URL (address) used by the listener to connect to the Sonic broker. The format of the URL is Protocol://host:port. If Service Manager is listening on a Sonic broker configured to listen on TCP, the format of the URL is tcp://host:port. The default TCP port on which Sonic listens is 2506. This value is configured in the Sonic broker.ini file. When iWay Service Manager listens to Sonic, the URL is in the form of http://host:port where the HTTP port is defined in the Sonic broker.ini file. The Sonic listener supports failover if unable to reach a particular broker. You must provide a comma-separated list of broker URLs. The client attempts to connect to brokers in this list, for example, tcp://host:port, tcp://host:port.
Pending Queue	If the system resources are down or temporarily unavailable, requests that cannot be completed can be placed into the pending system for later execution. When listening on a TOPIC, the iWay adapter uses a Sonic queue for pending.
Form of Acknowledgment	Acknowledgment mode support: The session acknowledgment mode is either Transacted (to send and receive a series of messages and then explicitly commit or roll back the group) or in one of the available acknowledgment modes. *Client Acknowledge* - An explicit acknowledge on a message acknowledges the receipt of all messages that are produced and consumed by the session that gives the acknowledgment. When a session is forced to recover, it restarts with its first unacknowledged message. *Duplicates OK Acknowledge* - The session lazily acknowledges the delivery of messages to consumers, possibly allowing some duplicate messages after a system outage. *Auto Acknowledge* - The session automatically acknowledges the client receipt of a message by successfully returning from a call to receive (synchronous mode) or when the session message listener successfully returns (asynchronous mode). The last message can be delivered again.

Property Name	Property Description
Send Persistently	The support for persistent and non-persistent messages. The persistent option prevents messages from being lost in the event of a network or system failure. In the event of a broker or Service Manager failure, non-persistent messages are volatile. Persistent messages are saved to disk.
Durable Topic Subscriber	Support for durable topic subscribers. In Publish and Subscribe messaging, messages are not stored for later delivery unless the client establishes a subscription name that is associated with its user identity on the message broker. **False (Non-Durable Subscribers).** A client uses a topic subscriber to receive messages that are published to a topic. A regular topic subscription is not durable. Messages are delivered when active but never retained. **True (Durable Subscribers).** If a client must receive all the messages published on a topic including those published while the subscriber is inactive, it uses a durable TopicSubscriber. The message broker retains a record of this durable subscription and ensures that all messages from the topic publishers are retained until they are either acknowledged by this durable subscriber or expire. Sessions with durable subscribers must always provide the same client identifier. In addition, each client must specify a name that uniquely identifies (within the client identifier) each durable subscription it creates. Within a session, durable topic subscribers must be uniquely named.
Message Priority	The Sonic broker attempts to deliver messages with a higher priority ahead of messages with lower values. Priority is suggested to the JMSQ provider and can only order the delivery of undelivered messages. JMSQ defines ten levels of message priority with values 0 through 9, where 0 is the lowest and 9 is the highest. Zero through four are considered normal settings and five through nine, expedited. The Message Priority field is the default priority value set in the JMSQ header.

Property Name	Property Description
Set Reply Correlation Id	If set to a value, that value is used as the correlation ID of the response.
Load Balance	Select the load balancing option on the Service Manager listener for load balancing. With load balancing enabled, a connect request can be redirected to another broker within a Sonic cluster, if load balancing was not disabled on the broker side.
Duplicates Detect	You can configure the broker to commit transactions to index a universally unique, 64-character identifier (UUID) supplied by the sender. The sender then uses a commit method that commits the transacted messages, unless a previously sent transaction identifier is still unexpired. Otherwise, the method forces a rollback of the transaction. For more information, see the INDEXED_TXN_ server properties in the installation section of the *Sonic MQ V5 Configuration and Administration Guide*.
User	A valid user ID defined to Sonic. You can use the Sonic Explorer Users option (under message broker) to display users defined to Sonic. User ID only required if Sonic security is enabled.
Password	A valid user ID and password combination defined to Sonic. Use the Sonic Explorer Users option (under message broker) to display users defined to Sonic.
Client ID	The broker retains messages for durable subscriber, using the userName and clientID of the connection plus the subscriptionName to index the subscription. The Client ID is a unique identifier that can prevent conflicts for durable subscriptions when many clients are using the same user name and the same subscription name.
Connection ID	Identifies the connection. When combined with Client ID, must be unique at the broker.

Property Name	Property Description
Subscription Name	A subscription name always includes the name of the topic. To distinguish between different message selectors used in subscriptions, you can include a string that helps identify the message selector. For example, you can use a subscription named Atlas_priority4 for a subscription to the Atlas topic with selector JMSPriority=4. This construct lets you create many durable subscriptions that are easily understood and nonconflicting.
Message Selector	In PTP domains, all message selection takes place on the server. However, in Pub/Sub domains, all messages for a subscribed topic are delivered by default to the subscriber, and then, the filter is applied to decide what is consumed. Message selectors can consist of: ❑ Literals and indefinites ❑ Operators and expressions ❑ Comparison tests ❑ Parentheses ❑ White space
Output Message Type	Select *Bytes*, *Text Message*, or *Dynamic*.

Property Name	Property Description
Prefetch Count	Only applies to queue messages; does not apply to topic messages. Prefetch count is the number of messages buffered locally. Any worker (thread) prefetches messages to this limit. Injudicious use of this configuration can result in unbalanced thread use and appear to cause worker starvation. For example, if a prefetch count is set to 3 and two workers are defined and three messages are in the queue, the first worker prefetches all three messages. The second worker finds the queue empty and awaits the arrival of yet another message. Thus, it appears that the first worker processes three messages while the second does no work. To achieve balance, set the prefetch count to 1 and the prefetch threshold to 0. Use of prefetch count and threshold can result in improved performance through overall reduction in network traffic to the broker.
Prefetch Threshold	Fewer messages cause prefetch to be initiated. For additional information, see the Prefetch Count property.
Duration	The maximum time that a document can remain in the retry pending queue.
Retry	The interval between retrying pending requests.
Preserve Undelivered	Preserves messages that are not retrieved to a dead letter queue.
Notify Undelivered	Notifies the broker administrator if undelivered are preserved.
Sequential	Determines the order in which the connection to the brokers is made when multiple brokers are listed. If this property is enabled, connection occurs sequentially. If disabled, connection occurs in a random manner.
Fault Tolerant	Determines whether the listener is connecting to the Sonic Broker as a fault tolerant client or not. The default value is false, that is, the listener is not connecting as a fault tolerant client. This feature is supported only when the Sonic Broker is installed with a Sonic Fault Tolerance license code.

Property Name	Property Description
Reconnect Fault Timeout	Sets the interval, in seconds, of how long to wait before attempting to reconnect to the broker if the session is lost.
Initial Connect Timeout	Sets the interval, in seconds, of how long to wait before attempting to connect to the broker, or to another broker on the list, if connection fails when the listener is first started.
Accepts non-XML (flat) only	The select if non-XML input is expected. If enabled, XML input still can be passed to the listener. Preparsers do not run.
Optimize Favoring	Use this option to customize listener performance. For smaller transactions, select *performance*. For large payloads that could monopolize the amount of memory used by Service Manager, select *memory*.
Multithreading	The number of worker threads. (Equivalent to the number of requests that 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 being processed. The total throughput of a system can depend on the number of threads operating. Default: 1 Max value: 99
Execution Time Limit	The maximum time a request can take to complete. A request that takes longer to complete terminates. Prevents runaway requests. **Note:** The kill interval property checks for runaway requests that exceed their maximum life. Default is 60 (seconds). Duration is xxhxxmxxs, for example, 1h2m3s = one hour, two minutes, and three seconds.
Polling Interval	Indicates frequency in seconds that the listener returns control to Service Manager to determine if a stop listener was requested. The listener is constantly connected to the queue to retrieve incoming messages. The default value is 2.0.
Default Java File Encoding	The default encoding if incoming message is not self-declaring (that is, XML).

Property Name	Property Description
Agent Precedence	Sets the order by which Service Manager selects agents. 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>. <document> overrides <listener> is the default value.
Always reply to listener default	If set to *true*, the default reply definition is used in addition to defined destinations.
Error Documents treated normally	If set to *true*, error documents are processed by configured preemitters.
Listener is Transaction Manager	If set to *true*, agents run in a local transaction. Agents can roll back uncompleted transactions.
Initialization Agent	The name (parameters) of the processing module called at listener start up.

Note: The Sonic listener supports streaming. Streaming is used for large documents or documents for which the application needs to split the input into sections under the same transaction. For more information on streaming and configuring streaming preparsers, see the *iWay Service Manager Component and Functional Language Reference Guide*.

Configuring a Sonic Listener Using HTTPS

To configure the Sonic listener for SSL (the user ID or certificate depending on how the broker is configured), modify the destination address to specify an HTTPS URL.

The format of the destination property is queue@url. When emitting to Sonic queues over HTTPS, the format is queue@https://host:port. The HTTP port is configured in the Sonic broker.ini file. To send a message to iWay.Reply hosted on a Sonic broker on your localhost, the URL is iWay.Reply@https://localhost:2507.

For instructions on creating a listener, see *Configuring Listeners* on page 153.

iWay Adapter for Sonic Emitter Functionality

To route an output document or error message to a protocol other than that of the listener, you must configure an emitter. An emitter sends documents outbound either on the same protocol that the document arrived on or across protocols. You can return a processed document to one or more alternate destinations. By default, an output document is returned using the same protocol on which it was received. For example, an application may send input over TCP but want to route the output to a Sonic queue. In this case, you would configure an emitter.

Note: Configuring a Sonic emitter is not required if the emitter protocol used in the outlet of the channel is the same as the listener protocol used in the inlet of the channel. For more information on inlets and outlets, see the *iWay Service Manager User's Guide*.

Sonic supports standard Internet protocols including Secure Socket Layer (SSL), HTTP, encryption, TCP, and security. Service Manager can send responses to Sonic queues through TCP, SSL, HTTP, and HTTPS protocols. Each protocol requires specific set up. This topic describes configuration information for each protocol.

Configuring a Sonic Emitter Using TCP or HTTP

The Transport Control Protocol (TCP) is a connection-oriented protocol that establishes a connection with another host before it sends its data. Before a connection is started between two hosts, control messages called a handshake are sent to initiate the connection. TCP is the default protocol of a Sonic broker installation. Client applications that are Internet-enabled generally use TCP/IP protocols.

Hypertext Transfer Protocol (HTTP), the underlying protocol used by the World Wide Web, defines how messages are formatted and transmitted, and the actions web servers and browsers must take in response to various commands.

HTTP operates over TCP connections. After a successful connection, the client transmits a request message to the server, which sends a reply message back.

By server design or by company security policy, proxy servers and firewalls frequently allow only HTTP-based traffic to pass through. You can establish a direct connection to a Sonic broker using HTTP tunneling as the protocol. However, because the HTTP tunneling protocol is significantly slower than TCP or SSL, this option is only recommended when TCP and SSL protocols are not available.

To emit messages to Sonic, JAR files supplied by Sonic are required. For more information, see *Registering Sonic Client JAR Files* on page 49.

Sonic Emitter Properties for TCP or HTTP

The following table lists and describes the Sonic emitter properties.

A Sonic message consists of a JMSQ header, body, and user-defined properties.

Property Name	Property Description
Destination (required)	The destination to which the message is delivered. The format of the destination property is queue@url. When emitting to a Sonic broker configured to listen over TCP, the format is queue@tcp://host:port. The default TCP port on which Sonic listens is 2506. This value is configured in the Sonic broker.ini file. To send a message to iWay.Reply hosted on a Sonic broker on your localhost, the URL is iWay.Reply@tcp://localhost:2506. For HTTP, the URL is in the format queue@http://host:port. The default HTTP port on which Sonic listens is 2580. To send a message to iWay.Reply hosted on a Sonic broker on your localhost, the URL is iWay.Reply@http://localhost:2580

Form of Output	`TOPIC`
	
	Is for a TopicSession in the Publish and Subscribe domain.
	
	`QUEUE`
	
	Is for a QueueSession in the PTP domain.

User	A valid user ID defined to Sonic. You can use the Sonic Explorer Users option (under the message broker) to display users defined to Sonic. User ID is only required if Sonic security is enabled.
Password	A valid user ID and password combination defined to Sonic. You can use the Sonic Explorer Users option (under message broker) to display users defined to Sonic.
Send Persistently	Support for persistent and non-persistent messages. In the event of a network or system failure, the persistent option prevents messages from being lost. In the event of a broker or Service Manager failure, non-persistent messages are volatile. Persistent messages are saved to disk.

Property Name	Property Description
Load Balance	Select *true* to enable the load balancing option on the Service Manager listener. With load balancing enabled, a connect request can be redirected to another broker within a Sonic cluster, if load balancing was not disabled on the broker side.
Duplicates Detect	You can configure the broker to commit transactions that index a universally unique, 64-character identifier (UUID) supplied by the sender. The sender then uses a commit method that commits the transacted messages, unless a previously sent transaction identifier is still unexpired. Otherwise, the method forces a rollback of the transaction. For more information, see the INDEXED_TXN_ server properties in the *Installation* section of the *Sonic MQ V5 Configuration and Administration Guide*.
Set Reply Correlation Id	If set to a value, that value is used as the correlation ID of the response.
Duration	The maximum time that a document can remain in the retry pending queue.
Message Priority	The Sonic broker attempts to deliver messages with a higher priority ahead of messages with lower values. Priority is suggested to the JMSQ provider and can only order delivery of undelivered messages. JMSQ defines ten levels of message priority with values 0 through 9, where 0 is the lowest and 9 is the highest. Zero through four are considered normal settings and five through nine, expedited. The Message Priority field is the default priority value set in the JMSQ header.
Output Message Type	Select *Bytes*, *Text*, or *Dynamic*.
Preserve Undelivered	Preserves messages that are not retrieved to a dead letter queue.
Notify Undelivered	Notifies the broker administrator if undelivered are preserved.

Property Name	Property Description
Sequential	Determines the order in which the connection to the brokers is made when multiple brokers are listed. If this property is enabled, connection occurs sequentially. If disabled, connection occurs in a random manner.
Fault Tolerant	Determines whether the listener is connecting to the Sonic Broker as a fault tolerant client or not. The default value is false, that is, the listener is not connecting as a fault tolerant client. This feature is supported only when the Sonic Broker is installed with a Sonic Fault Tolerance license code.
Reconnect Fault Timeout	Sets the interval, in seconds, of how long to wait before attempting to reconnect to the broker if the session is lost.
Initial Connect Timeout	Sets the interval, in seconds, of how long to wait before attempting to connect to the broker, or to another broker on the list, if connection fails when the listener is first started.

Sonic Emitter Configuration Example

You would like to read a file from its local file system and place it as a message on iWay.Reply queue defined at the Sonic broker, which listens on default Sonic MQ port on a machine named iwxfoc. The broker does not have security enabled.

To configure the components needed to support this scenario, you would need to do the following in Service Manager:

Note: For information on the steps required to accomplish these tasks, see the *iWay Service Manager User's Guide*.

- ❑ Copy the default file1 channel that is provided with the iSM installation and rename it to sonic_emit_channel.
- ❑ Create an emitter, named sonic_emit, that will place a message obtained from a file listener on iWay.Reply queue defined at the Sonic broker on iwxfoc.

  The sonic_emit_channel will use the default file1 inlet, which is actually a file listener also named file1. In addition, it will use an outlet that contains the sonic_emit emitter.
- ❑ Assign the emitter as an outlet of the channel.

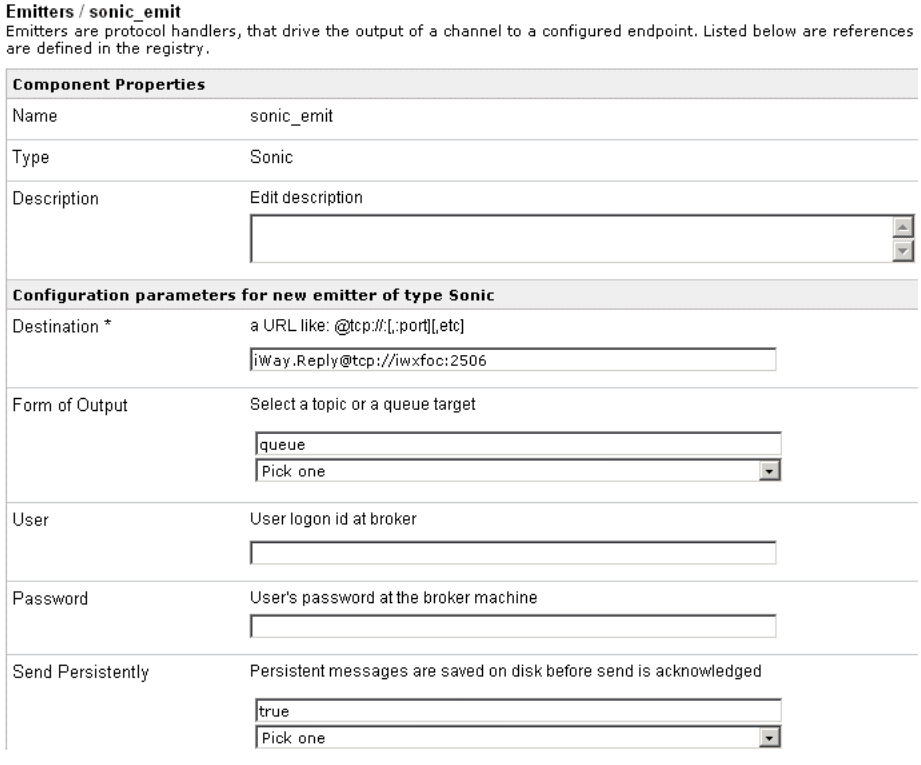
After building and deploying channel you can see the file dropped into file1 pickup folder being placed on iWay.Reply queue on iwxfoc Sonic MQ broker.

Specify the emitter properties as follows:

- **Destination:** iWay.Reply@tcp://iwxfoc:2506
- **Form of Output:** Queue

  Leave default values for the remaining properties.

The following image shows the Sonic emitter configuration pane.



Configuring a Sonic Emitter Using SSL

SSL provides authentication and encryption techniques that require that the broker configure appropriate properties and certificates on a security-enabled database. Also, the client must have the appropriate libraries and properties to call its side of an SSL connection. A complete implementation sample is provided in the protocol section of the *Sonic MQ V5 Configuration and Administration Guide*. You must have SSL libraries installed from RSA, with the appropriate Sonic license or from supported third-parties, such as the Institute for Applied Information Processing and Communications (IAIK) or Java Secure Socket Extension (JSSE).

The manner in which you configure SSL for your application depends on whether you implement client authentication with a user name and password or with a client certificate. When the Sonic broker is configured with the SSL property SSL_CLIENT_AUTHENTICATION=FALSE in the broker.ini configuration file, no client certificate is required, and the client is authenticated with a user name and password. The client reads the user name and password, then passes them to the broker.

When the Sonic broker is configured with the SSL property SSL_CLIENT_AUTHENTICATION=TRUE, authentication is achieved through the exchange of Digital Certificates between Service Manager and the Sonic broker. In addition to this mutual authentication, Service Manager can present a user name and password to the broker at run time (optional).

JAR files supplied by Sonic are required to emit messages to a Sonic message queue using the SSL protocol. For more information, see *Registering Sonic Client JAR Files* on page 49.

Reference: JVM Options for SSL With Certificates

The following table lists and describes the properties required to configure the appropriate environment for communication to Sonic using SSL with certificates. For instructions on configuring JVM options, see *Setting Java System Properties for Sonic SSL* on page 61.

Property	Description

`SSL_CA_CERTIFICATES_DIR`
`SSL_CA_CERTIFICATES_DIR={certs/ca|path}`
`SSL_CA_CERTIFICATES_DIR_n={certs/ca|path}`
	Specifies the location of the Certificate Authority (CA) certificate. Path must be fully qualified or relative to the Service Manager installation directory.
	
	The default directory is certs/ca.

Property	Description
SSL_CERTIFICATE_CHAIN SSL_CERTIFICATE_CHAIN={certs/server.p7c|path} SSL_CERTIFICATE_CHAIN_n={certs/server.p7c|path}	Specifies the location of the file containing the client keystore certificate chain for SSL. Path must be fully qualified or relative to the Service Manager installation directory. The default directory is certs/server.p7c.
SSL_CERTIFICATE_CHAIN_FORM SSL_CERTIFICATE_CHAIN_FORM={LIST|PKCS12|PKCS7} SSL_CERTIFICATE_CHAIN_FORM_n={LIST|PKCS12|PKCS7}	Specifies the format of the file containing the certificate chain. PKCS7 is the default value, a comma-delimited list of path names that point to files containing each individual certificate in the chain.
SSL_PRIVATE_KEY SSL_PRIVATE_KEY={serverKey.pkcs8|path} SSL_PRIVATE_KEY_n={serverKey.pkcs8|path}	Provides the location of the file containing the client encrypted private key for SSL. Path must be fully qualified or relative to the Service Manager installation directory. The default value is serverKey.pkcs8.
SSL_PRIVATE_KEY_PASSWORD SSL_PRIVATE_KEY_PASSWORD={password|password} SSL_PRIVATE_KEY_PASSWORD_n={password|password}	Provides the password that encrypts the private key for SSL. The default value is password.

Sonic Emitter Properties for SSL With Certificate

The following table lists and describes the Sonic emitter properties for SSL with certificate.

A Sonic message consists of a JMSQ header, body, and user-defined properties.

HTTPS is similar to HTTP except that data is transmitted over a Secure Socket Layer (SSL) instead of a normal socket connection. Web servers listen for HTTP requests on one port while another listens for HTTPS requests.

Property Name	Property Description
Destination (required)	The destination to which the message is delivered. The format of the destination property is queue@url. When emitting to Sonic queues over SSL, the format is queue@ssl://host:port. The SSL port is configured in the Sonic broker.ini file. To send a message to iWay.Reply hosted on a Sonic broker on your localhost, the URL is iWay.Reply@ssl://localhost:2507.

Form of Output	`TOPIC`
	Is for a TopicSession in the Publish and Subscribe domain.
	`QUEUE`
	Is for a QueueSession in the PTP domain.

User	You must specify the common name AUTHENTICATED from the certificate as the user name. The password is not required because you authenticate the client using the client certificate in this example. When Service Manager emits to a Sonic queue it can specify a user name and password to the broker (optional). The broker authenticates this user name and password if they are specified, but otherwise uses the information in the client certificate to identify the user.
Password	This property is optional and is only required if a user ID is specified.
Send Persistently	The support for persistent and non-persistent messages. In the event of a network or system failure, the persistent option prevents messages from being lost. In the event of a broker or Service Manager failure, non-persistent messages are volatile. Persistent messages are saved to disk.
Load Balance	If set to *true*, then this enables the load balancing option on the Service Manager listener. With load balancing enabled, a connect request can be redirected to another broker within a Sonic cluster, if load balancing was not disabled on the broker side.

Property Name	Property Description
Duplicates Detect	You can configure the broker to commit transactions that index a universally unique, 64-character identifier (UUID) supplied by the sender. The sender then uses a commit method that commits the transacted messages, unless a previously sent transaction identifier is still unexpired. Otherwise, the method forces a rollback of the transaction. For more information, see the INDEXED_TXN_ server properties in the Installation section of the *Sonic MQ V5 Configuration and Administration Guide*.
Set Reply Correlation Id	If set to a value, that value is used as the correlation ID of the response.
Duration	The maximum time that a document can remain in the retry pending queue.
Message Priority	The Sonic broker attempts to deliver messages with a higher priority ahead of messages with lower values. Priority is suggested to the JMSQ provider and can only order delivery of undelivered messages. JMSQ defines ten levels of message priority with values 0 through 9, where 0 is the lowest and 9 is the highest. Zero through four are considered normal settings and five through nine, expedited. The Message Priority field is the default priority value set in the JMSQ header.
Output Message Type	Select *Bytes*, *Text*, or *Dynamic*.
Preserve Undelivered	Preserves messages that are not retrieved to a dead letter queue.
Notify Undelivered	Notifies the broker administrator if undelivered are preserved.
Sequential	Determines the order in which the connection to the brokers is made when multiple brokers are listed. If this property is enabled, connection occurs sequentially. If disabled, connection occurs in a random manner.

Property Name	Property Description
Fault Tolerant	Determines whether the listener is connecting to the Sonic Broker as a fault tolerant client or not. The default value is false, that is, the listener is not connecting as a fault tolerant client. This feature is supported only when the Sonic Broker is installed with a Sonic Fault Tolerance license code.
Reconnect Fault Timeout	Sets the interval, in seconds, of how long to wait before attempting to reconnect to the broker if the session is lost.
Initial Connect Timeout	Sets the interval, in seconds, of how long to wait before attempting to connect to the broker, or to another broker on the list, if connection fails when the listener is first started.

Procedure: How to Configure Sonic Emitter Properties Using HTTPS

To configure Sonic emitter properties using HTTPS:

1. Refer to the values defined in *Sonic Emitter Properties for SSL With Certificate* on page 86 (user ID or certificate, depending on how the broker is configured) when creating the emitter.
2. Modify the destination address to specify an HTTPS URL.

   The format of the destination property is queue@url. When emitting to Sonic queues over HTTPS, the format is queue@https://host:port. The HTTP port is configured in the Sonic broker.ini file. To send a message to iWay.Reply hosted on a Sonic broker on your localhost, the URL is iWay.Reply@https://localhost:2507.

Sonic Message Queuing Troubleshooting

When using a Sonic listener, if the system cannot find the queue specified, you receive the following message:

```
Cannot emit reply to .XDSonicEmit<no dead letter path>: XD[FAIL] in emit
() error create queue.
```

Using Sonic Explorer, verify that the queue exists, that the name is spelled correctly, and that the name is in the correct case, as the Sonic listener is case-sensitive.

Chapter 3

Configuring iWay HTTP Services (Adapters)

iWay Service Manager includes many predefined services that you can use to enhance the business logic of a message. You can add these services to simple or complex business logic configurations using the iWay Service Manager Administration Console.

For reference purposes, this section lists and describes all the services that are supplied with iWay Service Manager.

In this chapter:

HTTP Services Configuration Overview

Services are executable Java procedures that are used to handle the business logic of a message in iWay Service Manager (iSM).

A service is the business layer that incorporates the logic for encapsulating the business process which interacts with other distributed component services to provide transactions for business state information. This business layer incorporates the application business logic. In an iSM environment, business logic consists of one or more services acting on an input document. Services can be stacked, such that the output of one service is passed to the next service, or multiple services can be executed in parallel.

Services are written in standard Java language and can make use of any available Java libraries or services. iWay Software supplies a comprehensive set of predefined services with iSM that you can use as part of your business logic.

HTTP Services

The following section provides a comprehensive reference for all the predefined HTTP services that are supplied with iWay Service Manager.

Service Name
Add Attachment From File Service (com.ibi.agents.XDAddAttachmentFromFileAgent) on page 92
Add Attachment Service (com.ibi.agents.XDAddAttachmentAgent) on page 93
Attachment Operations Service (com.ibi.agents.XDAttachOps) on page 94.
Attachment to Document Service (com.ibi.agents.XDAttachmentToDocAgent) on page 95
Cross-Origin Resource Sharing Service (com.ibi.agents.XDCorsAgent) on page 96
Document to Attachment Service (com.ibi.agents.XDAttachmentFromDocAgent) on page 102
HTTP Cookie Agent Service (com.ibi.agents.XDCookieAgent) on page 103
HTTP Emit Service (com.ibi.agents.XDHTTPEmitAgent) on page 104
HTTP Nonblocking Emit Service (com.ibi.agents.XDNHttpEmitAgent) on page 110
HTTP Read Agent (com.ibi.agents.XDHTTPReadAgent) on page 114
HTTP ReST Routing Service (XDReSTRouteAgent and XDReSTRouteReviewer) on page 117
HTTP Session Invalidator Service (com.ibi.agents.XDHttpSessionInvalidator) on page 38
OAuth 1.0 Authentication Service on page 126
OAuth 2.0 Authentication Service on page 131
WS HTTP Client Agent (com.ibi.agents.XDWSHttpClientAgent) on page 139

Add Attachment From File Service (com.ibi.agents.XDAddAttachmentFromFileAgent)

Syntax:

```
com.ibi.agents.XDAddAttachmentFromFileAgent
```

Description:

This service adds a new attachment with the contents determined by the contents of a file. This service is convenient to provide binary data without first going through a Java character set encoding. The attachment headers are specified by the special registers of type HDR in the MIME Header Namespace.

There are also four parameters to specify the most common MIME headers. When used, these parameters override special registers of the same name. Notice the value of the Content-ID header is taken as is, so the value must contain the surrounding angle brackets. For example, a valid value for Content-ID might be <cid>. This service follows the OnSuccess edge upon successful execution, otherwise it follows OnError.

Parameters:

Parameter	Description
Input Data *	Path to the file that contains the attachment data.
Content-Type *	Value of the Content-Type MIME header.
Content Description	Value of the Content-Description MIME header.
Content-Disposition	Value of the Content-Disposition MIME header.
Content-ID	Value of the Content-ID MIME header.
MIME Header Namespace	Special register namespace from which additional MIME headers for the attachment are taken. If no value is specified, no MIME headers are added beyond those generated by the header-specific agent parameters.

Add Attachment Service (com.ibi.agents.XDAddAttachmentAgent)

Syntax:

```
com.ibi.agents.XDAddAttachmentAgent
```

Description:

This service adds a new attachment with the contents determined by the value of a string expression. The Java Character Set parameter specifies how the Java characters in the string are converted to bytes in the body of the attachment. The attachment headers are specified by the special registers of type HDR in the MIME Header Namespace.

There are also four parameters available to specify the most common MIME headers. When used, these parameters override special registers of the same name. Notice the value of the Content-ID header is taken as is, so the value must contain the surrounding angle brackets. For example, a valid value for Content-ID might be <cid>. This service follows the OnSuccess edge upon successful execution, otherwise it follows OnError.

Parameters:

Parameter	Description
Input Data *	An expression that returns the contents of the attachment.
Java Character Set	The character set used to convert from Java characters to an array of bytes. If no value is specified, the default character set will be used.
Content-Type *	Value of the Content-Type MIME header.
Content Description	Value of the Content-Description MIME header.
Content-Disposition	Value of the Content-Disposition MIME header.
Content-ID	Value of the Content-ID MIME header.
MIME Header Namespace	Special register namespace from which additional MIME headers for the attachment are taken. If no value is specified, no MIME headers are added beyond those generated by the header-specific agent parameters.

Attachment Operations Service (com.ibi.agents.XDAttachOps)

Syntax:

```
com.ibi.agents.XDAttachOps
```

Description:

This service performs operations on document attachments.

Parameters:

Parameter	Description
Operation *	Determines which operation should be performed on this document. Select one of the following options from the drop-down list: ❏ deleteAll (default) ❏ deleteOne
Attachment Number	For operations on a specific attachment, which one, base 0. If a value is specified, this parameter takes precedence over the Content-ID parameter.
Content-ID	For operations on a specific attachment, the Content-ID of the attachment. This parameter is ignored if a value for the Attachment Number parameter is specified. Although the specification of the Content-ID header requires the value to be enclosed in angle brackets, you must not use the angle brackets in this property.

Attachment to Document Service (com.ibi.agents.XDAttachmentToDocAgent)

Syntax:

```
com.ibi.agents.XDAttachmentToDocAgent
```

Description:

This service finds an attachment and makes it the body of the document. The attachment can be selected by index or by Content-ID. If the Attachment Number is specified, it takes precedence over the Content- ID. The attachment index is base 0. If present, the Content- ID must NOT contain the surrounding angle brackets.

For example, the value cid for the Content-ID parameter will match an attachment with a Content-ID header equal to <cid>. The Header Namespace is the special register namespace where MIME headers for the selected attachment will be stored. The attachment will be stored as bytes if the Keep Document Flat parameter is enabled. Otherwise, the attachment will be parsed as XML. If the attachment is itself a Multipart, then the document will contain the parse of the Main Body Part and the other parts will appear as document attachments.

The MIME headers of the Main Body Part will be saved as registers in the Main Body Part Header Namespace. This is needed to keep them distinct from the Multipart headers. The Delete Attachment parameter determines what to do with the selected attachment when it is not a Multipart: the attachment can be preserved or deleted. This service follows OnSuccess upon successful execution, otherwise, it follows fail_notfound if the attachment cannot be found, or it follows fail_operation if there is another error.

Parameters:

Parameter	Description
Attachment Number	The number of the attachment to be retrieved. If specified, it takes precedence over the Content-ID.
Content-ID	The Content-ID of the attachment to be retrieved. This value is ignored if the attachment number is specified.
Header Namespace	Special register namespace where MIME headers for the selected attachment will be stored.
Main Body Part Header Namespace	If the current attachment is itself a Multipart, this is the special register namespace where the MIME headers for the main body part will be stored.
Keep Document Flat	Determines whether to keep the body of the document as an array of bytes. This parameter is set to *false* by default.
Delete Attachment	Determines whether to delete the attachment after it becomes the body of the document. This parameter is set to *false* by default.

Cross-Origin Resource Sharing Service (com.ibi.agents.XDCorsAgent)

Syntax:

```
com.ibi.agents.XDCorsAgent
```

Description:

This service implements the server-side processing of Cross-Origin Resource Sharing (CORS) as described in *http://www.w3.org/TR/cors/*.

Normally, browsers forbid cross-origin requests for security reasons. CORS specifies a mechanism that allows browsers to make cross-origin requests to resources that opt in to provide access. CORS defines a set of HTTP headers and the rules to process them. Most common browsers implement CORS natively. It is expected that the user agent making the request will be a browser. The CORS service implements the server-side rules to respond to CORS requests received by an nHTTP listener.

If the request is simple (such as GET, HEAD, or POST), then the user agent (for example, the browser) can make the request directly. The user agent adds the *Origin* header to indicate which site is asking for this resource. The syntax of an origin is:

```
scheme "://" host [ ":" port ]
```

The same syntax is used to configure the allowed origins in the service.

If the request is not simple, then the user agent must send a pre-flight request before the actual request to authorize it. The pre-flight request is an OPTIONS request for the same URL. The *Access-Control-Request-Method* header indicates the method of the actual request (for example, PUT). The optional *Access-Control-Request-Headers* header indicates which headers will be used in the actual request. Depending on the response of the pre-flight request, the user agent can abort or make the actual request following the same rules as a simple request.

The CORS service analyzes the request, and depending on its configuration, will report one of the following results:

- The request is allowed (allowed).
- The request is not allowed (notallowed).
- The request is not CORS compliant (notcompliant).

In accordance with the specification, the CORS service adds the CORS response headers only if the request is allowed. The new headers appear as Special Registers (SREGs) in the Response Header Namespace.

The CORS service sets the output document of an OPTIONS request to a zero-length byte array. This will return an empty body with Content-Length equal to 0. This is the most appropriate result, but an application can modify the output document afterwards if necessary. For other requests, the CORS service returns the input document as the output document. Unlike the headers, the output document will be the same irrespective of the actual outcome (allowed, notallowed, or notcompliant).

The nHTTP listener must be configured carefully to make CORS processing possible. The OPTIONS Handling property must be set to *event*. This instructs the listener to pass the request to the channel instead of responding directly. The Request Header Namespace and the Response Header Namespace must not be *none*, and they must be different from each other. This makes the request headers available to the CORS service, and allows the response headers be sent with the response. Of course, the Excluded Headers property should not list the CORS headers. The HTTP Response Code must be 200 for a CORS pre-flight response. The Authentication Scheme and the Authentication Realm properties should be considered carefully. CORS is a mechanism used to relax security. It does not replace the need for user credentials to protect sensitive data.

Parameters:

The following table describes the parameters for the Cross-Origin Resource Sharing Service.

Parameter	Description
Allowed Origins	List of origins that are allowed access to the resource. Leave empty or enter * to allow all origins. The syntax of an origin is: `scheme "://" host [ ":" port ]` This property is used to validate the Origin header in the request and influences the value of the Access-Control-Allow-Origin header in the response. The return value of the Access-Control-Allow-Origin header will be * if the Allowed Origins is * and Supports Credentials is *false*.
Allowed Methods	Comma-separated list of HTTP methods that are supported by the resource. Leave empty to allow all methods. This is used to validate the Access-Control-Request-Method header in the pre-flight request and to initialize the Access-Control-Allow-Methods header in the pre-flight response.
Supported Headers	Comma-separated list of HTTP header field names that are supported by the resource. Leave empty to claim support for all headers. This is used to validate the Access-Control-Request-Headers header in the pre-flight request and to initialize the Access-Control-Allow-Headers header in the pre-flight response.

Parameter	Description
Exposed Headers	Comma-separated list of HTTP header field names of headers (other than the simple response headers) that the resource might use and can be exposed. This is used to initialize the Access-Control-Expose-Headers header in the response of the actual request.
Supports Credentials	Indicates whether the resource supports user credentials in the request. This is used to initialize the Access-Control-Allow-Credentials header in the response. Notice the return value of the Access-Control-Allow-Origin header will never be * for a resource that supports credentials.
Max Age	Specifies the amount of seconds the user agent (for example, the browser) is allowed to cache the result of the pre-flight request. The value 0 means unbounded. This is used to initialize the Access-Control-Max-Age header in the response of the pre-flight request.

Edges:

The edges returned are described in the following table.

Edge	Description
success	The message was successfully analyzed. The cors Special Register contains the result.
fail_parse	An iFL expression could not be evaluated.
fail_operation	The service is not executing a request received by an nHTTP listener.

Special Registers:

The following table describes the Special Register (SREG) that is assigned upon a successful execution.

Special Register	Description
cors	The outcome of analyzing the request. This will be one of the following values: ❑ allowed ❑ notallowed ❑ notcompliant **Note:** These values are lowercase and do not contain any spaces.

Examples

The following examples show the result of executing the Cross-Origin Resource Sharing Service on various HTTP requests. The following table lists the parameter values for the Cross-Origin Resource Sharing Service that are used by the examples.

Parameter	Value
Allowed Origins	http://host1.com http://host2.com:8080
Allowed Methods	GET,POST
Supported Headers	ReqHdr1,ReqHdr2,ReqHdr3
Exposed Headers	RespHdr1,RespHdr2
Supports Credentials	true
Max Age	30

Example 1

This HTTP request is missing the Origin header. When analyzing this request, the service would set the cors Special Register to *notcompliant* and return *success*. No response headers would be added. The output document is the input document.

```
GET /app/page1 HTTP/1.1
Host: example.com
```

Example 2

Since the Allowed Origins is not empty or *, the origin must match exactly one of the listed origins. When analyzing this request, the service would set the cors Special Register to *notallowed* and return *success*. No response headers would be added. The output document is the input document.

```
GET /app/page1 HTTP/1.1
Host: example.com
Origin: http://host.com
```

Example 3

When analyzing this request, the service would set the cors Special Register to *allowed* and return *success*. The output document is the input document.

```
GET /app/page1 HTTP/1.1
Host: example.com
Origin: http://host1.com
```

The following response headers will be added:

```
Access-Control-Allow-Origin: http://host1.com
Access-Control-Expose-Headers: RespHdr1,RespHdr2
Access-Control-Allow-Credentials: true
```

Example 4

This pre-flight request is missing the Access-Control-Request-Method header. When analyzing this pre-flight request, the service would set the cors Special Register to *notcompliant* and return *success*. No response headers would be added. The output document is a zero-length byte array.

```
OPTIONS /app/page1 HTTP/1.1
Host: example.com
Origin: http://host1.com
```

Example 5

The actual request method HEAD is not allowed. When analyzing this pre-flight request, the service would set the cors Special Register to *notallowed* and return *success*. No response headers would be added. The output document is a zero-length byte array.

```
OPTIONS /app/page1 HTTP/1.1
Host: example.com
Origin: http://host1.com
Access-Control-Request-Method: HEAD
```

Example 6

The header names in Access-Control-Request-Headers are all supported headers. When analyzing this pre-flight request, the service would set the cors Special Register to *allowed* and return *success*. The output document is a zero-length byte array.

```
OPTIONS /app/page1 HTTP/1.1
Host: example.com
Origin: http://host1.com
Access-Control-Request-Method: GET
Access-Control-Request-Headers: ReqHdr1,ReqHdr2
```

The following response headers will be added:

```
Access-Control-Allow-Origin: http://host1.com
Access-Control-Allow-Methods: GET,POST
Access-Control-Allow-Headers: ReqHdr1,ReqHdr2,ReqHdr3
Access-Control-Allow-Credentials: true
Access-Control-Max-Age: 30
```

Document to Attachment Service (com.ibi.agents.XDAttachmentFromDocAgent)

Syntax:

```
com.ibi.agents.XDAttachmentFromDocAgent
```

Description:

This service adds a new attachment with the contents determined by the input document. The attachment headers are specified with user parameters. The name and value of the user parameter become the name and value of the header. Notice the value of the Content-ID header is taken as is, so the value must contain the surrounding angle brackets.

For example, a valid value for Content-ID might be <cid>. The Attachment Type parameter specifies the attachment Content-Type. If the Attachment Type is omitted, a Content-Type is chosen based on the document data:

- application/binary if the document contains bytes
- text/plain if the document is flat
- application/xml otherwise

The optional Attachment Name parameter controls the name sub-parameter of the Content-Type header. The Content-Type name can also be given directly in the Content-Type header value if desired.

The Body Result parameter specifies what to return in the output document: keep copies the input document to the output document, empty returns an empty document instead. This service follows OnSuccess upon successful execution, otherwise it follows the fail_attach edge.

Parameters:

Parameter	Description
Attachment Name	The name of this attachment, for example, file name.
Attachment Type	The MIME type. If no value is specified, the attachment type is generated based on the data format.
Body Result	Specify how to handle the body of this document by selecting one of the following values from the drop-down list: ❏ keep (default) ❏ empty

HTTP Cookie Agent Service (com.ibi.agents.XDCookieAgent)

Syntax:

```
com.ibi.agents.XDCookieAgent
```

Description:

This service adds a cookie to the HTTP responses, enabling them to retain stateful information.

Parameters:

Parameter	Description
Cookie Name (required)	The name of the cookie.

Parameter	Description
Cookie Value	The value of the cookie, which must comply with RFC 6265.
Path Attribute	Value of the Path attribute in the generated cookie. The value / matches any path.
Domain Attribute	Value of the domain attribute in the generated cookie. Leave empty to match the originating server.
Max-Age Attribute	Value of the Max-Age attribute in the generated cookie. This indicates the maximum lifetime of the cookie, represented as a number of seconds. Leave blank to omit the attribute which lets the user agent determine when the cookie expires.
Secure Attribute	Whether to add the Secure attribute in the generated cookie. Automatically adds the attribute if the listener is secure. Select one of the following options from the drop-down list: ❏ *automatic* (default) ❏ *false* ❏ *true*
HTTPOnly Attribute	Whether to add the HttpOnly attribute in the generated cookie. ❏ *true* ❏ *false* (default)

HTTP Emit Service (com.ibi.agents.XDHTTPEmitAgent)

Syntax:

```
com.ibi.agents.XDHTTPEmitAgent
```

Description:

This service is a general HTTP emitter for use within the service stack. The document is posted using HTTP to a designated server.

Parameters:

Parameter	Description
Configuration Parameters for HTTP Emit Agent Service	
Target URL (required)	Specify a URL to post this information.
Action Method	Select one of the following methods from the drop-down list: ❏ *GET* - Data on the URL and URL encoded. ❏ *POST* - (default) Content-Length header.
Request content type	This overrides content type of a request. Select one of the following options from the drop-down list: ❏ *application/EDI-X12* ❏ *application/EDIFACT* ❏ *application/XML* ❏ *text/html* ❏ *text/plain*
User ID	User ID for Basic Authentication challenges.
Password	Password for Basic Authentication challenges.
Response timeout value in seconds	Seconds to wait for response before signaling error.
IP Interface Host	Local IP Interface from which the outgoing IP socket originates.
IP Interface Port	Local IP Port from which the outgoing IP socket originates.
Relay Inbound Content Type	If set to *true*, relay headers as received (content type). *False* is the default.
Set TCP No Delay	If set to *true*, disables Nagle's Algorithm on the client socket. This will result in faster line turnaround at the expense of an increased number of packets.
Proxy	

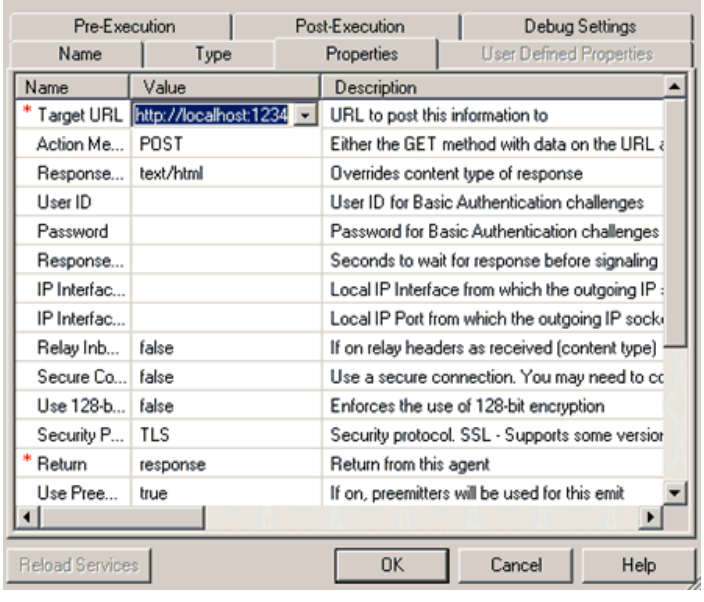
Parameter	Description
Proxy	If set to *true*, emit through proxy server. *False* is the default.
Proxy URL	URL of the proxy server.
Proxy User ID	User ID for proxy challenges.
Proxy Password	Password to access the proxy server.
HTTPS	
Secure Connection	*True* to use a secure connection. You may need to configure the Keystore under HTTPS section of the system properties if client authentication is required. Note, if keystore is configured in system properties make sure it has the CA certificate or the client certificate of the server you're connecting to. If keystore is not configured in system properties default truststore located under JRE_HOME/lib/security/cacerts will be used. *False* is the default.
Use 128-bit Encryption	*True* to use 128-bit encryption. *False* is the default.
Security Protocol	Select one of the following security protocols from the drop-down list: ❑ *SSL* - Supports some version of SSL; may support other versions. ❑ *SSLv2* - Supports SSL version 2 or higher. ❑ *SSLv3* - Supports SSL version 3; may support other versions. ❑ *TLS* - (default) Supports some version of TLS; may support other versions. ❑ *TLSv1* - Supports TLS version 1; may support other versions.
Agent Specific Parameters	

Parameter	Description
Return (required)	Return from this agent. Select one of the following options from the drop-down list: ❑ *input* ❑ *response* (default) ❑ *status*
Use Preemitters	If set to *true*, preemitters will be used for this emit. *True* is the default.
Response Wrapper Tag	The tag name with which to wrap the response if the response is non-XML and must be XML
Response Base64 Encoded	If set to *true*, the response will be Base64 encoded. *False*is the default.
Response Data Format	Format of the response, default is XML ❑ *XML* ❑ *flat* (default)

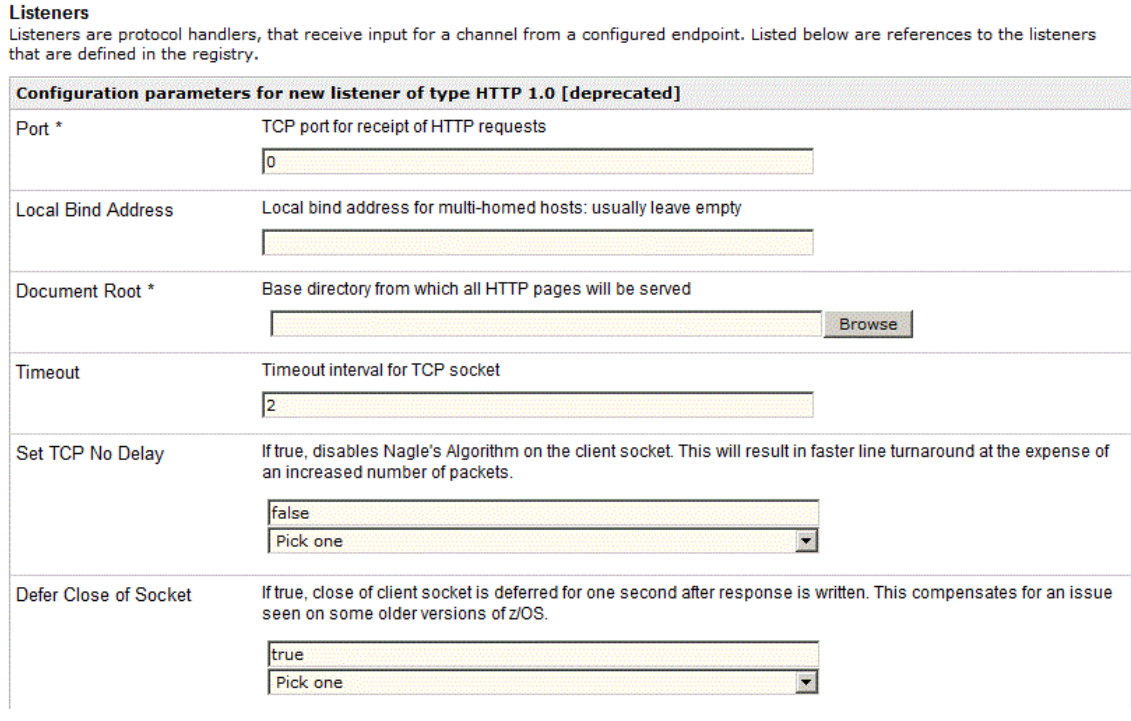
The result of the post appears in the <native> section of the <emitstatus> result. If the Return parameter value is STATUS, the status message is always generated. If it is RESPONSE (the default) the actual information returned from the POST is emitted, except in the cases of an error in which case the status information is emitted. The POST is considered successful if the POST can reach its destination and a 200 status is received. A successful return response is considered to be XML if the first character is a <. Otherwise, a flat document is created.

Example:

The following is an example of configuration settings for an HTTP Emit service object in iIT Designer:



A simple way to test your HTTP Read service is to first configure a channel that uses an HTTP listener as the inlet and a File emitter as the outlet to examine the messages that are received by the listener. Your HTTP listener configuration settings would look similar to the following example, where an HTTP port and a document root are specified.



After you deploy and start the channel with the HTTP listener and File emitter, you can test run your process flow, which contains an HTTP Emit service object. You must supply an incoming document with the appropriate content (for example, HTML):

```
<html>
  <head>
    <title>my iWay http</title>
   </head>
     <body> Testing HTTP Emit Service! </body>
</html>
```

If the service runs successfully, you will receive this HTML document in the output folder that is specified by your File emitter. The HTTP listener channel picks up the document that is emitted by the HTTP Emit service.

HTTP Nonblocking Emit Service (com.ibi.agents.XDNHttpEmitAgent)

Syntax:

```
com.ibi.agents.XDNHttpEmitAgent
```

Description:

Emits nHTTP messages to a client without interrupting a configured business process. For more information, see the *iWay Service Manager Protocol Guide*.

Parameters:

Parameter	Description
Configuration Parameters	
Target URL (required)	URL that is used to post this information.
HTTP Client Provider (required)	HTTP client Provider that is used to manage connections for this emitter.
Cookie Store Name	Allows thread-specific management of cookies. If not specified, a cookie store global to the HTTP Client provider will be used.
Action Method	Select one of the following supported methods from the drop-down list: ❑ *DELETE* ❑ *GET* - will send a data-encoded request to the configured URL. ❑ *HEAD* - will send a request to the configured URL and return a status document. ❑ *POST* - sends the current document as a request entity. ❑ *PUT*

Parameter	Description
Request Content Type	Overrides content type of response. Select one of the following options from the drop-down list: ❑ *application/EDI-X12* ❑ *application/EDIFACT* ❑ *application/XML* ❑ *text/html* ❑ *text/plain*
User ID	User ID for Basic Authentication challenges.
Password	Password for Basic Authentication challenges.
Domain	Domain for NTLM authentication challenges. Note that to use NTLM, you must enable connection persistence.
Request Header Namespace	Special register namespace from which HTTP headers for the outgoing request will be taken. Choose Default Namespace to send HDR type registers with no namespace prefix, or supply a namespace prefix here. None means that no special registers will be sent as HTTP headers. ❑ Supply a namespace prefix here to indicate which headers to send. ❑ *Default Namespace* - Sends HDR type registers with no namespace prefix. ❑ none - Means that no special registers will be sent as HTTP headers.
Request Main Part Header Namespace	Special register namespace from which MIME headers for the outgoing request will be taken. Provide a prefix to control the request Main BodyPart headers in the presence of attachments. Selecting none means that no special registers will be sent as MIME headers.

Parameter	Description
Response Header Namespace	Special register namespace into which HTTP headers from the incoming response will be saved. Choose Default Namespace to create special registers with no namespace prefix, or supply a namespace prefix here. None means that no special registers will be created. ❑ *Default Namespace* - Creates special registers with no namespace prefix. ❑ Pick one - Supply a namespace prefix here to indicate header namespace.
Excluded Headers	A comma delimited list (case insensitive) of headers that should not be sent with the request, even if they are found in the request header namespace.
Ask for Compressed Response	If set, requests will set the Accept-Encoding header to indicate that that the client can accept a compressed response, as described in RFC-2616. If the response has a compressed content encoding, the client will automatically inflate. ❑ *deflate* ❑ *either* ❑ *gzip* ❑ *none*
Compress Request	If set, the request entities will be compressed using the selected encoding and the content-encoding header will be set accordingly. ❑ *deflate* ❑ *gzip* ❑ *none*

Parameter	Description
Release Connection Immediately	If set to *true*, then the connection will be released to the connection pool immediately. This is the default. If set to *false*, then the connection identifier is stored in the *httpclient-key* special register and the HTTP Client Manager agent must be called later to release the connection explicitly. The main reason to retain the connection is to reuse it for streaming of other services.
Maximum HTTP Client Manager Delay	Maximum time the HTTP Client Manager can take to handle a particular connection before it is automatically aborted. The format is [xxh][xxm]xx[s]. The default is 60 seconds.
Try Expect/Continue Handshake?	If set to *true*, the client will send the HTTP Expect: 100-continue header and await HTTP 100 response before sending the request body. The default is *false*.
Chunk Encoded Request?	If set to *true*, request entity will be sent with chunk encoding. The default is *false*.
Maximum Request Size	Maximum size, after compression, of a request entity that can be sent with this emitter. A value of 0 means no maximum and if no value is specified, the parameter defaults to 256KB.
Maximum Response Size	Maximum size of a response entity that can be received by this emitter. 0 means no maximum and blank will default to 256KB.
TCP Properties	
Persistence	If set to *true*, ask the server to maintain the connection. The default is *false*.
Response Timeout value in Seconds (required)	The value in seconds to wait for a response before generating an error.
Agent Specific Parameters	

Parameter	Description
Return (required)	URL to post this information to, for example, *http://thehost:9876*. Choose: ❑ *input* - to return input document ❑ *status* - or an XML document with transaction parameters and status ❑ *response* - to capture output from the server
Preemitter	If set to *true*, (default) the preemitters will not run.
Response Wrapper Tag	The tag name with which to wrap the response if the response is non-XML and must be XML.
Response Base64 Encoded	If set to *true*, the response will use Base64 encoding. *False* is the default.

HTTP Read Agent (com.ibi.agents.XDHTTPReadAgent)

Syntax:

```
com.ibi.agents.XDHTTPReadAgent
```

Description:

This service reads an HTTP source using HTTP GET and returns a result. The GET facility input accepts a URL in the incoming document and issues an HTTP GET through that URL. The transform business service and standard output can accept and operate upon this output. It is presumed that some output is returned, otherwise an error is generated.

In a use case scenario, the HTTP Read service can be part of an architecture for a web search engine, similar to the Google search engine. The service could be used to traverse or spider the contents of webpages for indexing purposes.

Parameters:

Parameter	Description
URL Tag Name	TTag (element) name in input document containing the URL. Do not use if URL parm is used.

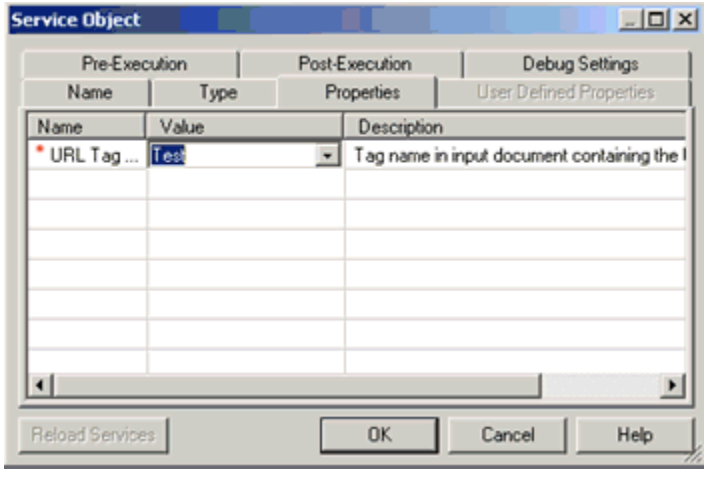
Parameter	Description
URL	URL for the read. Used if the tag (element) name is omitted
Access Timeout	Timeout in milliseconds for the access; applies if greater than 0.

The input document might be the following assuming that the tagname parameter is set to *inurl*:

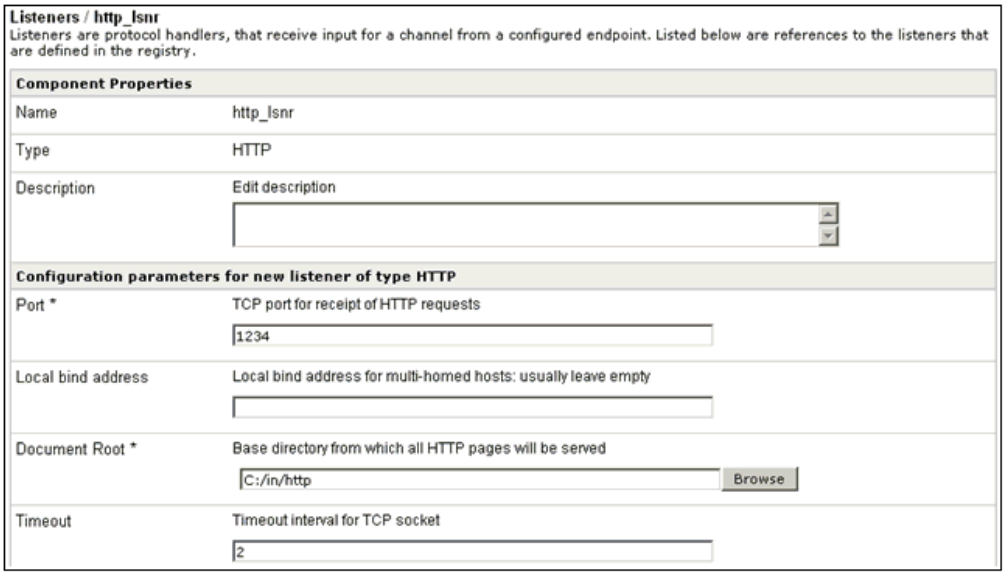
```
<?xml version="1.0"?>
<inurl>http://localhost:1234/xmlone.xml</inurl>
```

Example:

The following is an example of configuration settings for an HTTP Read service object in iIT Designer:



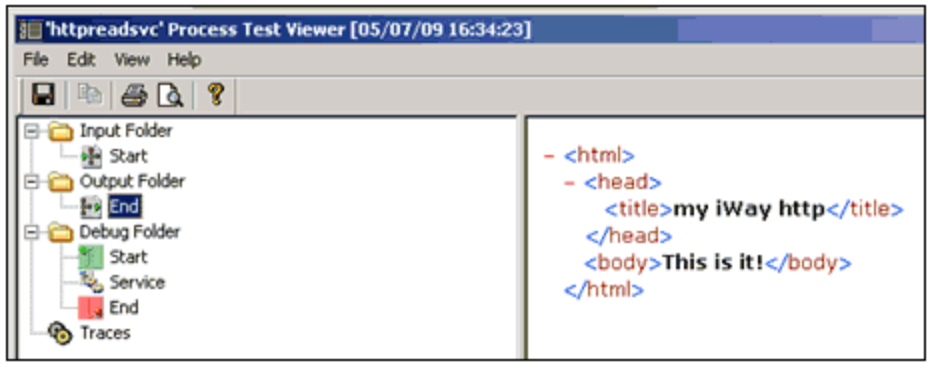
A simple way to test your HTTP Read service is to first configure a channel that uses an HTTP listener. Your HTTP listener configuration settings would look similar to the following example, where an HTTP port and a document root are specified.



After you deploy and start the channel with the HTTP listener, you can test run your process flow, which contains an HTTP Read service object. You must ensure that the file name you supply within the <Test> tag of your incoming document is valid and is also contained within the document root folder specified for your HTTP listener configuration settings. A sample HTTP read incoming document can have the following format:

```
<Test>http://localhost:1234/index1.htm</Test>
```

If the service ran successfully, you will receive the contents of index1.hml as the output from the service. For example:



HTTP ReST Routing Service (XDReSTRouteAgent and XDReSTRouteReviewer)

Syntax:

```
com.ibi.agents.XDReSTRouteAgent
```

```
com.ibi.agents.XDReSTRouteReviewer
```

iIT Service Object:

http: HTTP ReST Routing Service

Description:

This service routes a Representational State Transfer (ReST) request to a process flow based on the request method and path. An HTTP ReST Routing Reviewer (com.ibi.reviewer.XDReSTRouteReviewer) is also provided, which shares the same parameters as the service.

The main purpose of these components is to identify a process flow to execute, given a URI path and the type of the HTTP request. The name of the process flow will be stored in a special register (SREG). In addition to this routing function, some elements in the URI path, and all keys and values in the URI query string, will be stored as SREGs. This functionality will be made available as a service for use within a process flow, but also as a reviewer. This allows for earlier execution in the processing stage to allow conditional routing to handle the execution of the identified process flow.

The service uses an XML document whose structure corresponds to the elements in the path. This XML document has the following structure:

```
<route>
  <verb type="GET">
    <step type="literal" value="master">
      <step type="literal" value="vendor">
        <step type="regex" value="[0-9]+" flow="vendorRequest" sreg="true">
         <step type="literal" value="item" flow="vendorAllItemsRequest">
           <step type="regex" value="[0-9]+" flow="vendorItemRequest" sreg="true">
              <sreg name="extrareg1" value="something"/>
              <sreg name="extrareg2" value="somethingelse"/>
              <step type="literal" value="allproperties"
               flow="vendorItemAllProperties"/>
           </step>
           <step type="default" flow="VendorItemUnknownRequest"/>
         </step>
       </step>
     </step>
   <step type="default" flow="UnknownMasterRequest"/>
  </step>
 </verb>
</route>
```

A schema will follow for this XML document.

In this section, the individual components of the URI path are called *path elements* and the <step> elements in the XML document are called *steps*.

Each step can match a path element. How a step is matched to a path element depends on its type (as indicated by the *type* attribute of the step element).

1. A *literal* step matches if the value of the step (as indicated by the value of the *value* attribute of the step element) matches the path element exactly.
2. For a *regex* step, the value of the step is a regular expression. A regex step matches if the regular expression matches the path element.
3. A *default* step matches if no other step matches.

The search process starts at the verb node that corresponds to the type of the HTTP request. For the first element in the path, each step child node of the verb node is checked to see if it matches the path element. The first step that matches is taken. This is repeated for the second path element, looking at the step children of the first step, and so on, until the final path element is reached.

Note: The order of step nodes in the routing XML document is very important.

Consider the following example, which shows a path element "1234" and two step nodes:

```
<step type="regex" value="[0-9]+" flow="generalIntRequest"/>
<step type="literal" value="1234" flow="specific1234Request"/>
```

In this scenario, the *regex* step is always taken, since it is seen first and also matches 1234, even though the literal step is an exact match.

If a step has the *sreg* attribute set to *true*, a Special Register (SREG) will be created using the previous path element as the SREG name and the current path element as the SREG value.

When the final path element is reached, the value of the *flow* attribute for the last step found is the name of the process flow to be executed. If, for any path element, no matching step can be found, the service will return the *fail_notfound* edge.

For example, using the document above, consider that the following URI is provided:

```
/master/vendor/1428/item/2345/allproperties
```

In this scenario, the target register is set to *vendorItemAllProperties* and the following SREGs are set:

- vendor=1428
- item=2345

Additional SREGs to set for the final step can be specified by adding <*sreg*> nodes as children of the step node. For example, consider the following URI:

```
/master/vendor/1234/item/2345
```

The target register would be set to *vendorItemRequest*, the SREGs as shown above, and the following additional SREGs would be set:

- extrareg1=something
- extrareg2=somethingelse

If the incoming URI includes a query string, the service will store the key-value pairs in the query string as SREGs. Query string parameters will not be used in the route search.

To facilitate debugging, the XML schema will include two additional attributes:

1. The <*route*> node can have a *verbose* attribute, which when set to *true* indicates that the service will generate additional tracing at the iSM DEBUG level, with details of the route search process.
2. Any <*step*> node can have a *name* attribute, the value of which, if present, will be included in the verbose tracing to make it easier to correlate traces to the routing tree.

Step Attributes:

Attribute	Description
type	The type of attribute value: ❑ **literal.** Value is a literal, exact match required. ❑ **ci_literal.** Case insensitive literal. ❑ **regex.** Regular expression. ❑ **default.** Any other (must be last in a compare set).
value	A compare value of type in the *type* attribute.
flow	Name of a process flow if this is the final step (iFL allowed).
sreg	If true, then a Special Register (SREG) will be created using the value of the previous path element as the key and this value of the current path element as the register value.
role	The user must have this role to descend on this step.
name	A node name used in tracing.

Inputs:

The service is agnostic as to the input. The service operates on the incoming URL.

Outputs:

If a process flow is found for this path, then the input will be output. In any other case, the output will either be the input document or an iSM status/error document, as determined by the Output Type parameter for this service.

Parameters:

Parameter	Description
Route File Path (required)	The path to the XML file describing the route. The file is loaded when the service is first executed and remains in effect until the listener is restarted.

Parameter	Description
ReST Verb (required)	The HTTP verb (DELETE, GET, POST, or PUT). In nHTTP this is in the SREG called *reqType*.
URI (required)	The URI of the incoming HTTP request. In nHTTP, this is stored in the SREG called *url*. This may include a path and a query string. **Note:** This is slightly different from the Object parameter in the prototype service.
Flow Name Register (required)	An SREG of this name will be created with the name of the process flow to run.
Route Namespace	SREG namespace prefix to use when creating SREGs from path steps, including the process flow name register. Applies to the routing portion of the URL.
Query Namespace	SREG namespace prefix to use when creating SREGs from the query string portion of the URL. Defaults to the same as the Route Namespace. Using a separate namespace can allow these registers to be passed and managed as a group.
Output Type	Determines in the event of a routing error, what document should be output from this service. Select one of the following document types: ❑ **status.** A standard iSM error document. ❑ **input.** The original input document.

Edges:

Parameter	Description
Success	A process flow was located.
fail_notfound	No process flow was located.

Parameter	Description
fail_parse	Unable to process configuration parameters, which is most likely an iFL error.
fail_route	Other failure specific to this agent (for example, unable to locate or parse the routing file).
fail_security	A role does not match, meaning that this user is not permitted to use this URL (security trigger flow runs in iSM Version 6.1.9).

ReST Process Flow Message:

When a non-POST verb in encountered by nHTTP, a default message is generated. The schema for this message is located in the following directory:

<iwayhome>/etc/manager/extensions/iwxnhttp/misc/doc

From time to time, iWay Software may add additional, non-conflicting elements to this schema in future releases.

The input document to the process flow holds the *parsed* information from the arriving HRRP input, including all headers and the URL split up for use.

For example (spacing for display purposes only):

```
<http user="auto" type="GET">
 <parms>
  <parm name="ibse-port">9000</parm>
  <parm name="Host">clientbox:10000</parm>
  <parm name="Connection">Keep-Alive</parm>
  <parm name="pdm">0</parm>
  <parm name="version">1.1</parm>
  <parm name="Accept-Language">en-AU</parm>
  <parm
name="action">user.req?user=999999999&account=1234567890123456&tranid=tid1234</parm>
<parm name="Accept">text/html, application/xhtml+xml, */*</parm>
  <parm name="User-Agent">Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64;
Trident/5.0)</parm>
  <parm name="url">/user.req?user=999999999&account=1234567890123456&tranid=tid1234</
parm>
```

```
<parm name="ip">127.0.0.1</parm>
  <parm name="source">hostname unknown</parm>
  <parm name="Accept-Encoding">gzip, deflate</parm>
  <parm name="reqType">GET</parm>
 </parms>
 <body />
 <url secure="false">
  <host>clientbox</host>
  <port>10000</port>
  <path>/user.req</path>
  <query>user=999999999&account=1234567890123456&tranid=tid1234</query>
  <queryparms>
   <parm name="user">99999999</parm>
   <parm name="account">1234567890123456</parm>
   <parm name="tranid">tid1234</parm>
  </queryparms>
  <incomingurl>    http:clientbox:1000/
user=999999999&account=1234567890123456&tranid=tid1234
  </incomingurl>
  </url>
 <version>1.1</version>
</http>
```

Using an XML iterator, the process flow can easily step through the <*queryparms*>, or using standard XSLT, can extract a specific URI value as required. All registers set in the channel are housed in the <*http/parms*> elements and can be extracted if required.

The user (source of the message) can be authenticated using standard nHTTP services, avoiding the need to do so within the executing process flow.

iWay Functional Language (iFL) Support:

The iFL language used within iSM provides facilities to support RESTful services. Key functions for this support are listed and described in the following table:

Function	Description
_urlparse()	Parses a URL into its component parts, including searching for specific keywords in the query section of the URI.
_token()	Enables extraction of specific portions of the URI as required.

The information extracted by the iFL is represented in the incoming document. The iFL simply makes it possible to look at the URL directly.

For more information on _urlparse() and _token(), see the *iWay Functional Language Reference Guide*.

Route Step XML Schema Document:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://
www.iwaysoftware.com/RouteStep" xmlns:tns="http://www.iwaysoftware.com/RouteStep"
elementFormDefault="qualified">
    <complexType name="sregType">
        <annotation>
            <documentation>
                Represents a special register to create if its parent
                step matches the last path element.
            </documentation>
        </annotation>
        <attribute name="name" type="string" use="required"></attribute>
        <attribute name="value" type="string" use="required"></attribute>
    </complexType>
    <complexType name="stepType">
        <annotation>
            <documentation>
                Represents one step in the URI path for looking up the
                route. The lookup process attempts to match elements in
                the path to steps.
            </documentation>
        </annotation>
        <choice minOccurs="0" maxOccurs="unbounded">
            <element name="sreg" type="tns:sregType"/>
            <element name="step" type="tns:stepType"/>
        </choice>
        <attribute name="type" use="required">
            <simpleType>
    <restriction base="string">
                    <enumeration value="literal"></enumeration>
                    <enumeration value="ci_literal"></enumeration>
                    <enumeration value="regex"></enumeration>
                    <enumeration value="default"></enumeration>
                </restriction>
            </simpleType>
        </attribute>
        <attribute name="value" type="string" use="optional">
            <annotation>
    <documentation>
                    When type is 'literal', the value attribute holds a
                    literal that must match the path element. Type "ci_literal"
                    is similiar, but ignores case when matching the path element.
                    When type is 'regex', this attribute holds a regular
                    expression to match the path element. When type is
                    'default', this attribute is not used.
                </documentation>
            </annotation>
```

```
        </attribute>
        <attribute name="flow" type="string" use="optional">
            <annotation>
                <documentation>
                    The flow to run for this URI path. Will be set into
                    the target special register.
                </documentation>
            </annotation>
        </attribute>
        <attribute name="sreg" type="boolean">
            <annotation>
                <documentation>
                    If true, the value of this path element will be
                    saved as a special register, with key as the value
                    of the previous path element.
                </documentation>
            </annotation>
        </attribute>
        <attribute name="name" type="string" use="optional">
            <annotation>
                <documentation>
                    Use the name attribute to identify this step in
                    debug tracing.
                </documentation>
            </annotation>
        </attribute>
        <attribute name="role" type="string" use="optional">
            <annotation>
                <documentation>
                    Optionally, the role required to the flow that
                    would be run if this were the final element in the URI path.
Assumes
                    that the user has been authenticated by an iSM auth realm.
                </documentation>
            </annotation></attribute>
    </complexType>
    <complexType name="verbType">
        <annotation>
            <documentation>Matches the ReST verb or HTTP method for the current
request.</documentation>
        </annotation>
        <sequence>
            <element name="step" type="tns:stepType" minOccurs="1"
maxOccurs="unbounded"></element>
        </sequence>
        <attribute name="type" use="required">
            <simpleType>
                <restriction base="string">
                    <enumeration value="GET"></enumeration>
                    <enumeration value="POST"></enumeration>
                    <enumeration value="PUT"></enumeration>
                    <enumeration value="DELETE"></enumeration>
                </restriction>
            </simpleType>
        </attribute>
    </complexType>
    <complexType name="routeType">
        <sequence>
            <element name="verb" type="tns:verbType" minOccurs="1" maxOccurs="4"></
element>
        </sequence>
        <attribute name="verbose" type="boolean" use="optional"></attribute>
    </complexType>
    <element name="route" type="tns:routeType"></element>
```

HTTP Session Invalidator Service (com.ibi.agents.XDHttpSessionInvalidator)

This service is used to terminate the session. The following table lists and describes its parameter.

Parameter	Description
Expire Cookie	Determines how the termination is to be effected. If set to *false*, it removes the session information from the server. No further action is taken. If set to *true*, in addition to removing the session information, it sends an instruction to the client to delete the JSESSIONID cookie itself.

The service returns the input document on the success edge.

As a good practice, you can use this service when the application has completed the session, so as to reduce server resources.

OAuth 1.0 Authentication Service

Syntax:

```
com.ibi.agents.XDOAuth1Agent
```

Description:

This service creates the HTTP Authorization header for OAuth 1.0a as specified in RFC5849. This RFC describes a 3-legged protocol where the user authorizes the client application to access a protected resource hosted by a service provider.

The OAuth 1.0 Authentication supports a variant called the 0-legged protocol where the request is signed without the user credentials. The signature is computed using just the consumer key and the consumer secret. These credentials are obtained once when the application programmer registers his client application with the service provider during development. This service assumes the consumer secret is a private key.

Parameters:

The following table lists and describes the parameters for the OAuth 1.0 Authentication service.

Parameter	Description
HTTPS URL	Request URL used in the computation of the Signature Base String.
HTTP Method	HTTP Method used in the computation of the Signature Base String. Selecting POST will also cause the current document to be hashed to produce the oauth_body_hash.
Header Namespace	Special register namespace where the Authorization HTTP header is stored. If not supplied, the default namespace will be used.
Client ID	The consumer key of the client credentials.
KeyStore Provider	Provider for the keystore containing the client private key.
Private Key Alias	Alias of the private key within the keystore.
Private Key Password	Password for the private key. If left blank, the password for accessing the keystore will be used.

The OAuth Authentication service only creates the Authorization header. The HTTP request must be sent in a separate step, usually with the NHTTP Emit service.

The HTTPS URL and HTTP Method parameters are used in the Signature Base String. They must match the Target URL and Action Method of the NHTTP Emit service. The URL scheme must be HTTPS because an SSL connection is needed to protect the information that is passed in clear. Choosing the POST method also instructs the service to compute a hash of the entity body to be part of the signature. This algorithm was specified by Google in its OAuth Request Body Hash extension.

The Authorization header will be stored in the specified Header Namespace. This parameter should match the Request Header Namespace in the NHTTP Emit Agent. This will ensure that the header is sent with the request. It is possible to use different namespaces, as long as the Authorization register is copied to the Request Header Namespace before the request is sent.

The Client ID is the consumer key supplied by the service provider when the developer registered the client application with the service provider. This serves as the user name for the client application. The service provider uses the client ID to retrieve the public key to validate the signature.

The KeyStore Provider is the name of the provider that holds the client private key. The Private Key Alias and Private Key Password are the Alias and Password for the private key. This key is used as the consumer secret when signing the Authorization header.

The output document is the same as the input document.

For the POST method, the document contains the same data but it will be stored as bytes if it was not already. This is to guarantee the document will not be altered before it is sent because any change to the document would invalidate the signature.

Edges:

The following table lists and describes the edges that are returned by the OAuth Authentication service.

Edge	Description
success	The Authorization header was successfully created.
fail_parse	An iFL expression could not be evaluated.
fail_operation	The operation could not be completed successfully.

Example 1

This example shows the creation of an OAuth 1.0a Authorization header for a GET method. The following table lists the parameter values for the service.

Parameter	Value
HTTPS URL	*https://sandbox.api.mastercard.com/atms /v1/atm?Format=XML&PageOffset=0&PageLength=10&PostalCode=46312&Country=USA*
HTTP Method	`GET`
Header Namespace	`hdrns`

Parameter	Value
Client ID	DKB0vGSHs4r1Vv308yObMj4QhhJkIMP5G 3a14KmEa7f96b5e!414a78536b4a6f6272634a41446e 4566483851625a7a413d
KeyStore Provider	keyprov
Private Key Alias	key1
Private Key Password	key1pass

This assumes key1 is the alias of a private key entry in the KeyStore provider keyprov. The service will compute the following Signature Base String. The oauth_nonce and oauth_timestamp will obviously change each time the service executes.

```
GET&https%3A%2F%2Fsandbox.api.mastercard.com%2Fatms%2Fv1%2Fat
m&Country%3DUSA%26Format%3DXML%26PageLength%3D10%26PageOffset%3D0
%26PostalCode%3D46312%26oauth_consumer_key%3DDKB0vGSHs4r1Vv308yObMj4QhhJ
kIMP5G3a14KmEa7f96b5e%2521414a78536b4a6f6272634a41446e4566483851625a7a41
3d%26oauth_nonce%3D180284899533025%26oauth_signature_method%3DRSA
SHA1%26oauth_timestamp%3D1396020436%26oauth_version%3D1.0
```

The service will store the following header value in the hdrns. Authorization special register. The oauth_signature changes every time the service is executed because the oauth_nonce and oauth_timestemp varies.

```
OAuth
oauth_signature="JjBI1gi5EMHwcihnCyK0RX7UzCC2SCtplutEjUgUXaI2nhGd4IR3L7b
WMtpJKkyUnR667lpkI7zqbM3oR3CHc2%2FgxPerD%2FSDGibHTAcTHCfV9%2F0xBVzv%2Fzo
1egU4CEqjZGSeIAeJKQYOflKSrfX8ken0MsXwXv5s9TLQuO8pRPwCfrqgrmVa%2FHhlzRxU7
pEv2kpJn4opG3Cvn01aKlotztxG8u476aEydFq03emqjVh8GMArtGDt8RhJqisJ0OB9SsaWU
K%2FsV%2BQtvghmX7G0pyQ6hLJUa3NSqlINU2k19cLOhUEnylDVD62sTZGrPe9%2B3zKLj%2
BX77eGLFKrDqOxk9w%3D%3D",oauth_version="1.0",oauth_nonce="18028489953302
5",oauth_signature_method="RSASHA1",oauth_consumer_key="DKB0vGSHs4r1Vv30
8yObMj4QhhJkIMP5G3a14KmEa7f96b5e%21414a78536b4a6f6272634a41446e456648385
1625a7a413d",oauth_timestamp="
1396020436"
```

If the Request Header Namespace is hdrns in the NHTTP Emit service, this will add the following HTTP header to the HTTP request.

```
Authorization: OAuth
oauth_signature="JjBI1gi5EMHwcihnCyK0RX7UzCC2SCtplutEjUgUXaI2nhGd4IR3L7b
WMtpJKkyUnR667lpkI7zqbM3oR3CHc2%2FgxPerD%2FSDGibHTAcTHCfV9%2F0xBVzv%2Fzo
1egU4CEqjZGSeIAeJKQYOflKSrfX8ken0MsXwXv5s9TLQuO8pRPwCfrqgrmVa%2FHhlzRxU7
pEv2kpJn4opG3Cvn01aKlotztxG8u476aEydFq03emqjVh8GMArtGDt8RhJqisJ0OB9SsaWU
K%2FsV%2BQtvghmX7G0pyQ6hLJUa3NSqlINU2k19cLOhUEnylDVD62sTZGrPe9%2B3zKLj%2
BX77eGLFKrDqOxk9w%3D%3D",oauth_version="1.0",oauth_nonce="18028489953302
5",oauth_signature_method="RSASHA1",oauth_consumer_key="DKB0vGSHs4r1Vv30
8yObMj4QhhJkIMP5G3a14KmEa7f96b5e%21414a78536b4a6f6272634a41446e456648385
1625a7a413d",oauth_timestamp="1396020436"
```

Example 2

This example shows the creation of an OAuth 1.0a Authorization header for a POST method. The following table lists the parameter values for the service.

Parameter	Value
HTTPS URL	*https://sandbox.api.mastercard.com/fraud/merchant/v1/termination-inquiry?Format=XML&PageLength=10&PageOffset=0*
HTTP Method	`POST`
Header Namespace	`hdrns`
Client ID	`DKB0vGSHs4r1Vv308yObMj4QhhJkIMP5G3a14KmEa7f96b5e!414a78536b4a6f6272634a41446e4566483851625a7a413d`
KeyStore Provider	`keyprov`
Private Key Alias	`key1`
Private Key Password	`key1pass`

The following input document is the parsed XML document:

```
<ns2:TerminationInquiryRequest xmlns:ns2="http://mastercard.com/
termination"><AcquirerId>1996
</AcquirerId><TransactionReferenceNumber>1</TransactionReferenceNumber>
<Merchant><Name>TEST</Name><DoingBusinessAsName>TEST
</DoingBusinessAsName><PhoneNumber>5555555555</PhoneNumber>
<NationalTaxId>1234567890</NationalTaxId><Address><Line1>5555 Test Lane
</Line1><City>TEST</City><CountrySubdivision>XX</CountrySubdivision>
<PostalCode>12345</PostalCode><Country>USA</Country></Address>
<Principal><FirstName>John</FirstName><LastName>Smith</LastName>
<NationalId>1234567890</NationalId><PhoneNumber>5555555555</PhoneNumber>
<Address><Line1>5555 TestLane</Line1><City>TEST</City><CountrySubdivision>XX
</CountrySubdivision><PostalCode>12345</PostalCode><Country>USA
</Country></Address><DriversLicense><Number>1234567890</Number>
<CountrySubdivision>XX</CountrySubdivision></DriversLicense></Principal>
</Merchant></ns2:TerminationInquiryRequest>
```

The service will compute the following Signature Base String. Notice the extra attribute oauth_body_hash compared to Example 1.

```
POST&https%3A%2F%2Fsandbox.api.mastercard.com%2Ffraud%
2Fmerchant%2Fv1%2Fterminationinquiry&Format%3DXML%26PageLength%3D10%
26PageOffset%3D0%26oauth_body_hash%3Dh3%252BhLMkT%252B3pBvRolKEc95fobEB8
%253D%26oauth_consumer_key%3DDKB0vGSHs4r1Vv308yObMj4QhhJkIMP5G3a14KmEa7f
96b5e%2521414a78536b4a6f6272634a41446e4566483851625a7a413d%26oauth_nonce
%3D180286176383600%26oauth_signature_method%3DRSA-SHA1%26oauth_timestamp
%3D1396020438%26oauth_version%3D1.0
```

The service will store the following header value in the hdrns Authorization special register. The oauth_signature, oauth_nonce, and oauth_timestamp will change every time the service is executed.

```
OAuth
oauth_signature="GSgJ6wUiYDznurpspn2ztn9PZeuXIBy4LZZHOSuMQrQ8OskwdWdaX0i
UXfNELxEQUniy6z5b2c06yVCut4XoYtV5XJaYnoG78bqkJ3LLVBqZ%2Brv%2F%2FTbIQmz0c
enMAinlR09QeduIHV7gPGqd%2FBi9Rkj%2BHnxI5bLNGn0nQoOie%2BSNUAPCjnn2Ydoj44l
Sufmur6N2U7paJAuEIfp3VANbLwCI%2Bts5EBr3ecCn7eEqbuQMzs8hW2c%2FdzZqoOvyEda
O86SVcTX9vT5XI8V%2FRluupobCRy8xSuxubnCJrf5USfT%2FB5rudqNkHW0%2BmtE8hxVLI
L9v2dKPSRxtqsU75GsrgA%3D%3D",oauth_body_hash="h3%2BhLMkT%2B3pBvRolKEc95f
obEB8%3D",oauth_version="1.0",oauth_nonce="180286176383600",oauth_signat
ure_method="RSASHA1",oauth_consumer_key="DKB0vGSHs4r1Vv308yObMj4QhhJkIMP
5G3a14KmEa7f96b5e%21414a78536b4a6f6272634a41446e4566483851625a7a413d",
oauth_timestamp="1396020438"
```

The output document is the same as the input but the data is now stored as bytes.

If the Request Header Namespace is hdrns in the NHTTP Emit service, this will add an Authorization header to the HTTP request.

OAuth 2.0 Authentication Service

Syntax:

`com.ibi.agents.XDOAuth2Agent`

Description:

This service emits an HTTPS request authenticated by OAuth 2.0 using the credentials of a Google service account.

OAuth 2.0 is described in RFC6749 and RFC6750. It is an authorization framework that enables an application to obtain access to an HTTP resource. The role of the client and the resource owner are separate. The client does not use the resource owner credentials. Instead, it requests an access token from a trusted authorization server. The client presents the token together with the request to the resource server which grants access if the token is valid.

The OAuth 2.0 Authentication service manages the creation and renewal of access tokens by communicating with the authorization server. If it obtains a valid access token, it incorporates the token with the outgoing HTTPS request to make the authenticated call.

When OAuth 2.0 is used interactively, the user is often redirected to a consent screen to enter his credentials. Since the iSM service operates in a server-to-server scenario, there is no consent screen involved. Instead, the client provides its credentials with the private key associated with a Google service account. The authorization server will be accessed at the same location as the resource server.

When obtaining a token, the client must specify the scope of the access requested. For example, there could be a scope for read-only and another for read-write permissions. It is also possible to request multiple scopes in a single access token.

The scopes are not standardized. The resource servers are free to define the scopes that make sense for their application. The documentation of the resource server API will make it clear which OAuth 2.0 scopes it supports. It remains the responsibility of the application designer to request and use the appropriate scopes.

Parameters:

The following table lists and describes the parameters for the OAuth 2.0 Authentication service.

Parameter	Description
HTTP Client Provider	HTTP Client provider that manages connections for this emitter.
Destination URL	URL where the request will be addressed. The URL should be fully specified, including the HTTPS scheme.

Parameter	Description
HTTP Method	POST sends the current document as a request entity. GET and HEAD will send a request to the configured URL.
Content Type	Content type for the HTTP request to be sent by this emitter.
Request Header Namespace	Special register namespace from which HTTP headers for the outgoing request will be taken. Select *Default Namespace* to send the HDR type registers with no namespace prefix, or supply a namespace prefix here. *None* means that no special registers will be sent as HTTP headers.
Response Header Namespace	Special register namespace into which HTTP headers from the incoming response will be saved. Select *Default Namespace* to create special registers with no namespace prefix, or supply a namespace prefix here.
Scopes	Determines which services the application requests access to. Select one from the list or enter a space-separated list of scopes.
Project ID	Value of the x-goog-project-id header.
Service Account Email	Email address of the service account.
KeyStore Provider	Provider for the keystore containing the service account private key.
Private Key Alias	Alias of the service account private key within the keystore.
Private Key Password	Password for the private key. If left blank, the password for accessing the keystore will be used.

The HTTP Client provider can be defined on the pooling providers page in the iWay Service Manager console. Google recommends that OAuth 2.0 should always be used with HTTPS, therefore the HTTP Client provider should specify an SSLContext Provider.

The authenticated request will be sent to the Destination URL. The service will access the authorization server at the same location to obtain access tokens. The HTTP method specifies the type of request: GET, PUT, POST, HEAD, PATCH or DELETE. The Content Type parameter specifies the content type of the authenticated request.

If additional headers are needed, they can be declared as HDR registers in the Request Header Namespace. The default value is *none*, which means there is no Request Header Namespace and no extra headers are added.

The headers of the HTTP response are stored as special registers in the specified Response Header Namespace.

The Scopes parameter specifies the access scope requested from the authorization server. The access token returned will grant those scopes. For example, if the scope in the token is read-only-access and a write operation is attempted, the resource server will reject the call because the scope is not sufficient. The scopes are application specific. For more information, see the resource server API documentation to learn which scopes it supports.

The Project ID is optional. When present, it specifies the value of the x-goog-project-id header. In the cloud platform of Google, a Project consists of a set of users, a set of APIs, and billing, authentication, and monitoring settings for those APIs.

The Service Account Email acts like a user name for a service account. It looks like an email address but there is no actual email involved. The credential for the service account is a private key. The KeyStore Provider is the name of the iSM keystore provider that holds the private key. The Private Key Alias and the Private Key Password are the alias and password of the private key entry within the Keystore.

Edges:

The following table describes the edges that are returned by the OAuth 2.0 Authentication Service.

Edge	Description
Success	The request message was successfully sent and the response received.
fail_parse	An iFL expression could not be evaluated.
fail_operation	The operation could not be completed successfully.

Example:

This example shows how to retrieve a document from the Google Cloud Storage using the OAuth 2.0 Authentication Service.

Google Cloud Storage is a service to store data in the cloud and retrieve it later. It has a RESTful API, authenticated with OAuth 2.0. The equivalent of a file is called an object. Objects are stored in folders called buckets. Buckets are like directories, except they are not hierarchical. Every bucket exists in a single namespace shared by all users of the service. The slash (/) character is invalid in a bucket name but is accepted in an object name.

The Google Cloud platform supports multiple varieties of credentials. For server to server applications, a service account must be used. This is a name associated with a public key. The client proves his identity by encrypting with the private key which is kept secret. Service accounts are tied to a specific Google project. They are created in the Google Developer Console. Google chooses the service account name and creates the public-private key pair automatically. The private key is downloaded at the time the service account is created.

The Google Developer Console may change without notice. Proceed with the following steps:

From the following website, *https://console.developers.google.com*:

1. Sign in with your Google account or create a new google account on the sign-in page if you do not already have one. This is your regular Google account, not a service account yet.
2. Create a new Google project, specify the project name and the project ID, and then write down the project ID.
3. In the left menu, select *APIs & auth* and then click on *APIs*. Ensure that the Google Cloud Store is set to *ON*.
4. In the left menu, select *APIs & auth*, click *Consent*. and then proceed to the next step. The consent screen is not used in this example, but it must be selected and meet the requirements of Google.
5. Enter any Product Name and click *Save*.
6. In the left menu, select *APIs & auth*, click *Credentials*, and select *Create new Client ID*.
7. In the dialog that appears, select *Service Account* and click *Create client ID*. This will download a PKCS12 Keystore containing the private key of this service account.
8. Save the keystore in a convenient location and write down the keystore password shown on the screen, for example, notasecret.
9. Record the email address of the service account shown on the screen. The email address will look similar to the following:

   ```
   298643775104-81neakmsco3agrv956tl8inu8ci7oedl@developer.gserviceaccount.com
   ```

10. In the left menu, select *Billing & settings*, then enable the billing, and fill in the financial information.

11. In the left menu, click *Storage*, select *Cloud Storage*, *Storage Browser* and then click *Create a bucket*, and enter the bucket name. This must be a unique name among all buckets in the Google Cloud Storage. A good practice is to use your bucket name with your domain name changing the period with a dash. For example: test99-example-com
12. Click *Upload* and select a file to store in the Google Cloud Storage, then write down the name of the bucket and the file name in the bucket.
13. Go to *http://pki.google.com* and download the CA certificate for Google Internet Authority G2. This will be used to create a truststore.

    The following information will be available:

    - ❏ The project ID.
    - ❏ The Keystore containing the private key.
    - ❏ The Keystore password.
    - ❏ The service account email address.
    - ❏ The bucket name.
    - ❏ The object name in the bucket.
    - ❏ The certificate for Google Internet Authority G2.
14. Import the Google CA certificate into a Java Keystore with the following command. This assumes the certificate is stored in a file called GIAG2.cer

```
keytool -import -trustcacerts -file GIAG2.cer -alias giag2 -keystore
GIAG2.jks -storepass secret -storetype JKS
```

15. In the left menu of the iWay Service Manager console, click *Security Providers*, and then click *New* to create a new Keystore provider to be used as the truststore.

    The following table lists the parameters and values of the Keystore provider.

Parameter	Value
Name	giag2
Description	Google Internet Authority G2
Keystore	Path to GIAG2.jks
Keystore Password	secret
Keystore type	JKS

Parameter	Value
KeyStore JCE Provider	SUN

16. Click *New* again to create another Keystore provider for the private key of the service account.

Parameter	Description
Name	cloudkey
Keystore	Path to the keystore containing the service account private key.
Keystore Password	The keystore password chosen by Google, for example, notasecret.
Keystore type	PKCS12-DEF
KeyStore JCE Provider	BC

17. Create an SSLContext provider to be used by the HTTPS connections, and accept all default parameters except the following:

Parameter	Description
Name	GoogleSSL
Keystore Provider	Not used but required. You can enter giag2
Truststore Provider	giag2
Security Protocol	TLS

18. Create an HTTP Client provider to manage connections to the Google Cloud Storage by clicking *Pooling Providers* in the left menu of the iWay Service Manager console, and then clicking *New*.

19. Accept all default parameters except those found in the following table:

Parameter	Description
Name	GoogleClient

Parameter	Description
SSL Context Provider	GoogleSSL

20. In your process flow, create an instance of the OAuth 2.0 Authentication Service configured as shown in the following table. Replace the values in angle brackets with the actual value obtained in the Google Developers console.

Parameter	Description
HTTP Client Provider	GoogleClient
Destination URL	URL of the object in the Google Cloud Storage.
HTTP Method	GET
Scopes	*https://www.googleapis.com/auth/devstorage.read_only*
Project ID	The Google Project ID.
Service Account Email	The service account email address chosen by Google.
KeyStore Provider	cloudkey
Private Key Alias	privatekey
Private Key Password	Password chosen by google, for example, notasecret.

The Destination URL can take one of the following forms:

- *https://<bucketname>.storage.googleapis.com/<objectname>*
- *https://storage.googleapis.com/<bucketname>/<objectname>*

For example, if the bucket name is *mybucket-example-com*, and the object name is *root.xml*, the destination URL can be one of two equivalent URLs:

- *https://mybucket-example-com.storage.googleapis.com/root.xml*
- https://storage.googleapis.com/mybucket-example-com/root.xml

This instance of the OAuth 2.0 Authentication Service accepts any input document since the GET method does not send a request entity in the body of the message. The output document will be the contents of the object found in the Google Cloud Storage or an error document.

WS HTTP Client Agent (com.ibi.agents.XDWSHttpClientAgent)

Syntax:

```
com.ibi.agents.XDWSHttpClientAgent
```

Description:

This service executes a web service through an HTTP Client provider and allows a transformation to be applied to the response document.

Parameters:

Parameter	Description
End point (required)	The location where the web service has been deployed.
SOAP Action	Value of the SOAPAction header for HTTP.
Content-Type	Value of the Content-Type MIME Header. The default value is *text/xml*
User ID	User ID for Basic Authentication challenges.
Password	Password for Basic Authentication challenges.
Timeout	The timeout value in seconds. The default value is *0*
Header	The header of the web service message. This value can be a file name, transform, or actual data with SREG, XPATH, and so on.
Body	The body of the web service message. Can be a file name, transform, or actual data with SREG, XPath, and so on. The default value is *_flatof(false)*
Fault Transform	Transformation to apply when a web service fault occurs.

Parameter	Description
Strip SOAP Envelope	If set to *true*, then the SOAP envelope from the response document is removed. By default, this parameter is set to *false*. For more information on the behavior and usage of this parameter, see the description that follows this table.
Response Transform	Transformation to apply for the web service response document.
HTTP Client Provider (required)	The HTTP Client provider that manages connections for this web service agent.

Strip SOAP Envelope Parameter

The Strip SOAP Envelope parameter moves namespaces to the payload node when a SOAP response is received. For example, assume that the target web service returns the following message to the service:

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:somens="http://somens.com">
      <SOAP-ENV:Body>
          <somens:payload>Hello</somens:payload>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If set to *true*, then the Strip SOAP Envelope parameter cuts the child of the <SOAP-ENV:Body> element, which in this example, is <somens:payload>. The result is not, by itself, a well-formed XML document because the xmlns attribute that declares the somens namespace prefix is lost during the cut.

Enabling the Strip SOAP Envelope parameter causes the service to ascend the XML tree looking for namespace declarations and copies these attributes to the payload node. For example, after the SOAP envelope is removed, the response document has the following structure:

```
<somens:payload xmlns:somens="http://somens.com">Hello</somens:payload>
```

Chapter 4

Configuring HTTP Preparsers

iWay Service Manager includes many predefined preparsers that you can use to convert incoming messages into processable documents. You can add these preparsers to simple or complex business logic configurations using the iWay Service Manager Administration Console.

For reference purposes, this section lists and describes all the predefined preparsers that are supplied with iWay Service Manager.

In this chapter:

HTTP Preparser Configuration Overview

A preparser is designed to convert incoming messages into processable documents. The preparsed document then passes through the standard transformation services to reach the designated processing service. An example of a preparser is a class that accepts an EDI-formatted document and converts it to XML for further processing. For more information on the methodology used in writing preparsers, see the *iWay Service Manager Programmer's Guide*.

Each preparser uses a class file that must be located in a directory which is in the Java classpath. iWay Service Manager includes preparsers that have been preconfigured.

HTTP Preparser (com.ibi.preparsers.XDHTTPpreParser)

Syntax:

```
com.ibi.preparsers.XDHTTPpreParser
```

Description:

The HTTP preparser transforms a key=value query string into an XML string with the GET action as the root. This preparser can be used to construct an XML document with the HTTP parameters that are being passed as a query string. The HTTP preparser can only be used with an HTTP listener.

For example, if you need to generate an XML document with the HTTP request parameters, the HTTP preparser can be used to transform the HTTP GET request for a web application.

Example:

1. Select the *HTTP (com.ibi.preparsers.XDHTTPpreParser)* preparser from the Type drop-down list and click *Next*.

   The configuration parameters pane opens.

2. Enter a name (for example, HttpPreparser) and an optional description.
3. Click *Finish*.
4. Configure an HTTP listener (for example, http) that is listening on port 9984.
5. Construct a simple index.html page. For example:

```
<html xmlns="urn:schemas-microsoft-com:xslt">
<head>
<title>Posting a File</title>
</head>
<body>
<form id ="f" action="http://TestMachine:8080/newproject/
postfile60.html" enctype="multipart/form-data">
<input type="file" name="fdata"/>
</form>
</body>
</html>
```

6. In the Document Root field for the HTTP listener, specify the base directory from which the sample index.html page is served.
7. Construct an inlet, for example, HttpPreparseInlet, which is associated with the HTTP listener (for example, http) and the new preparser (HttpPreparser).
8. Define a simple move route and a default outlet for the channel.
9. Build, deploy, and start the channel.
10. Invoke the following URL from a web browser:

```
http://informat-2a8d8e:9984/index.html?value1=4&value2=two
```

    The following output is obtained:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<post>
<value1>2</value1>
<value2>two</value2>
</post>
```

Multipart for nHTTP Preparser (com.ibi.preparsers.XDMultiPartForNHTTP)

Syntax:

`com.ibi.preparsers.XDMultiPartForNHTTP`

Description:

The Multipart for nHTTP preparser divides multipart documents into body and attachments. This preparser can also be used with an nHTTP listener. This preparser must be last in the chain. When used, the multipart message is broken up so that the body of the multipart message is part 0 and subsequent attachments are 1 to n.

In a use case scenario, the Multipart for nHTTP preparser is useful when the incoming message is a multipart document and the headers and attachment portions of the document need to be extracted from the document for additional message processing.

Parameters:

Parameter	Description
Keep Message Flat	Determines whether to keep the body of the message as an array of bytes. Select true or false from the drop-down list.
Message Header Namespace	A special register namespace where message headers can be found. If not supplied, HDR registers from the default namespace will be used.
Payload Header Namespace	A special register namespace into which any headers on the extracted body part will be stored as HDR registers. If not supplied, body part headers will be saved in the default namespace.

Example:

1. Select the *Multi Part For NHTTP (com.ibi.preparsers.XDMultiPartForNHTTP)* preparser from the Type drop-down list and click *Next*.

   The configuration parameters pane opens.

2. Leave the Payload Header Namespace and Message Header Namespace fields blank.
3. Select *false* from the Keep Message Flat drop-down list and click *Next*.

   The name and description pane opens.

4. Enter a name (for example, nhttpmultipart) and an optional description.
5. Click *Finish*.

6. Construct an inlet, for example, FileInlet, which is associated with a file listener and the new preparser (nhttpmultipart).
7. Define a simple move route.
8. Define an outlet that consists of a File emitter, which emits the output document (outmpart*.xml) to an output directory.
9. Build, deploy, and start the channel.
10. Use a multipart document for the input, which is generated from an email attachment. For example:

```
content-type: multipart/mixed;
boundary="----=_Part_5_23514719.1234266657921"
content-length: 571
mime-version: 1.0
message-id: <17237217.1234266657984.JavaMail.soumya@ibi>
------=_Part_5_23514719.1234266657921
Content-Type: application/xml
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment; filename=a.xml
<a>a</a>
------=_Part_5_23514719.1234266657921
Content-Type: text/document
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment; filename=b.xml
Content-Id: mydata1
<b>b</b>
------=_Part_5_23514719.1234266657921
Content-Type: text/document
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment; filename=c.xml
Content-Id: mydata2
<c>c</c>
------=_Part_5_23514719.1234266657921-
```

When this input document is processed by the channel, an output document (outmpart*.xml) is generated with the following contents:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

Chapter 5

Configuring HTTP Headers and Special Registers

This section provides additional information regarding HTTP headers and special registers.

In this chapter:

HTTP Header and SREG History

In earlier versions, all HTTP receive headers were collected as special registers of type HDR. These registers were examined by the process, which decided whether to create new HDR registers or delete existing registers. When an HTTP emit agent was configured, the user had the option to relay headers. If configured to relay, any registers of type HDR found in the context, with a few specific exceptions, were made into outbound headers. If not configured to relay, HDR registers were ignored.

If the emitted HTTP request document received a response with headers, these headers were then converted into special registers of type HDR.

Issue to be Addressed

The existing technique makes it difficult to distinguish information when there are several HTTP emit agents in a flow, or where it is desired to be able to selectively relay header information in a gateway. Currently, the headers sent are filtered only by a pre-set list of registers not to be sent.

For gateway flows, it is important to be able to group header information and to select and operate upon these groups.

Special Registers and HTTP

Special register names can be preceded with a namespace prefix. The namespace is simply the first part of the name before the first decimal. For example, register *xyz* is in the default namespace, while register *abc.xyz* is in the *abc* namespace. This notation is compatible with registers defined in the iWay Service Manager SP1 registry as well as existing notation. IWAF context registers also follow this pattern, as does the iway.home register, which can be considered as the home in the iWay namespace. Only register names that do not contain any decimals are namespace-less, which is to say in the default namespace.

The HTTP/ASx listener is extended to place all incoming header information into a configured namespace. If no namespace name is specified, the default namespace (none) is be used. The HTTP/ASx emitter and the associated agents have new configuration parameters, which are listed and described in the following table.

Parameter	Role
request namespace	The output headers are taken from the configured namespace. This is a combo box, enabling a user to select a pre-defined setting or enter the name of a specific namespace as the source of headers. Selections and their meaning are: ❑ [listener] - Select from the namespace configured on the listener. **Note:** The [listener] option is not applicable to the listener itself. ❑ [default] - Select from the default namespace. The default namespace is one with no namespace prefix. ❑ [none] - Send no headers from special registers, only send those generated by this emitter.

Parameter	Role
response namespace	If there is a response with headers, then the incoming headers are placed in this namespace. The selections and their meaning are: ❏ [listener] - Use the namespace configured as the response namespace on the listener. **Note:** The [listener] option is not applicable to the listener itself. ❏ [default] - Use default namespace. The default namespace is one with no namespace prefix. ❏ [none] - Create no special registers from the input context, only send those generated internally by this emitter.
Excluded Headers	This is a comma delimited list (case-insensitive) of headers that should not be sent with the response, even if they are found in the response header namespace.

A new agent called XDSREGNamespaceAgent, is available to perform the following operations on registers in namespaces:

- *Copy* - Duplicates registers from a source namespace to a destination namespace. After a copy operation is performed, the registers are available in both namespaces.
- *Move* - Moves registers from one namespace to another.
- *Delete* - Deletes all registers in the namespace.
- *Exist* - If any registers exist in the named namespace, pass the flow down the success edge, else down the notfound edge.

The XDSREGNamespaceAgent offers *from name* and *to name* combo-box parameters. Each of these parameters offers the [listener] and [default] options which are interpreted as described above.

Chapter 6

Configuring Common Parameters

This section provides a reference for common configuration parameters used by iWay Service Manager (iSM) components.

In this chapter:

Listener Configuration Parameters

The following table lists and describes common parameters used by the HTTP, Secure HTTP (HTTPS), and nHTTP Server listeners.

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 *iWay 5.x Compatibility* for compatibility with earlier versions. The compatibility mode is designed to remove newlines in pretty-printed documents while preserving the whitespace in text-only elements.
Accepts non-XML (flat) only	If *true*, the listener expects non-XML (flat). Automatic parsing is not performed.
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*.

Parameter	Description
Multithreading	Indicates the number of documents that can be processed in parallel for this listener. 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 is 1. The max 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.
Execution Time Limit	Time limit for document execution (in seconds) before cancellation is attempted. (Also see system property "Kill Interval". This applies to agent stacks and sets a lower limit for process flows.)
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>.
Always reply to listener default	If set to *true*, the default reply definition is used in addition to defined replies.
Error Documents treated normally	If set to *true*, error documents are processed by any configured preemitters.
Listener is Transaction Manager	If *true*, agents/services run within a local transaction managed by the listener. In this case the process flow serves as the transaction boundary and use of the fail service or an error can trigger a rollback. The message is considered to be acquired and does not roll back to the original source. Use Retry Queues or other application-specific services to handle such a case.
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.
AES Key	If the channel will receive encrypted AFTI messages, set the AES key (maximum 16 characters) to be used for decryption.

Parameter	Description
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 published process flow to run if this channel cannot start or fails during message use. The server will attempt to call this process flow during channel close down due to the error.
Startup Dependencies	A comma-separated list of channel names that must be started before this one is called.

Chapter 7

Configuring iWay Service Manager Components

During the HTTP solutions development process, you are required to configure listeners and services using iWay Service Manager (iSM). This section provides the steps that are needed to access and configure these iSM components. Descriptions of the parameters for each component are provided within the corresponding sections.

In this chapter:

Configuring Listeners

This section describes how to configure listeners using iSM.

Procedure: How to Configure a Listener

1. Ensure that iSM is running.

   On Windows, you can start iSM by clicking *Start*, selecting *Programs*, *iWay 7.0 Service Manager*, and then *Start Service Manager* for the configuration you are currently using.

   For more information on starting and stopping iSM, see the *iWay Service Manager User Guide*.

2. Open a browser window and point to the following URL:

   `http://host:port/ism`

   where:

   host

   Is the host machine on which iSM is installed.

   port

   Is the port on which iSM is listening. The default port is 9999.

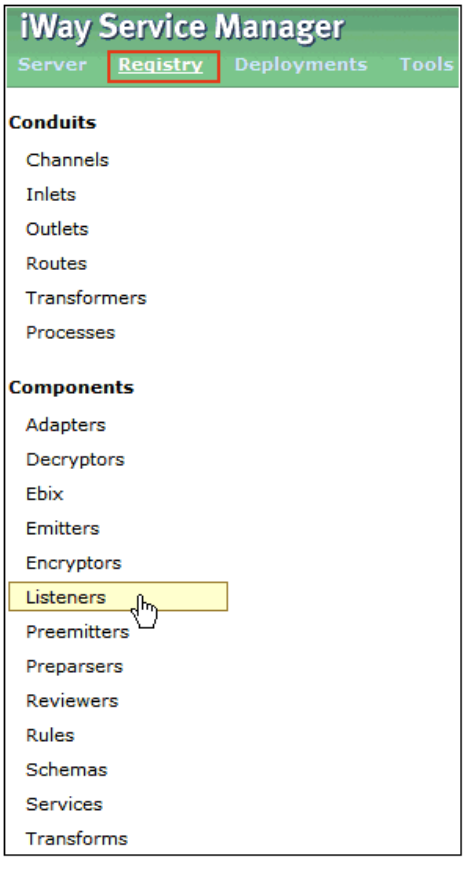
   On Windows, alternatively, you can click *Start*, select *Programs*, *iWay 7.0 Service Manager*, and then click *Console*.

   A logon dialog box opens.

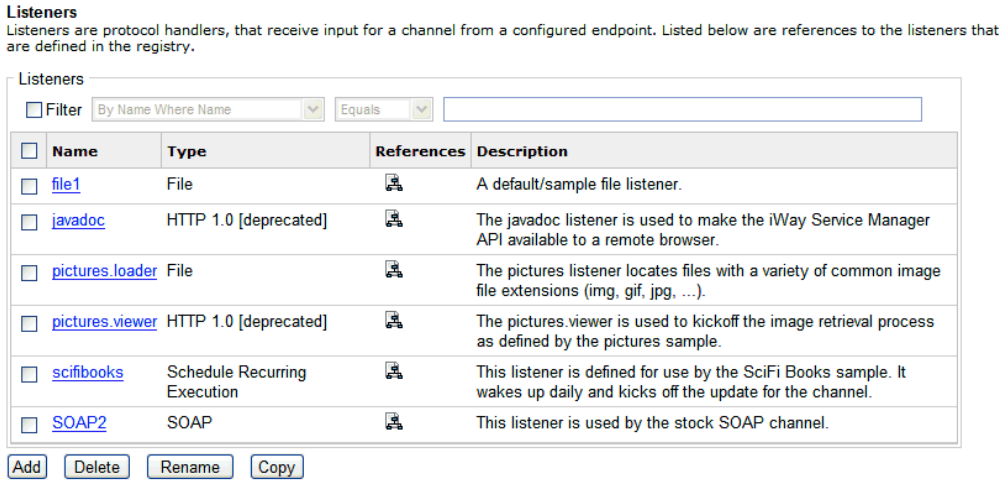
3. Type a user name and password for the configuration you are using, and click *OK*.

The iWay Service Manager Administration Console opens.

4. Click *Registry* in the top pane, and then click *Listeners* in the left pane, as shown in the following image.



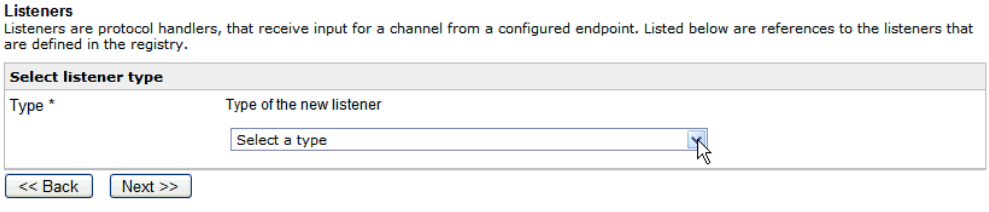
The Listeners pane opens, as shown in the following image.



The table provided lists all the previously configured listeners and a brief description for each.

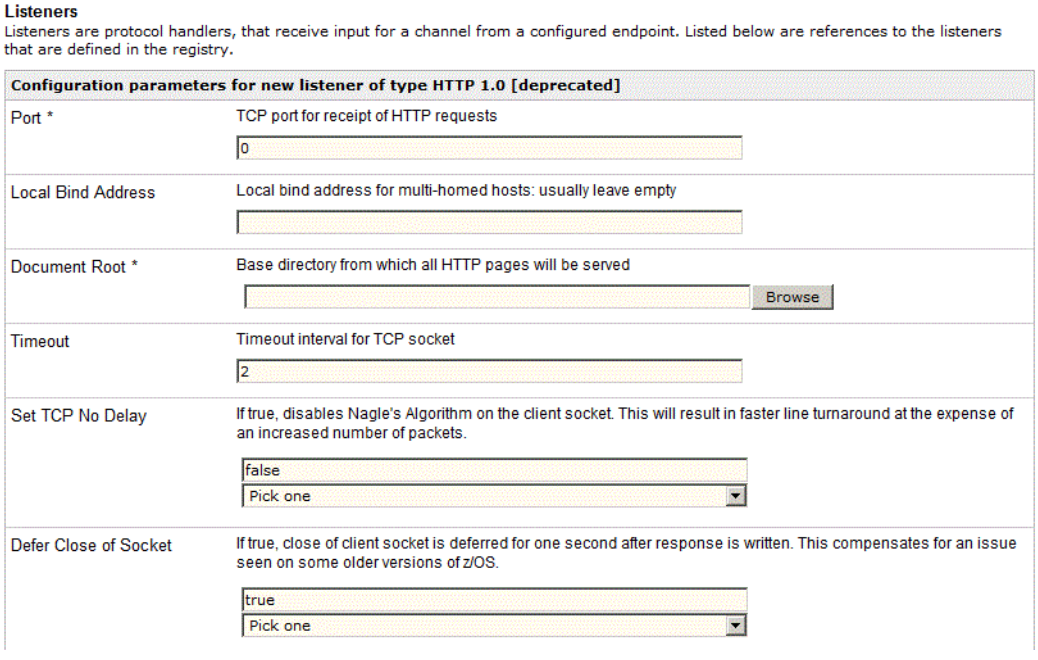
5. Click *Add*.

   The Select listener type pane opens, as shown in the following image.



6. Select the type of listener that you want to configure (for example, *HTTP 1.0 [deprecated]* from the Type drop-down list and click *Next*.

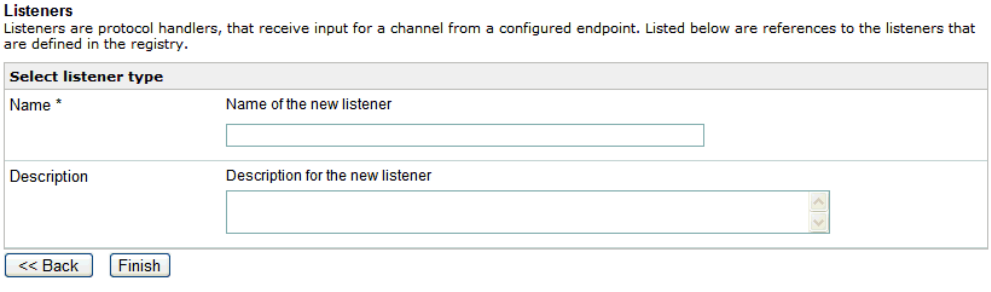
A configuration parameters pane for the selected listener opens. For example, the following image shows a portion of the HTTP 1.0 listener configuration pane.



Note: The parameters indicated with an asterisk (*) in the listener configuration pane are required.

7. Provide the appropriate values for the listener parameters. In most cases, you will need to scroll down the page to view all of the available parameters for the selected listener.
8. Click *Next* at the bottom of the page to continue.

   A listener name and description pane opens, as shown in the following image.



9. Enter a name for the selected listener and a brief description (optional).

10. Click *Finish*.

    You return to the Listeners pane, where the new listener that has been configured is added to the table of available listeners.

    You can use this listener as part of your channel configuration where the business logic will be applied to the received messages.

Configuring Services

This section describes how to configure services using iSM.

Procedure: How to Configure a Service

1. Ensure that iSM is running.

   On Windows, you can start iSM by clicking *Start*, selecting *Programs*, *iWay 7.0 Service Manager*, and then *Start Service Manager* for the configuration you are currently using.

   For more information on starting and stopping iSM, see the *iWay Service Manager User Guide*.

2. Open a browser window and point to the following URL:

   ```
   http://host:port/ism
   ```

   where:

   host

   Is the host machine on which iSM is installed.

   port

   Is the port on which iSM is listening. The default port is 9999.

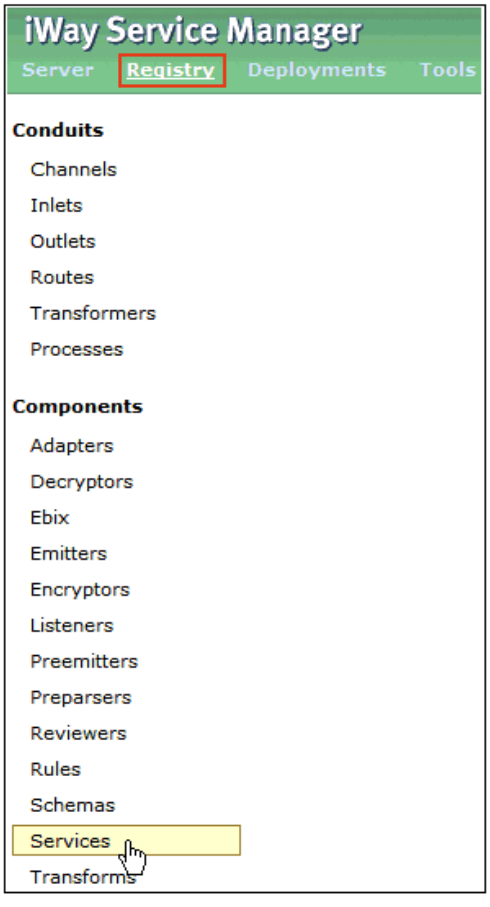
   On Windows, alternatively, you can click *Start*, select *Programs*, *iWay 7.0 Service Manager*, and then click *Console*.

   A logon dialog box opens.

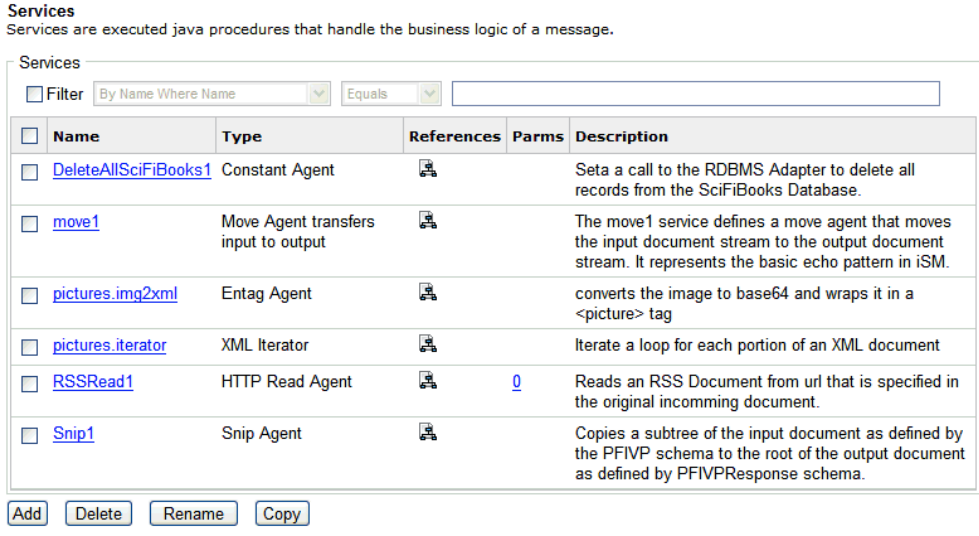
3. Type a user name and password for the configuration you are using, and click *OK*.

   The iWay Service Manager Administration Console opens.

4. Click *Registry* in the top pane, and then click *Services* in the left pane, as shown in the following image.



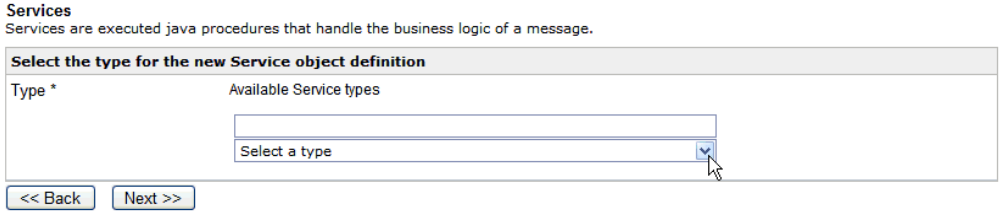
The Services pane opens, as shown in the following image.



The table provided lists all the previously configured services and a brief description for each.

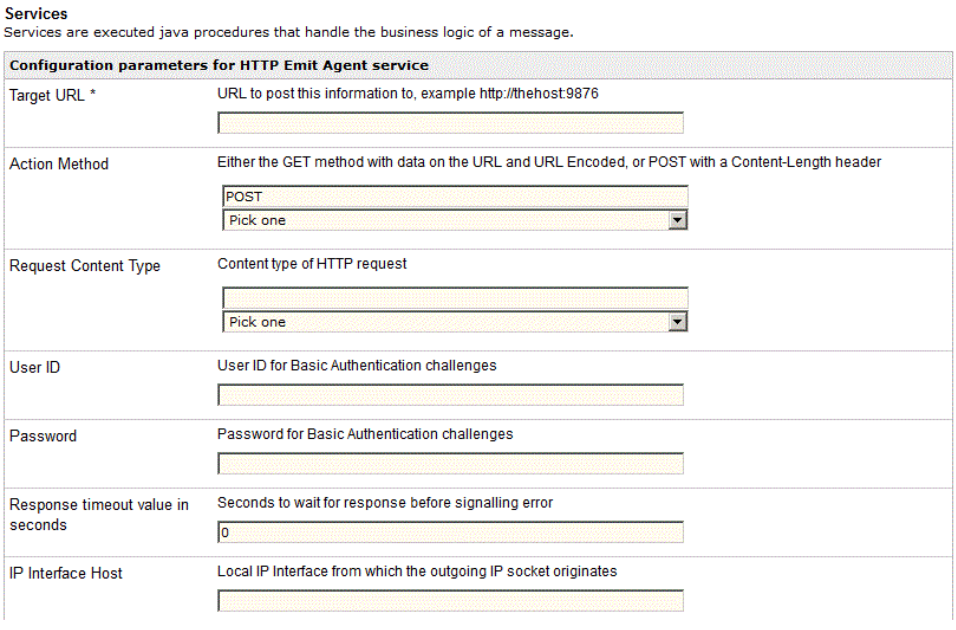
5. Click *Add*.

   A select service type pane opens, as shown in the following image.



6. Select the type of service that you want to configure (for example, *HTTP Emit Agent {com.ibi.agents.XDHTTPEmitAgent}* from the Type drop-down list and click *Next*.

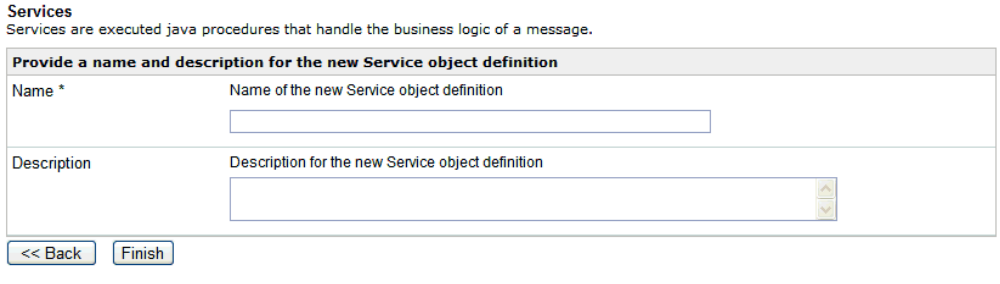
A configuration parameters pane for the selected service opens. For example, the following image shows a portion of the HTTP Emit Agent service configuration pane.



Note: The parameters indicated with an asterisk (*) in the service configuration pane are required.

7. Provide the appropriate values for the service parameters. In most cases, you will need to scroll down the page to view all of the available parameters for the selected service.
8. Click *Next* at the bottom of the page to continue.

   A service name and description pane opens, as shown in the following image.



9. Enter a name for the selected service and a brief description (optional).
10. Click *Finish*.

You return to the Services pane, where the new service that has been configured is added to the table of available services.

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.