Configure SIBus queues and topics in WebSphere Application Server

Write scripts by using wasadmin and Jython

Denis Guillemenot   August 23, 2017

You can configure the SIBus by using the Integrated Solutions Console (ISC) web console, or by using the command line in the wsadmin tool using Jython. But, all the available examples that show how to send/receive messages to SIBus destinations are done in Java only. This tutorial shows how you can do these operations with wsadmin and Jython, which is the preferred scripting language to write scripts to test or administer WebSphere Application Server.

Introduction

> **More information**
> For more information about the SIBus, see the following developerWorks articles:
>
> - Deploying MDB beans and JMS applications into the SIBus
> - Deploying publish and subscribe applications into SIBus.

IBM® WebSphere® Application Server provides a service integration bus (SIBus) where you can create destinations (queues and topics) that you can map to Java™ Messaging Service (JMS) queues and topics. You can configure the SIBus by using the Integrated Solutions Console (ISC) web console, or by using the command line in the wsadmin tool using Jython. But, all the available examples that show how to send/receive messages to SIBus destinations are done in Java only.

This tutorial shows how you can do these operations with wsadmin and Jython, because Jython is the preferred scripting language to write scripts to test or administer WebSphere Application Server. To complete the steps in this tutorial, you must have WebSphere Application Server V8.5 or V9. You can download all of the scripts in this tutorial on GitHub.

Because these steps use Jython only, you don't need to create activation specifications because you will not use a message-driven bean (MDB).

1. Configure the test environment

The test environment consists of a stand-alone WebSphere Application Server V9 instance with one server *server1* on node *Node01*. In this task, you create the following items:

- A SIBus with:
  - One server member, which will contain a MessagingEngine
  - Two SIBus destinations: a queue and a topic
- JMS resources (to work with the SIBus by using the JMS Java API):
  - A connection factory to connect to the SIBus
  - A JMS queue that points to the SIBus destination queue
  - A JMS topic that points to the SIBus destination topic space

The messages that are sent to the queue and topic are stored in a file (default) on the destination.

To configure the test environment:

1. Create a bus instance:
   a. Open the ISC. (The WebSphere server must be running.)
   b. Select **Service Integration -> Buses**, and then click **New**.
   c. For the new bus name, enter `myBus` (with Bus security enabled).
   d. Clear the **Require clients use SSL protected transports** check box.

e. In the next pane, click **Next** to accept the the default option of **Inherit the cell level security domain**. You now see the confirmation pane.

f. Save your changes. You now see the new bus in the

list.

A bus needs at least one bus member to function. The bus members can be one or more application servers, clusters, or both. When a bus member is added, a messaging engine is created in that WebSphere instance, which by default has its own message store for messages. Message store can be a file store (default), which uses the file system, or a data store, which uses a database. The environment for this tutorial has one bus with only one bus member, which is the stand-alone WebSphere Application Server instance.

2. Create a bus member:

a. Select myBus.

b. In the next pane, select **Bus Members**, and then click **Add**.

c. For member, select **Node01:server1**.

d. Select the default File Store option, which is **${USER_INSTALL_ROOT}/filestores/ com.ibm.ws.sib/Node01.server1-MyBus-<ME_UUID>/**.

e. In the next pane, click **Next** to accept the default file store properties.

f. If necessary, accept the modification of the JVM heap size.

g. Click **Finish** to create the new member.

Buses			?	
Buses > myBus > Bus members				
Bus members are the servers, WebSphere MQ servers and clusters that have been added to the bus.				
⊞ Preferences				
Add Remove				
☑ 📋 ⤓ 🏷				
Select	Name ⇕	Type ⇕	Messaging engine policy assistance ⇕	
You can administer the following resources:				
☐	Node01:server1	Server	Not applicable	
Total 1				
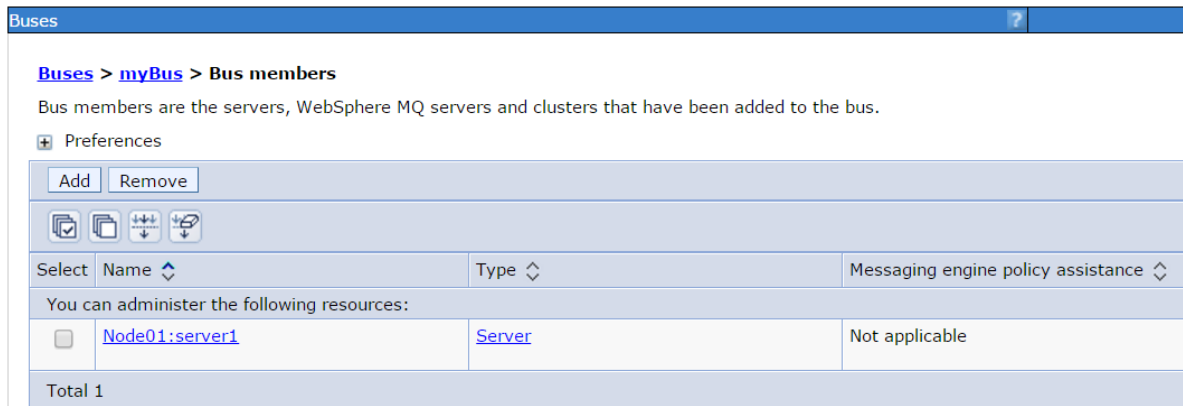
3. Create a queue and a topic destinations:

    a. Select **Service Integration -> Buses -> myBus -> Destinations**.

    b. Click **New**.

    c. Create a queue, called `myBusQueue`, on bus member Node01:server1.

    d. Create a topic space, called `myBusTopic`. (No bus member is required for localization.) The following figure shows the new queue and topic.

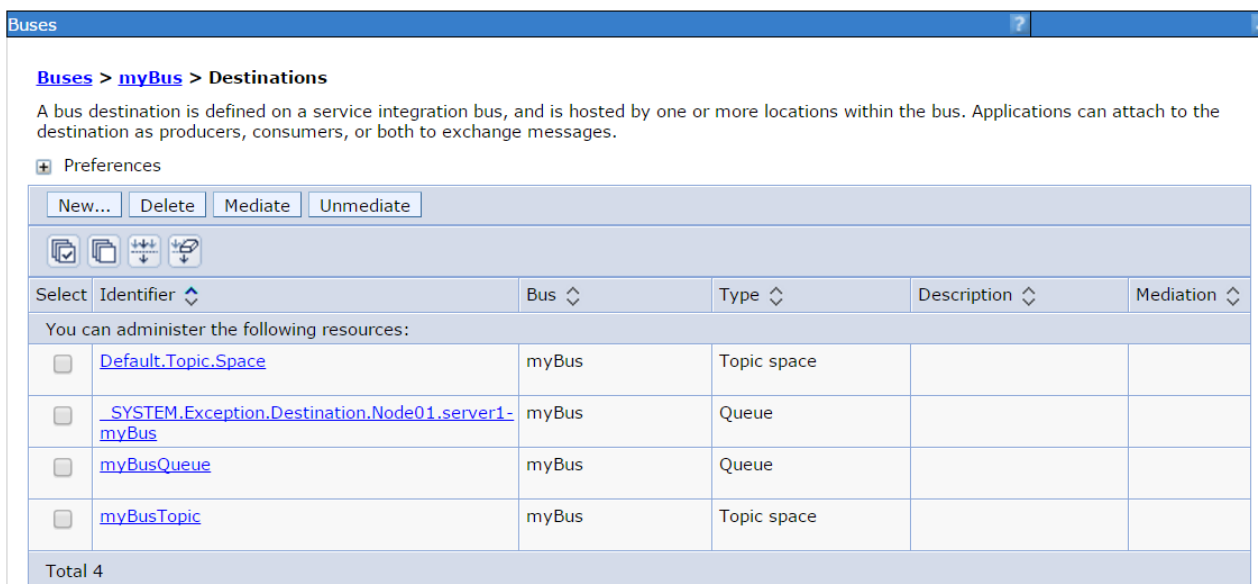| Buses					?	
Buses > myBus > Destinations						
A bus destination is defined on a service integration bus, and is hosted by one or more locations within the bus. Applications can attach to the destination as producers, consumers, or both to exchange messages.						
⊞ Preferences						
New... Delete Mediate Unmediate						
☑ 📋 ⤓ 🏷						
Select	Identifier ⇕	Bus ⇕	Type ⇕	Description ⇕	Mediation ⇕	
You can administer the following resources:						
☐	Default.Topic.Space	myBus	Topic space			
☐	_SYSTEM.Exception.Destination.Node01.server1-myBus	myBus	Queue			
☐	myBusQueue	myBus	Queue			
☐	myBusTopic	myBus	Topic space			
Total 4						

4. Set the security of the bus:

    a. Select **Service integration -> Buses -> myBus -> Security for bus myBus**.

    b. Select **Allow the use of all defined transport channel chains**.

    c. Select users and groups that are in the bus connector role. Click **New**.

    d. Select **The built in special groups**. Click **Next**.

    e. Add **AllAuthenticated**, and click **Next**.

    f. Click **Finish**.

    The security is now configured.

Buses

Buses > myBus > Security for bus myBus > Users and groups in the bus connector role

Users in the bus connector role are able to connect to the bus to perform messaging operations. Users can have this role either by specifically having that role, or because they are in a group with that role.

⊞ Preferences

| New... | Delete |

Select	Name ↕	Type ↕
☐	AllAuthenticated	Group
☐	Server	Group
Total 2		

5. Create the JMS resources, starting with a connection factory:

  a. Select **Resources -> JMS -> Connection factories**.

  b. Select the **Node01:server1** scope.

  c. Click **New**.

  d. In the Administration box, for Provider, select **Default messaging provider**. For Name enter `myConnectionFactory`, with a JNDI name of `jms/myConnectionFactory`.

Connection factories

Connection factories > Default messaging provider > New...

A JMS connection factory is used to create connections to the associated JMS provider of JMS destinations, for both point-to-point and publish/subscribe messaging. Use connection factory administrative objects to manage JMS connection factories for the default messaging provider.

Configuration

General Properties

Administration

Scope

Node=Node01,Server=server1

Provider

Default messaging provider

✱ Name

myConnectionFactory

✱ JNDI name

jms/myConnectionFactory

Description

Category

Connection

✱ Bus name

myBus ▾

The additional properties will not be available until the general properties for this item are applied or saved.

Additional Properties

▪ Connection pool properties

Related Items

▪ JAAS - J2C authentication data

▪ Buses

  e. In the Connection box, for Bus name, select **myBus**.

f. For Provider endpoints, enter `<hostname>:7276:BootstrapBasicMessaging,`
`<hostname>:7286:BootstrapSecureMessaging.`

The ports are SIB_ENDPOINT_ADDRESS (port 7276) and
SIB_ENDPOINT_SECURE_ADDRESS (port 7286). You can also check the ports by
selecting **Service Integration -> Buses -> myBus -> Bootstrap members**.
Important: These two endpoints are mutually exclusive. Depending on the bus security
configuration, you can only use one:
- If security *is* enabled, only BootstrapSecureMessaging is used.
- If security *is not* enabled, only BootstrapBasicMessaging is used.

g. Click **OK** to create the connection factory.

6. Create a JMS queue:
    a. Select **Resources -> JMS -> Queues**.
    b. Select the **Node01:server1** scope.
    c. Click **New**.
    d. Select **Default messaging provider**. For queue name, enter `myQueue`, and for JNDI
       name, enter `jms/myQueue`.

e. Set Bus to `myBus`, and set the queue target to `myBusQueue`.

Scope specifies the level at which the resource definition is visible. For detailed information on what
scope is and how it works, see the scope settings help.

Node=Node01, Server=server1 ▼

⊞ Preferences

| New | Delete |

Select	Name ⬍	JNDI name ⬍	Provider ⬍	Description ⬍	Scope ⬍
You can administer the following resources:					
☐	myQueue	jms/myQueue	Default messaging provider		Node=Node01,Server=server1
Total 1					

7. Create a JMS topic:
   a. Select **Resources -> JMS -> Topics**.
   b. Select the **Node01:server1** scope.
   c. Click **New**.
   d. Select **Default messaging provider**. For topic name, enter `myTopic`, and for JNDI name,
      enter `jms/myTopic`.
   e. Set Bus to `myBus`, and set the topic target to `myBusTopic`.

Node=Node01, Server=server1 ▼

⊞ Preferences

| New | Delete |

Select	Name ⬍	JNDI name ⬍	Provider ⬍	Description ⬍	Scope ⬍
You can administer the following resources:					
☐	myTopic	jms/myTopic	Default messaging provider		Node=Node01,Server=server1
Total 1					

8. For server1, verify that SIBservice is enabled at startup. SIBservice is enabled by default. If it
   is not enabled:
   a. Select **Servers -> Server Types -> WebSphere application Servers -> server1 ->
      Server messaging -> SIB service**.
   b. Select **Enable service at server startup**.
   c. If the Messaging Engine on server1 is not running, start it. Select **Service Integration ->
      Buses -> myBus -> Messaging engines**.

You can now use the test environment.

2. Activate SIBus traces to gather timing data

Before you start to code in Jython, you can activate SIBus traces to gather time and date
information for messages that are sent, published, read, or consumed in the SIBus queues and
topics.

For message timing information, you need the `*=info:`
`com.ibm.ws.sib.processor.utils.UserTrace=all` trace setting. For information about the trace
settings, see Tracing WebSphere's Service Integration Bus For Your Own Use.

In the generated traces, for each JMS message, you can see the following details:

- CWSJU at the start of all messages
- The JMS message ID (JMSMessageID)
- The SIBus system message ID (JMS_IBM_System_MessageID)
- The SIBus destination name (queue or topic space)

Listing 1 shows the trace when a message is sent to a SIBus queue.

Listing 1. Trace when a message is sent to a queue

```
[3/14/17 11:12:36:084 CET] 000000bf UserTrace 3 (com.ibm.ws.sib.processor.utils.UserTrace) [:] CWSJU0002I: A
 producer CA5C8F27FAAC58D5688E1352 sent a message with ID ID:7e97f69d7de579d9c8f97491110a134f0000000000000001
 and correlation ID null to destination myBusQueue.
[3/14/17 11:12:36:084 CET] 000000bf UserTrace 3 (com.ibm.ws.sib.processor.utils.UserTrace) [:] CWSJU0004I: A
 message with ID ID:7e97f69d7de579d9c8f97491110a134f0000000000000001, system message ID null and correlation
 ID null is put to queue myBusQueue
[3/14/17 11:12:36:093 CET] 000000bf UserTrace 3 (com.ibm.ws.sib.processor.utils.UserTrace) [:/6173fd2a]
 CWSJU0003I: A message with ID ID:7e97f69d7de579d9c8f97491110a134f0000000000000001, system message ID
 B745072DF9E084A0_6000001 and correlation ID null is committed to destination myBusQueue, which is targetted
 for messaging engine Node01.server1-myBus.
```

Listing 2 shows the trace when a message is read from a SIBus queue.

Listing 2. Trace when a message is read (consumed) from a queue

```
[3/14/17 11:13:50:280 CET] 000000c0 UserTrace 3 (com.ibm.ws.sib.processor.utils.UserTrace) [:]
 CWSJU0041I: A message with ID ID:7e97f69d7de579d9c8f97491110a134f0000000000000001, system message ID
 B745072DF9E084A0_6000001 and correlation ID null was delivered to a consumer 2 from destination myBusQueue
```

Listing 3 shows the trace when a message is published to a SIBus topic.

Listing 3. Trace when a message is published to a topic

```
[3/14/17 11:15:26:711 CET] 000000c0 UserTrace 3 (com.ibm.ws.sib.processor.utils.UserTrace) [:] CWSJU0002I: A
 producer C02338B0F0A455FC355CD217 sent a message with ID ID:09789a17744d69aa2f79a7fd110a134f0000000000000001
 and correlation ID null to destination myBusTopic.
[3/14/17 11:15:26:711 CET] 000000c0 UserTrace 3 (com.ibm.ws.sib.processor.utils.UserTrace) [:] CWSJU0005I: A
 message with ID ID:09789a17744d69aa2f79a7fd110a134f0000000000000001 and correlation ID null is put to topic
 space myBusTopic
[3/14/17 11:15:26:717 CET] 000000c0 UserTrace 3 (com.ibm.ws.sib.processor.utils.UserTrace) [:] CWSJU0008I:
 A message with ID ID:09789a17744d69aa2f79a7fd110a134f0000000000000001 on destination myBusTopic matched 1
 consumers
[3/14/17 11:15:26:723 CET] 000000c0 UserTrace 3 (com.ibm.ws.sib.processor.utils.UserTrace) [:/232782af]
 CWSJU0003I: A message with ID ID:09789a17744d69aa2f79a7fd110a134f0000000000000001, system message ID
 B745072DF9E084A0_6000002 and correlation ID null is committed to destination myBusTopic, which is targetted
 for messaging engine Node01.server1-myBus.
```

Listing 4 shows the trace when a message is consumed from a SIBus topic.

Listing 4. Trace when a message is consumed from a topic

```
[3/14/17 11:16:33:112 CET] 000000dd UserTrace 3 (com.ibm.ws.sib.processor.utils.UserTrace) [:]
 CWSJU0041I: A message with ID ID:09789a17744d69aa2f79a7fd110a134f0000000000000001, system message ID
 B745072DF9E084A0_6000002 and correlation ID null was delivered to a consumer 3 from destination myBusTopic
```

Now that you have set up the SIBus environment, you can use the wsadmin tool with Jython to:

- Inspect the SIBus MBean objects.
- Send and receive messages to the JMS queue and topic by using the JMS Java API.
- Browse the SIBus destinations (queue and topic) messages with the SIBus MBeans.

3. Inspect the SIBus MBean obects

> **Jython scripts**
> You can download the Jython scripts for this tutorial on GitHub.

Launch the wsadmin tool as shown in Listing 5. Check the Jython version and the line separator.

Listing 5. Launching wsadmin

```
$ cd <WAS_PROFILE_HOME>/bin
$ ./wsadmin.sh -lang jython
WASX7209I: Connected to process "server1" on node Node01 using SOAP connector;
The type of process is: UnManagedProcess
WASX7031I: For help, enter: "print Help.help()"

wsadmin> print sys.version
2.7.0 (default:9987c746f838, Apr 29 2015, 02:25:11)
[IBM J9 VM (IBM Corporation)]

wasadmin> lineSeparator
u'\r\n'
```

WebSphere Application Server V9 uses Jython V2.7 instead of V2.1 as in previous versions of WebSphere Application Server. To see how you can adapt your existing Jython scripts, see Jython 2.7 behavior changes.

Listing 6 shows the SIBus MBeans (`type=SIB*`) that are available for `server1`. The output is formatted for readability. In this listing, queues with names that start with `_PSIMP` or `_PTRM` are for SIBus internal use only. This destination contains messages that are not well formatted. If necessary, you can set messages in `_SYSTEM.Exception.Destination` to be consumed for reprocessing or delete them.

The following types of MBeans are available for `server1`:

- **SIBMain**. Start or stop MessagingEngine
- **SIBus**. Provides statistics
- **SIBMessagingEngine**. List queue points, read messages, or manage transactions
- **SIBJMSResource**. Provides statistics
- **SIBQueuePoint**. For queues; the number of messages; and list, read, clear, or flush messages
- **SIBPublicationPoint**. For topic spaces; list subscriptions; the number of messages; and list, read, clear, or flush messages

Listing 6. SIBus MBeans available for server1

```
wsadmin> print AdminControl.queryNames('type=SIB*,process=server1,*')
WebSphere:name=Default.Topic.Space,
```

```
  process=server1,platform=dynamicproxy,node=Node01,
  SIBus=myBus,version=9.0.0.0,ID=29,
  type=SIBPublicationPoint,
  mbeanIdentifier=...,cell=Cell01,spec=1.0,
  SIBMessagingEngine=Node01.server1-myBus

WebSphere:name=MyBus,process=server1,platform=dynamicproxy,
  node=Node01,version=9.0.0.0,
  type=SIBus,
  mbeanIdentifier=...

WebSphere:name=Node01.server1-MyBus,
  process=server1,platform=dynamicproxy,node=Node01,version=9.0.0.0,
  type=SIBMessagingEngine,
  mbeanIdentifier=...

WebSphere:name=SIBMain,
  process=server1,platform=dynamicproxy,node=Node01,version=9.0.0.0,
  type=SIBMain,
  mbeanIdentifier=...

WebSphere:name=_PSIMP.PROXY.QUEUE_840C30BEFC7DF9E7,
  process=server1,platform=dynamicproxy,node=Node01,
  SIBus=myBus,version=9.0.0.0,ID=C9B54E5F9B0762FD7FF66A4B_QUEUE_8,
  type=SIBQueuePoint,
  mbeanIdentifier=...,cell=Cell01,spec=1.0,
  SIBMessagingEngine=Node01.server1-myBus

WebSphere:name=_PSIMP.TDRECEIVER_840C30BEFC7DF9E7,
  process=server1,platform=dynamicproxy,node=Node01,
  SIBus=myBus,version=9.0.0.0,ID=2DCFB5C22B0940597C679EC4_QUEUE_40,
  type=SIBQueuePoint,
  mbeanIdentifier=...,cell=Cell01,spec=1.0,
  SIBMessagingEngine=Node01.server1-myBus

WebSphere:name=_PTRM_840C30BEFC7DF9E7,
  process=server1,platform=dynamicproxy,node=Node01,
  SIBus=myBus,version=9.0.0.0,ID=2687C492B5FB1505476DB04B_QUEUE_46,
  type=SIBQueuePoint,
  mbeanIdentifier=...,cell=Cell01,spec=1.0,
  SIBMessagingEngine=Node01.server1-myBus

WebSphere:name=_SYSTEM.Exception.Destination.Node01.server1-MyBus,
  process=server1,platform=dynamicproxy,node=Node01,
  SIBus=myBus,version=9.0.0.0,ID=77A2990358FFE651908947E5_QUEUE_14,
  type=SIBQueuePoint,
  mbeanIdentifier=...,cell=Cell01,spec=1.0,
  SIBMessagingEngine=Node01.server1-myBus

WebSphere:name=com.ibm.ws.sib.admin.impl.SIBJMSResource,
  process=server1,platform=dynamicproxy,node=Node01,
  j2eeType=JMSResource,J2EEServer=server1,version=9.0.0.0,
  type=SIBJMSResource,
  mbeanIdentifier=...,cell=Cell01,spec=1.0

WebSphere:name=myBusQueue,
  process=server1,platform=dynamicproxy,node=Node01,
  SIBus=myBus,version=9.0.0.0,ID=4D6A95886439F4CD88E16D0C_QUEUE_20,
  type=SIBQueuePoint,
  mbeanIdentifier=...,cell=Cell01,spec=1.0,
  SIBMessagingEngine=Node01.server1-myBus

WebSphere:name=myBusTopic,
  process=server1,platform=dynamicproxy,node=Node01,
  SIBus=myBus,version=9.0.0.0,ID=22,
  type=SIBPublicationPoint,
  mbeanIdentifier=...,cell=Cell01,spec=1.0,
```

```
  SIBMessagingEngine=Node01.server1-myBus

WebSphere:name=server1,
  process=server1,platform=common,node=Node01,version=9.0.0.0,
  type=SIBMQResourceDiscovery,
  mbeanIdentifier=null,cell=Cell01,spec=1.0
```

For the complete list of MBeans (and their Javadocs), including additional SIB and JMS MBeans, see the "WebSphere Application Server Public MBean Interfaces" topic in the WebSphere Application Server Network Deployment traditional 9.0.0.x documentation in IBM Knowledge Center.

Listing 7 and the table that follows it show some of the methods that are provided by these MBeans. Listing 7 shows that the SIBus MBean gives access to only statistics data.

Listing 7. Methods and attributes for the SIBus MBean

```
wsadmin> sibus = AdminControl.queryNames('type=SIBus,process=server1,*')
wsadmin> print sibus
  WebSphere:name=myBus,
  process=server1,platform=dynamicproxy,node=Node01,version=9.0.0.0,
  type=SIBus,
  mbeanIdentifier=...,cell=Cell01,spec=1.0

wsadmin> print Help.operations( sibus)
Operation
javax.management.j2ee.statistics.Stats getStats()

wsadmin> print Help.attributes( sibus)
Attribute                      Type                                     Access
stats                          javax.management.j2ee.statistics.Stats   RO
```

With the wsadmin Help object methods, you can find the methods and attributes of the other MBeans.

Table 1. Attributes and methods of the MBeans

MBean	Attributes	Methods

SIBMain	`EventTypes`
	`stats`
		`showMessagingEngines()`
		`startMessagingEngine( ...)`
		`stopMessagingEngine( ...)`
		`stopMessagingEngine( ...)`
		`getEventTypes()`
		`getStats()`
		`...`

SIBMessagingEngine	`EventTypes`
	`stats`
		`GetDepth( ...)`
		`isStarted()`
		`state()`
		`start()`
		`stop()`
		`listQueuePoints()`
		`getQueuePoint( ...)`
		`getQueuePointMessages( ...)`
		`getQueuePointMessageDetail( ...)`
		`getHealth()`
		`dump( String)`
		`getStats()`
		`...`

SIBQueuePoint	`depth`
	`state`
	`id`
	`identifier`
	`highMessageThreshold`
	`sendAllowed`
		`getDepth()`
		`getState()`
		`getQueuedMessages()`
		`getQueuedMessageDetail( ...)`
		`getMessageData( ...)`
		`deleteQueuedMessage( ...)`
		`deleteAllQueuedMessages( ...)`
		`getId()`
		`getIdentifier()`
		`…`

SIBPublicationPoint	`depth`
	`id`
	`identifier`
	`highMessageThreshold`
	`sendAllowed`
		`getDepth()`
		`getSubscriptions()`
		`deleteSubscription( ...)`
		`getSubscriptionMessages( ...)`
		`getSubscriptionMessageData( ...)`
		`deleteSubscriptionMessage( ...)`
		`getRequestMessageDetail( ...)`
		`getRequestMessageData( ...)`
		`republishMessages( ...)`
		`getId()`
		`getIdentifier()`
		`...`

For the SIBMessagingEngine, SIBQueuePoint, and SIBPublicationPoint MBean, you can read messages. However, they do not provide methods to send or publish messages.

4. Connect to the SIBus with Jython via JMS

You must connect most of the Jython scripts in this tutorial to the SIBus via JMS:

1. Import some Java classes (Listing 8).

Listing 8. Importing JMS APIs

```
wsadmin> import javax.naming
wsadmin> import javax.jms
wsadmin> import javax.naming.Context

wsadmin> dir( javax.naming.Context)
['APPLET', 'AUTHORITATIVE', 'BATCHSIZE', 'DNS_URL', 'INITIAL_CONTEXT_FACTORY', 'LANGUAGE',
 'OBJECT_FACTORIES', 'PROVIDER_URL', 'REFERRAL', '
SECURITY_AUTHENTICATION', 'SECURITY_CREDENTIALS', 'SECURITY_PRINCIPAL', 'SECURITY_PROTOCOL',
 'STATE_FACTORIES', 'URL_PKG_PREFIXES', 'addToEn
vironment', 'bind', 'close', 'composeName', 'createSubcontext', 'destroySubcontext', 'environment',
 'getEnvironment', 'getNameInNamespace',
'getNameParser', 'list', 'listBindings', 'lookup', 'lookupLink', 'nameInNamespace', 'rebind',
 'removeFromEnvironment', 'rename', 'unbind']
```

2. Initialize a javax.naming.Context to connect to the SIBus:
   - The PROVIDER_URL uses the BOOTSTRAP_ADDRESS (2811) of WebSphere Application Server.
     Important: If the BOOTSTRAP_ADDRESS of the JVM is different, the script cannot connect to the SIBus. Therefore, check the list of ports for your server to see whether they differ from the default value.
   - SECURITY_AUTHENTICATION is set to simple. If bus security is not enabled, use the default authentication of none.

In Listing 9, a context is initialized.

Listing 9. Initialized context

```
wsadmin> h = java.util.Hashtable()
wsadmin> h[ javax.naming.Context.INITIAL_CONTEXT_FACTORY] =
 "com.ibm.websphere.naming.WsnInitialContextFactory"
wsadmin> h[ javax.naming.Context.PROVIDER_URL] = "corbaloc:iiop:localhost:2811"
wsadmin> h[ javax.naming.Context.SECURITY_AUTHENTICATION] = "simple"
wsadmin> h[ javax.naming.Context.SECURITY_PRINCIPAL] = "wasadmin"
wsadmin> h[ javax.naming.Context.SECURITY_CREDENTIALS] = "admin"
wsadmin> initcontext = javax.naming.InitialContext( h)
```

You can obtain the initial context methods with the Java introspection getClass() and getMethods(), as shown in Listing 10 (not all methods are shown).

Listing 10. Initial context methods

```
wsadmin> for i in initcontext.getClass().getMethods(): print( i)
…
public Object lookup( String) throws javax.naming.NamingException
…
public Hashtable getEnvironment() throws javax.naming.NamingException
public void close() throws javax.naming.NamingException
public String getNameInNamespace() throws javax.naming.NamingException
…

wsadmin> initcontext.getNameInNamespace()
'Cell01/nodes/Node01/servers/server1'
```

3. Connect to the SIBus. Use a JMS connection factory and queue, as shown in Listing 11.

Listing 11. Find connection factory and destination queue

```
wsadmin> factory = initcontext.lookup("jms/myConnectionFactory")
wsadmin> factory
com.ibm.ws.sib.api.jms.impl.JmsConnectionFactoryImpl@2728d095

wsadmin> destination = initcontext.lookup('jms/myQueue')
wsadmin> destination
queue://myBusQueue?busName=myBus
```

Because a direct JNDI lookup is used and bus security is enabled, pass the user ID and password to connect to the bus by using the connection factory createConnection(...) method. For more information, see the following articles:

- SINotAuthorizedException when a JMS application attempts to access SIBus
- Direct and indirect JNDI lookup methods for data sources

Bus security and SSL are two different levels (and kinds) of security. In general, most clients use bus security for applications that are deployed into WebSphere Application Server that connect to the Bus. They use SSL for application connections outside of WebSphere Application Server. If Bus security is disabled, you do not need to authenticate with the createConnection() method. No authentication is required. For applications that are deployed into WebSphere Application Server, no bus security is the most common configuration. Listing 12 shows how to pass a user credential to create the connection.

Listing 12. Create a connection

```
wsadmin> connection = factory.createConnection( 'wasadmin', 'admin')

wsadmin> connection
ConnectionId: 1bbddcf, MEName: Node01.server1-MyBus, Sessions: 0, TemporaryDestinations: 0
```

This tutorial uses the following connection methods:

- createSession(…)
- start()
- stop()
- close()

4. Create a session as shown in Listing 13.

Listing 13. Create a session

```
wsadmin> session = connection.createSession( java.lang.Boolean('false'),
 javax.jms.Session.AUTO_ACKNOWLEDGE)

wsadmin>session
com.ibm.ws.sib.api.jms.impl.JmsSessionImpl@1cd082d
```

Again, the following session methods are used:

- createTextMessage(…)
- createProducer(…)
- createConsumer(…)

- createDurableSubscriber(…)

Listing 14 resumes the steps to initialize the connection context to connect to the SIBus and initialize some variables that are needed for the scripts shown in Listings 16, 22, 27, and 32. This listing was created in a wsadmin session. However, for readability, wsadmin prompts have been removed.

Listing 14. Context initialization

```
# import JMS APIs
import javax.naming
import javax.jms
import javax.naming.Context

# initialize variables
was_port = 2811
was_user = 'wasadmin'
was_pswd = 'admin'

connection = None
session = None

# define context intitializatin function
def init_context( cf_name, dest_name, client_id = ''):
    '''
        cf_name : connection factory as in (jms/cf_name)
        dest_name: destination queue or topic as in (jms/dest_name)
        client_id: for reading topic only
    '''
    global destination, connection, session
    #
    # initialize context
    h = java.util.Hashtable()
    h[ javax.naming.Context.INITIAL_CONTEXT_FACTORY] =
 "com.ibm.websphere.naming.WsnInitialContextFactory"
    h[ javax.naming.Context.PROVIDER_URL] = "corbaloc:iiop:localhost:%s" % was_port
    h[ javax.naming.Context.SECURITY_AUTHENTICATION] = "simple"
    h[ javax.naming.Context.SECURITY_PRINCIPAL] = was_user
    h[ javax.naming.Context.SECURITY_CREDENTIALS] = was_pswd
    initcontext = javax.naming.InitialContext( h)
    #
    # search connection factory
    factory = initcontext.lookup( 'jms/%s' % cf_name)
    # search destination: queue or topic
    destination = initcontext.lookup( 'jms/%s' % dest_name)
    #
    # close JNDI context
    initcontext.close()
    #
    # create connection and session for destina
    connection = factory.createConnection( was_user, was_pswd)
    if client_id:
      connection.setClientID('%s' % client_id)
    session = connection.createSession( java.lang.Boolean('false'), javax.jms.Session.AUTO_ACKNOWLEDGE)
    # start connection (needed reading, not for sending)
    connection.start()
    #
    return destination
```

The next step shows how to interact with SIBus queues and topics by using the wsadmin tool and Jython with the SIBus MBeans or the JMS Java API.

5. Send and receive a message

To send and receive a message, you need Jython scripts for SIBus queues and for SIBus topic spaces.

Jython scripts for SIBus queues

To interact with SIBus queues, you can use Jython scripts. The Jython scripts in the following table are provided as examples. Note the following explanation of the script name:

- ***_soap.py** The script needs the WebSphere Application Server instance up and running.
- ***_jms_*.py** The script name connects to the JMS resources through the JMS API. If the script does not have this designation, the script accesses the SIBus directly through the SIBus MBeans and the wsadmin `AdminControl` object.

Table 2. Jython scripts for SIBus queues

Name	Parameters	Object
`sibus_destinations_depth_soap.py`	`-`	Display SIBus destinations depth (queue and topics)
`sibus_queue_browser_soap.py`	`BusName` `QueueName`	Browse messages in a SIBus queue

`sibus_queue_jms_reader_soap.py`	`ConnectionFactory`
	`QueueName`
	`'all'`
		Consume one (or all) messages on a JMS queue

`sibus_queue_jms_writer_soap.py`	`ConnectionFactory`
	`QueueName`
	`text message`
		Emit a text message on a JMS queue

Send a text message to a SIBus queue

To display the SIBus destinations depth, you can use the `sibus_destinations_depth_soap.py` script as shown in Listing 15.

Listing 15. Script to display the SIBus destination depth

```
$ ./wsadmin.sh -lang jython -f sibus_destinations_depth_soap.py
#--- JMS Connections to SIBus -----------------------------

        1 to myBus

          myConnectionFactory (jms/myConnectionFactory)

         1 to defaultBus

            built-in-jms-connectionfactory (jms/built-in-jms-connectionfactory)

#--- SIBus Queue(s) Depth -- [SIBus:queue_name (JMS JNDI)] -----------------------------

        0 messages in myBus:_PSIMP.PROXY.QUEUE_B745072DF9E084A0
        0 messages in myBus:_PSIMP.TDRECEIVER_B745072DF9E084A0
        0 messages in myBus:_PTRM_B745072DF9E084A0
```

```
        0 messages in myBus:_SYSTEM.Exception.Destination.Node01.server1-myBus
        0 messages in myBus:myBusQueue (jms/myQueue)

#--- SIBus Topic(s) Depth -- [SIBus:topicspace_name (JMS JNDI)] -----------------------------

        0 messages in myBus:Default.Topic.Space
        0 messages in myBus:myBusTopic (jms/myTopic)
```

Listing 16 shows the methods to send a text message to a SIBus queue via JMS as shown in the
`sibus_queue_jms_writer_soap.py` script.

Listing 16. Script to send a JMS message to a SIBus queue

```
# initialize connection to SIBus with
destination = init_context( 'myConnectionFactory', 'myQueue')

# write message to queue
qsender = session.createProducer( destination)
qsender.send( session.createTextMessage( 'Hello World'))

connection.close()
session.close()
```

Listing 17 shows that, the first time that the script is launched, it generates the `sibus_conf.py` file.
You can update this file to adapt to your environment user ID, password, and port.

Listing 17. Script to generate the configuration file

```
$ ./wsadmin.sh -lang jython -f sibus_queue_jms_writer_soap.py myConnectionFactory myQueue hello world
#-----------------------------
INFO: File 'sibus_conf.py' can be used to set 'was_user', 'was_pswd' and 'was_port'
Generating one for you !

    sibus_conf.py:
        was_user = 'wasadmin'
        was_pswd = 'admin'
        # (BOOTSTRAP_ADDRESS)
        was_port = 2811
```

If needed, modify the `sibus_conf.py` script. Then, launch again the script to send a text message
to the SIBus queue via JMS, as shown in Listing 18.

Listing 18. Script to send a JMS message to a SIBus queue

```
$ ./wsadmin.sh -lang jython -f sibus_queue_jms_writer_soap.py myConnectionFactory myQueue
#-----------------------------
INFO: Using WAS user 'wasadmin'and port '2811' defined in 'sibus_conf.py'...

Writing message to [jms/myQueue]
Message is: Hello World
```

In Listing 19, notice that the depth of the queue is now set to 1.

Listing 19. Checking the depth of the SIBus queue

```
$ ./wsadmin.sh -lang jython -f sibus_destinations_depth_soap.py
#--- JMS Connections to SIBus -----------------------------
```

```
     1 to myBus

       myConnectionFactory (jms/myConnectionFactory)

     1 to defaultBus

       built-in-jms-connectionfactory (jms/built-in-jms-connectionfactory)

#--- SIBus Queue(s) Depth -- [SIBus:queue_name (JMS JNDI)] ----------------------------

     0 messages in myBus:_PSIMP.PROXY.QUEUE_B745072DF9E084A0
     0 messages in myBus:_PSIMP.TDRECEIVER_B745072DF9E084A0
     0 messages in myBus:_PTRM_B745072DF9E084A0
     0 messages in myBus:_SYSTEM.Exception.Destination.Node01.server1-myBus
     1 messages in myBus:myBusQueue (jms/myQueue)

#--- SIBus Topic(s) Depth -- [SIBus:topicspace_name (JMS JNDI)] ----------------------------

     0 messages in myBus:Default.Topic.Space
     0 messages in myBus:myBusTopic (jms/myTopic)
```

Browse a SIBus queue

The content of the SIBus queue is available from the ISC. Select **Service Integration → Buses**, and select your bus. Click **Destinations** and select your **queue**. Click **Queue points**, and select your **queue point**. Then, under **runtime**, select the **messages** property.

You can get almost the same result by using the ISC when you look for the message body of any available messages in the queue.

You can also browse the SIBus queue by using Jython and wsadmin as shown in Listing 20.

Listing 20. Browse a SIBus queue with wsadmin

```
# get SIBQueuePoint of our queue
wsadmin> sibqueue = AdminControl.queryNames('type=SIBQueuePoint,process=server1,name=myBusQueue,*')

# get queue depth
wsadmin> AdminControl.invoke( sibqueue, 'getDepth')
'3'

# get queue id
wsadmin> sibqueue_id = AdminControl.getAttribute( sibqueue, 'id')
wsadmin> sibqueue_id
'1F28997FD5BF8B06BF764920_QUEUE_20'

# get SIBMessagingEngine
wsadmin> sibme = AdminControl.queryNames('type=SIBMessagingEngine,process=server1,*')
wsadmin> sibme_obj = AdminControl.makeObjectName( sibme)

# get all messages from our queue
wsadmin> messages = AdminControl.invoke_jmx( sibme_obj, 'getQueuePointMessages',[sibqueue_id],
 ['java.lang.String'])
wsadmin> messages
array([com.ibm.ws.sib.admin.impl.SIBQueuedMessageImpl@1e805ee,
 com.ibm.ws.sib.admin.impl.SIBQueuedMessageImpl@1e80651], com.ibm.websphere.si
b.admin.SIBQueuedMessage)

for msg in messages:
     msg_id = msg.getId()
     msg_sysid = msg.getSystemMessageId()
```

```
    msg_length = msg.getApproximateLength()
    msg_detail = AdminControl.invoke_jmx( sibmeobj, 'getQueuePointMessageData',[sibqueueid, msg_id, 100],
 ['java.lang.String','java.lang.String', 'java.lang.Integer'])
    print( '-'*10 )
    print( ' id: %s' % msg_id)
    print( ' sysid: %s' % msg_sysid)
    print( 'length: %d' % msg_length)
    print( ' hexa: %s' % ':'.join( [ '%02x' % x for x in msg_detail]))
    print( ' ascii: %s' % ''.join( [ chr(x) for x in msg_detail]))
----------
id: 2000001
sysid: B745072DF9E084A0_1000002
length: 636
Hexa: 48:65:6c:6c:6f:20:57:6f:72:6c:64
Ascii: Hello World
```

Listing 21 shows how to use the `sibus_queue_browser_soap.py` script.

Listing 21. Browsing message in a SIBus queue

```
$ ./wsadmin.sh -lang jython -f sibus_queue_browser_soap.py myBus myBusQueue
#----------------------------
Browsing messages from SIBus Queue destination [myBus:myBusQueue]

1 message(s) available

[msgId:2000001 (636)]:
    array('b', [72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100])
    Hello World
```

Read a message from a SIBus queue via JMS

Listing 22 shows the methods to receive a message from a SIBus queue via JMS as shown in the `sibus_queue_jms_reader_soap.py` script.

Listing 22. Script to consume a JMS message from a SIBus queue

```
# initialize connection to SIBus with
destination = init_context( 'myConnectionFactory', 'myQueue')

# create queue reader
reader = session.createConsumer( destination)

# read the queue
qmessage = reader.receiveNoWait()
if qmessage:
  print( qmessage)

connection.close()
session.close()
```

Listing 23 shows how to use the `sibus_queue_jms_reader_soap.py` script to read a text message from a SIBus queue via JMS.

Listing 23. Reading a JMS message from a SIBus queue

```
$ ./wsadmin.sh -lang jython -f sibus_queue_jms_reader_soap.py myConnectionFactory myQueue
#----------------------------
INFO: Using WAS user 'wasadmin'and port '2811' defined in 'sibus_conf.py'...
```

```
Reading message from [jms/myQueue]

    JMSMessage class: jms_text
    JMSType: null
    JMSDeliveryMode: 2
    JMSExpiration: 0
    JMSPriority: 4
    JMSMessageID: ID:6545bd82ec22841a2eee5eb7110a134f0000000000000001
    JMSTimestamp: 1488534915285
    JMSCorrelationID: null
    JMSDestination: queue://myBusQueue?busName=myBus
    JMSReplyTo: null
    JMSRedelivered: false
    JMSDeliveryTime: 1488534915285
        JMSXDeliveryCount: 1
        JMSXUserID: wasadmin
        JMS_IBM_System_MessageID: B745072DF9E084A0_1000002
        JMSXAppID: Service Integration Bus
Hello World
```

You can check now that the SIBus queue depth is 0 again (Listing 24).

Listing 24. SIBus queue is empty

```
$ ./wsadmin.sh -lang jython -f sibus_destinations_depth_soap.py
#--- JMS Connections to SIBus -----------------------------

        1 to myBus

          myConnectionFactory (jms/myConnectionFactory)

        1 to defaultBus

          built-in-jms-connectionfactory (jms/built-in-jms-connectionfactory)

#--- SIBus Queue(s) Depth -- [SIBus:queue_name (JMS JNDI)] -----------------------------

      0 messages in myBus:_PSIMP.PROXY.QUEUE_B745072DF9E084A0
      0 messages in myBus:_PSIMP.TDRECEIVER_B745072DF9E084A0
      0 messages in myBus:_PTRM_B745072DF9E084A0
      0 messages in myBus:_SYSTEM.Exception.Destination.Node01.server1-myBus
      0 messages in myBus:myBusQueue (jms/myQueue)

#--- SIBus Topic(s) Depth -- [SIBus:topicspace_name (JMS JNDI)] -----------------------------

      0 messages in myBus:Default.Topic.Space
      0 messages in myBus:myBusTopic (jms/myTopic)
```

Jython scripts for SIBus topic spaces

To interact with SIBus topic spaces, the following Jython scripts are provided as examples.

Table 3. Jython scripts to interact with the SIBus topic spaces

Name	Parameters	Object
sibus_destinations_depth_soap.py	-	Display SIBus destinations depth (queue and topics)
sibus_topic_browser_soap.py	BusName TopicName	Browse messages in a SIBus topic space

sibus_topic_jms_durable_subscriber_s	ConnectionFactory TopicName client_id subscriber_id 'create' | 'delete'	Manage topic durable subscribers
sibus_topic_jms_reader_soap.py	ConnectionFactory TopicName client_id subscriber_id 'all' wait_time	Consume one (or all) message(s) on a JMS topic using a durable subscriber or not
sibus_topic_jms_writer_soap.py	ConnectionFactory TopicName text message	Publish a text message on a JMS topic

Create a durable subscriber to a SIBus topic

Now you create a durable subscriber (client *client01* and subscriber *sub01*) to the SIBus topic space. This way, the message that is sent to the topic can be kept until the consumer script is launched to read the message. Listing 25 shows how to use the `sibus_topic_jms_durable_subscribers_soap.py` script to create a durable subscriber.

Listing 25. Creating a durable subscriber to the SIBus topic

```
$ ./wsadmin.sh -lang jython -f sibus_topic_jms_durable_subscriber_soap.py myConnectionFactory myQueue
 client01 sub01 create
#----------------------------
INFO: Using WAS user 'wasadmin'and port '2811' defined in 'sibus_conf.py'...
#----------------------------

#----------------------------
create subscriber [client01##sub01] for Topic [myTopic]
```

You can see the durable subscriber when you display the SIBus destinations depth as shown in Listing 26.

Listing 26. Showing the SIBus destination depth

```
$ ./wsadmin.sh -lang jython -f sibus_destinations_depth_soap.py
#--- JMS Connections to SIBus -----------------------------

     1 to myBus

       myConnectionFactory (jms/myConnectionFactory)

     1 to defaultBus

       built-in-jms-connectionfactory (jms/built-in-jms-connectionfactory)

#--- SIBus Queue(s) Depth -- [SIBus:queue_name (JMS JNDI)] ----------------------------

     0 messages in myBus:_PSIMP.PROXY.QUEUE_B745072DF9E084A0
     0 messages in myBus:_PSIMP.TDRECEIVER_B745072DF9E084A0
     0 messages in myBus:_PTRM_B745072DF9E084A0
     0 messages in myBus:_SYSTEM.Exception.Destination.Node01.server1-myBus
     0 messages in myBus:myBusQueue (jms/myQueue)

#--- SIBus Topic(s) Depth -- [SIBus:topicspace_name (JMS JNDI)] ----------------------------
```

```
      0 messages in myBus:Default.Topic.Space
      0 messages in myBus:myBusTopic (jms/myTopic)
        0 messages for subscriber client01##sub01
```

Send a text message to a SIBus topic

Listing 27 shows how to publish a text message to a SIBus topic space via JMS as shown in the
`'sibus_topic_jms_writer_soap.py` script.

Listing 27. Script to publishing a JMS message to a SIBus topic

```
# initialize connection to SIBus with
destination = init_context( 'myConnectionFactory', 'myTopic')

# write message to topic
tsender = session.createProducer( destination)
tsender.send( session.createTextMessage( 'Hello World'))

session.close()
connection.close()
```

In Listings 16 and 27, the code is the same to send a message to a SIBus queue or topic via the
JMS API.

Listing 28 shows how to publish a text message to the SIBus topic by using the
`sibus_topic_jms_writer_soap.py` script.

Listing 28. Publishing a JMS message to a SIBus topic

```
$ ./wsadmin.sh -lang jython -f sibus_topic_jms_writer_soap.py myConnectionFactory myTopic Hello World
#-----------------------------
INFO: Using WAS user 'wasadmin'and port '2811' defined in 'sibus_conf.py'...

Writing message to [jms/myTopic]
Message is: Hello World
```

The depth of the subscriber for the topic is 1 as shown in Listing 29.

Listing 29. The SIBus destination depth

```
$ ./wsadmin.sh -lang jython -f sibus_destinations_depth_soap.py
#--- JMS Connections to SIBus -----------------------------

      1 to myBus

        myConnectionFactory (jms/myConnectionFactory)

      1 to defaultBus

        built-in-jms-connectionfactory (jms/built-in-jms-connectionfactory)

#--- SIBus Queue(s) Depth -- [SIBus:queue_name (JMS JNDI)] -----------------------------

      0 messages in myBus:_PSIMP.PROXY.QUEUE_B745072DF9E084A0
      0 messages in myBus:_PSIMP.TDRECEIVER_B745072DF9E084A0
      0 messages in myBus:_PTRM_B745072DF9E084A0
      0 messages in myBus:_SYSTEM.Exception.Destination.Node01.server1-myBus
      0 messages in myBus:myBusQueue (jms/myQueue)
```

```
#--- SIBus Topic(s) Depth -- [SIBus:topicspace_name (JMS JNDI)] -----------------------------

    0 messages in myBus:Default.Topic.Space
    1 messages in myBus:myBusTopic (jms/myTopic)
      1 messages for subscriber client01##sub01
```

Browse the SIBus topic

The content of the SIBus topic is available from the ISC. Click **Buses**, and select your bus. Click **Destinations**, and select your **topic space**. Click **Subscription points**, and select your subscription. Click **Subscriptions**, and select your subscription. Then, under **runtime**, select the **messages** property.

Listing 30 shows the code to browse SIBus topic messages with Jython and SIBus MBeans. Notice that AdminControl.invoke() returns a string, and AdminControl.invoke_jmx() returns an array.

Listing 30. Browsing the SIBus topic with Jython and SIBus MBeans

```
# get SIBPublicationPoint of our topic
wsadmin> sibtopic = AdminControl.queryNames('type=SIBPublicationPoint,process=server1,name=myBusTopic,*')

# get topic depth
wsadmin> AdminControl.invoke( sibtopic, 'getDepth')
'1'

# get topic id
wsadmin> sibtopic_id = AdminControl.getAttribute( sibtopic, 'id')
wsadmin> sibtopic_id
'14000001'

# get SIBPublicationPoint object
wsadmin> sibtopic_obj = AdminControl.makeObjectName( sibtopic)

# get topic subscriptions
wsadmin> subs = AdminControl.invoke( sibtopic, 'getSubscriptions').split( lineSeparator)
wsadmin> subs
 ['com.ibm.ws.sib.admin.impl.SIBSubscriptionImpl@dcf2fbaf']

# same with invoke_jmx
wsadmin> subs_obj = AdminControl.invoke_jmx( sibtopic_obj, 'getSubscriptions',[],[])
wsadmin> subs_obj
array([com.ibm.ws.sib.admin.impl.SIBSubscriptionImpl@7255752e], com.ibm.websphere.sib.admin.SIBSubscription)

wsadmin> type( subs_obj)
<jclass org.python.core.PyArray at 26558124>

# get informations on first topic subscriber
wsadmin> subs_obj[0].getIdentifier()
 'myBusTopic'
wsadmin> subs_obj[0].getName()
 'myBusTopic'
wsadmin> subs_obj[0].getId()
 'DDA30B2D628875CBA57BF89C'
wsadmin> subs_obj[0].getTopics()
array(['myTopic'], java.lang.String)

# browse messages of our topic for each subscribers
wasadmin> for msg in messages:
    msg_id = msg.getId()
    msg_sysid = msg.getSystemMessageId()
    msg_length = msg.getApproximateLength()
```

```
    msg_detail = AdminControl.invoke_jmx( sibtopic_obj, 'getSubscriptionMessageData',
[sub.getId(),msg.getId(),100], ['java.lang.String','java.lang.String', 'java.lang.Integer'])
    print( '-'*10 )
    print( 'subName: %s' % sub.getName())
    print( ' subId: %s' % sub.getSubscriberId())
    print( ' id: %s' % msg_id)
    print( ' sysid: %s' % msg_sysid)
    print( ' length: %d' % msg_length)
    print( ' hexa: %s' % ':'.join( [ '%02x' % x for x in msg_detail]))
    print( ' ascii: %s' % ''.join( [ chr(x) for x in msg_detail]))

----------
subName: myBusTopic
    subId: client01##sub01
        id: 2000004
     sysid: 0AC32A3495A1AF48_1000004
   length: 772
      hexa: 54:6f:70:69:63:3a:20:48:65:6c:6c:6f:20:57:6f:72:6c:64
     ascii: Topic: Hello World
```
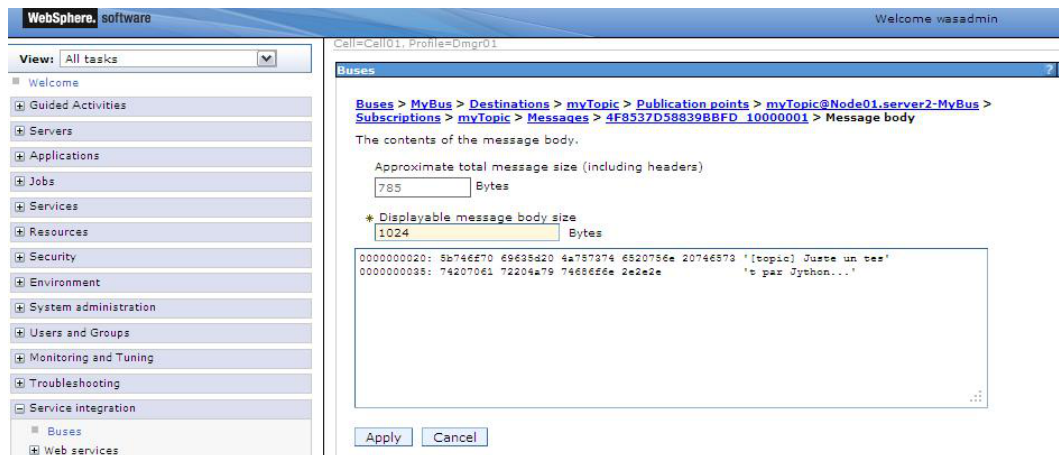
You can achieve almost the same result by using the ISC when you look for the message body of any available messages in the topic subscriber.

The contents of the SIBus topic are also available for each subscriber by using the `sibus_topic_browser_soap.py` script as shown in Listing 31.

Listing 31. Browsing messages in a SIBus topic

```
$ ./wsadmin.sh -lang jython -f sibus_topic_browser_soap.py myBus myBusTopic
#-----------------------------
Browing message(s) from SIBus Topic destination [myBus:myBusTopic]

1 message(s) available

subscriber: myBusTopic## client01##sub01 (depth: 1)

    [msgId:3000006 (668)]:
        array('b', [72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100])
        Hello World
```

Read a message from a SIBus topic via JMS as a subscriber

Listing 32 shows the methods to consume a message from a SIBus topic space via JMS as shown in the `sibus_topic_jms_reader_soap.py` script with a durable or nondurable subscriber.

Listing 32. Script to read a JMS message from a SIBus topic

```
# initialize connection to SIBus with client ID
destination = init_context( 'myConnectionFactory', 'myTopic', 'client01')

# set session subscriber
subscriber = session.createDurableSubscriber( destination, 'sub01')

# read first available message only
tmessage = subscriber.receiveNoWait()
if tmessage:
  print( tmessage)

session.close()
connection.close()
```

Listing 33 shows how to use the `sibus_topic_jms_reader_soap.py` script to read a text message from a SIBus topic via JMS.

Listing 33. Reading a JMS message from a SIBus topic (subscriber consumer)

```
$ ./wsadmin.sh -lang jython -f sibus_topic_jms_reader_soap.py myConnectionFactory myTopic client01 sub01
#-----------------------------
INFO: Using WAS user 'wasadmin'and port '2811' defined in 'sibus_conf.py'...

Reading message from [jms/myTopic]
(client_id ## subscriber_id) : client01##sub01

   JMSMessage class: jms_text
   JMSType: null
   JMSDeliveryMode: 2
   JMSExpiration: 0
   JMSPriority: 4
   JMSMessageID: ID:a3fee3f448aecba2eb395479110a134f0000000000000001
   JMSTimestamp: 1488811732252
   JMSCorrelationID: null
   JMSDestination: topic://myTopic?topicSpace=myBusTopic&busName=myBus
   JMSReplyTo: null
   JMSRedelivered: false
   JMSDeliveryTime: 1488811732252
        JMSXDeliveryCount: 1
        JMSXUserID: wasadmin
        JMS_IBM_System_MessageID: B745072DF9E084A0_1500005
        JMSXAppID: Service Integration Bus
Hello World

Read 1 messages
```

Now you can check that the SIBus queue depth is 0 again (Listing 34).

Listing 34. SIBus topic is empty

```
C:\IBM\WebSphere\AppServer9\profiles\AppSrv01>bin\wsadmin.bat -lang jython -f
 sibus_destinations_depth_soap.py
#--- JMS Connections to SIBus -----------------------------

        1 to myBus

            myConnectionFactory (jms/myConnectionFactory)

        1 to defaultBus

            built-in-jms-connectionfactory (jms/built-in-jms-connectionfactory)
```

```
#--- SIBus Queue(s) Depth -- [SIBus:queue_name (JMS JNDI)] -----------------------------

        0 messages in myBus:_PSIMP.PROXY.QUEUE_B745072DF9E084A0
        0 messages in myBus:_PSIMP.TDRECEIVER_B745072DF9E084A0
        0 messages in myBus:_PTRM_B745072DF9E084A0
        0 messages in myBus:_SYSTEM.Exception.Destination.Node01.server1-myBus
        0 messages in myBus:myBusQueue (jms/myQueue)

#--- SIBus Topic(s) Depth -- [SIBus:topicspace_name (JMS JNDI)] -----------------------------

        0 messages in myBus:Default.Topic.Space
        0 messages in myBus:myBusTopic (jms/myTopic)
         0 messages for subscriber client01##sub01
```

Delete a durable subscriber from a SIBus topic

Listing 35 shows how to use the `sibus_topic_jms_durable_subscribers_soap.py` script to delete a durable subscriber.

Listing 35. Deleting a durable subscriber from the SIBus topic

```
./wsadmin.sh -lang jython -f sibus_topic_jms_durable_subscriber_soap.py myConnectionFactory myTopic client01
 sub01 delete
#------------------------------
INFO: Using WAS user 'wasadmin'and port '2811' defined in 'sibus_conf.py'...
#------------------------------

#------------------------------
delete subscriber [client01##sub01] for Topic [myTopic]
```

Listing 36 shows that the subscriber does not exist anymore.

Listing 36. Checking the deletion of the durable subscriber

```
C:\IBM\WebSphere\AppServer9\profiles\AppSrv01>bin\wsadmin.bat -lang jython -f
 sibus_destinations_depth_soap.py
#--- JMS Connections to SIBus -----------------------------

        1 to myBus

          myConnectionFactory (jms/myConnectionFactory)

        1 to defaultBus

          built-in-jms-connectionfactory (jms/built-in-jms-connectionfactory)

#--- SIBus Queue(s) Depth -- [SIBus:queue_name (JMS JNDI)] -----------------------------

        0 messages in myBus:_PSIMP.PROXY.QUEUE_B745072DF9E084A0
        0 messages in myBus:_PSIMP.TDRECEIVER_B745072DF9E084A0
        0 messages in myBus:_PTRM_B745072DF9E084A0
        0 messages in myBus:_SYSTEM.Exception.Destination.Node01.server1-myBus
        0 messages in myBus:myBusQueue (jms/myQueue)

#--- SIBus Topic(s) Depth -- [SIBus:topicspace_name (JMS JNDI)] -----------------------------

        0 messages in myBus:Default.Topic.Space
        0 messages in myBus:myBusTopic (jms/myTopic)
```

Read a message from a SIBus topic via JMS when not a subscriber

To read a message that is sent to a SIBus topic, a consumer without a subscription must be connected when the message is published. To test this:

1. In a command line (CLI) console, launch the consumer reading `sibus_topic_jms_reader_soap.py` script with the options shown in Listing 37.
2. In another CLI console, publish a message to the SIBus topic by using the `sibus_topic_jms_writer_soap.py` script.

Listing 37 shows the `sibus_topic_jms_reader_soap.py` script to read a text message from a SIBus topic via JMS if the consumer is not a subscriber.

Listing 37. Reading a JMS message from a SIBus topic (nonsubscriber consumer)

```
C:\IBM\WebSphere\AppServer9\profiles\AppSrv01>bin\wsadmin.bat -lang jython -f sibus_topic_jms_reader_soap.py
 myConnectionFactory myTopic clientXX

#-----------------------------
INFO: Using WAS user 'wasadmin'and port '2811' defined in 'sibus_conf.py'...

Reading message from [jms/myTopic]
client_id: client02
Wait time: 30
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

  JMSMessage class: jms_text
  JMSType: null
  JMSDeliveryMode: 2
  JMSExpiration: 0
  JMSPriority: 4
  JMSMessageID: ID:ad6d131b629990307434d639110a134f0000000000000001
  JMSTimestamp: 1490091298087
  JMSCorrelationID: null
  JMSDestination: topic://myTopic?topicSpace=myBusTopic&busName=myBus
  JMSReplyTo: null
  JMSRedelivered: false
  JMSDeliveryTime: 1490091298087
      JMSXDeliveryCount: 1
      JMSXUserID: wasadmin
      JMS_IBM_System_MessageID: B745072DF9E084A0_8500001
      JMSXAppID: Service Integration Bus
Hello World

15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
Read 1 messages
```

The message is read by the consumer, which is not a subscriber.

6. Run all the tests for SIBus queues and topics

A script is available to run most of the scripts that are presented in this article.

Table 4. Script to run scripts in article

Name	Parameters	Object
`sibus_tests_jms_soap.py`	-	Run all the queues and topics scripts

Listing 38 shows the output of the test script. The test script shows how to:

- Create a topic subscriber
- Check the SIBus destinations depths (after each step)
- Send a JMS message to the SIBus queue myBusQueue (jms/myQueue)
- Read the previous message from myBusQueue
- Send a JMS message to the SIBus topic myBusTopic (jms/myTopic)
- Read the previous message from myBusTopic
- Delete the topic subscriber

Listing 38. Run all the scripts

```
$ ./wsadmin.sh -lang jython -f sibus_tests_jms_soap.py

topic_id: id.U9I881 topic_sub: sub.4X4QO8

#-----------------------------
INFO: Using WAS user 'wasadmin'and port '2811' defined in 'sibus_conf.py'...
#-----------------------------

#-----------------------------
create subscriber [id.U9I881##sub.4X4QO8] for Topic [myTopic]

Finished.


#--- JMS Connections to SIBus -----------------------------

        1 to myBus

          myConnectionFactory (jms/myConnectionFactory)

#--- SIBus Queue(s) Depth -- [SIBus:queue_name (JMS JNDI)] -----------------------------

        0 messages in myBus:_PSIMP.PROXY.QUEUE_0AC32A3495A1AF48
        0 messages in myBus:_PSIMP.TDRECEIVER_0AC32A3495A1AF48
        0 messages in myBus:_PTRM_0AC32A3495A1AF48
        0 messages in myBus:_SYSTEM.Exception.Destination.Node01.server1-myBus
        0 messages in myBus:myBusQueue (jms/myQueue)

#--- SIBus Topic(s) Depth -- [SIBus:topicspace_name (JMS JNDI)] -----------------------------

        0 messages in myBus:Default.Topic.Space
        0 messages in myBus:myBusTopic (jms/myTopic)
        0 messages for subscriber id.U9I881##sub.4X4QO8

Writing message 0

#-----------------------------
INFO: Using WAS user 'wasadmin'and port '2811' defined in 'sibus_conf.py'...

Writing message to [jms/myQueue]
Message is: Queue: test message 0

Finished.

#-----------------------------
INFO: Using WAS user 'wasadmin'and port '2811' defined in 'sibus_conf.py'...

Writing message to [jms/myTopic]
Message is: publisher Topic: test message 0

Finished.
```

```
#--- JMS Connections to SIBus ------------------------------

        1 to myBus

           myConnectionFactory (jms/myConnectionFactory)

#--- SIBus Queue(s) Depth -- [SIBus:queue_name (JMS JNDI)] ------------------------------

        0 messages in myBus:_PSIMP.PROXY.QUEUE_0AC32A3495A1AF48
        0 messages in myBus:_PSIMP.TDRECEIVER_0AC32A3495A1AF48
        0 messages in myBus:_PTRM_0AC32A3495A1AF48
        0 messages in myBus:_SYSTEM.Exception.Destination.Node01.server1-myBus
        1 messages in myBus:myBusQueue (jms/myQueue)

#--- SIBus Topic(s) Depth -- [SIBus:topicspace_name (JMS JNDI)] ------------------------------

        0 messages in myBus:Default.Topic.Space
        1 messages in myBus:myBusTopic (jms/myTopic)
           1 messages for subscriber id.U9I881##sub.4X4QO8

Reading messages 0

#------------------------------
INFO: Using WAS user 'wasadmin'and port '2811' defined in 'sibus_conf.py'...

Reading message from [jms/myQueue]
Reading all messages

    JMSMessage class: jms_text
    JMSType: null
    JMSDeliveryMode: 2
    JMSExpiration: 0
    JMSPriority: 4
    JMSMessageID: ID:5c44fb2e692723e0da2ba17f110a134f0000000000000001
    JMSTimestamp: 1489506224376
    JMSCorrelationID: null
    JMSDestination: queue://myBusQueue?busName=myBus
    JMSReplyTo: null
    JMSRedelivered: false
        JMSXDeliveryCount: 1
        JMS_IBM_System_MessageID: 0AC32A3495A1AF48_3500001
        JMSXUserID: uid=wasadmin,o=defaultWIMFileBasedRealm
        JMSXAppID: Service Integration Bus
Queue: test message 0

Finished.

#------------------------------
INFO: Using WAS user 'wasadmin'and port '2811' defined in 'sibus_conf.py'...

Reading message from [jms/myTopic]
(client_id ## subscriber_id) : id.U9I881##sub.4X4QO8
Reading all messages

    JMSMessage class: jms_text
    JMSType: null
    JMSDeliveryMode: 2
    JMSExpiration: 0
    JMSPriority: 4
    JMSMessageID: ID:ad81e615a534a505ba645aaf110a134f0000000000000001
    JMSTimestamp: 1489506224433
    JMSCorrelationID: null
    JMSDestination: topic://myTopic?topicSpace=myBusTopic&busName=myBus
    JMSReplyTo: null
    JMSRedelivered: false
        JMSXDeliveryCount: 1
```

```
        JMS_IBM_System_MessageID: 0AC32A3495A1AF48_3500002
        JMSXUserID: uid=wasadmin,o=defaultWIMFileBasedRealm
        JMSXAppID: Service Integration Bus
publisher Topic: test message 0

Read 1 messages

Finished.


#--- JMS Connections to SIBus -----------------------------

        1 to myBus

            myConnectionFactory (jms/myConnectionFactory)

#--- SIBus Queue(s) Depth -- [SIBus:queue_name (JMS JNDI)] -----------------------------

        0 messages in myBus:_PSIMP.PROXY.QUEUE_0AC32A3495A1AF48
        0 messages in myBus:_PSIMP.TDRECEIVER_0AC32A3495A1AF48
        0 messages in myBus:_PTRM_0AC32A3495A1AF48
        0 messages in myBus:_SYSTEM.Exception.Destination.Node01.server1-myBus
        0 messages in myBus:myBusQueue (jms/myQueue)

#--- SIBus Topic(s) Depth -- [SIBus:topicspace_name (JMS JNDI)] -----------------------------

        0 messages in myBus:Default.Topic.Space
        0 messages in myBus:myBusTopic (jms/myTopic)
            0 messages for subscriber id.U9I881##sub.4X4QO8

#-----------------------------
INFO: Using WAS user 'wasadmin'and port '2811' defined in 'sibus_conf.py'...
#-----------------------------

#-----------------------------
delete subscriber [id.U9I881##sub.4X4QO8] for Topic [myTopic]

Finished.
```

Conclusion

The wsadmin interactive session that is powered with Jython can be really helpful when testing or creating tools. Because it's interactive when you create a script, you benefit from direct feedback of the commands that you run. On the contrary, this result is not possible with Java, for example, because you must compile it first. Now, you have the knowledge to send and receive SIBus queues and to publish, subscribe, and consume topic spaces by using Jython scripting language.

Acknowledgment

Thank you to Ty E. Shrake, Subject Matter Expert on SIBus, for his review.

Related topics

- Direct and indirect JNDI lookup methods for data sources
- Jython introspection and Websphere PMI
- WebSphere Application Server Administration Using Jython
- IBM Middleware User Community
- Python Tutorial
- Configuration IDs, containment paths and object names
- Getting Java objects methods
- Java class for introspection: java.lang.Object
- Java class for introspection: java.lang.Class