PEX: Developing Productions with Java

Version 2019.4
2020-01-28

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

PEX: Developing Productions with Java
InterSystems IRIS Data Platform Version 2019.4 2020-01-28

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

About This Book

This book describes how to use the PEX framework to write production business hosts and adapters in Java rather than ObjectScript. It contains the following chapters:

1

Introduction

The PEX framework allows you to write business hosts and adapters in Java and integrate them within a interoperability production. The PEX framework is powerful because you can integrate just one Java-based component into a production that also uses built-in components, or write the entire production in Java, depending on your use case. Once integrated, the production components written in Java are called at runtime and use the PEX framework to send messages to other components in the production. Some reasons to use the PEX framework include:

- Creating new adapters for protocols using available Java libraries.
- Using available Java libraries to perform complex analysis or calculations.
- Implementing persistent messaging and long-running business processes without using ObjectScript in your application's technology stack.

You can create the following production components using Java:

- Inbound Adapter
- Outbound Adapter
- Business Service
- Business Process
- Business Operation

Collectively, business services, business processes, and business operations are referred to as *business hosts*. For an introduction to adapters and business hosts and how they fit into an interoperability production, see Introducing Interoperability Productions.

Though you are writing code in Java to create your adapters and business hosts outside of InterSystems IRIS, the Management Portal is still the place to go to create and maintain the production. For example, you'll use the Management Portal to create the production, integrate the custom business hosts and adapters into the production, start and stop the production, and trace persistent messages running through the production.

1.1 Basic Workflow

Implementing Java components into an interoperability production consists of three basic steps:

1. *In you favorite IDE, write the business host or adapter in Java.* The PEX Java library, `com.intersystems.enslib.pex`, includes superclasses for each type of production component. For example, `com.intersystems.enslib.pex.BusinessOperation` is the superclass for business operations written in

Java. Implementing a business host or adapter in Java begins by creating the appropriate subclass and overriding its methods.

2. *In the Management Portal, add a built-in PEX component that points to the new Java class.* To integrate a custom Java business host into a production, use the Management Portal to open the production and add a built-in PEX business host to the production that acts as a proxy between the custom business host and the rest of the production. This built-in PEX business host has a setting that specifies the Java class of the custom business host. Integrating an adapter into the production also leverages a built-in PEX class as a proxy between the business host and the adapter written in Java.
3. *From the Management Portal or InterSystems Terminal, manually start the Java Gateway with the required Jar files in the classpath.* The PEX framework requires the Java Gateway be running on the port that is defined in the settings of the built-in PEX components. For more details, see Running the Java Gateway.

2

About Business Hosts and Adapters

This chapter discusses information that applies to all business hosts and adapters written in Java. For implementation details about a specific production component, refer to that component's chapter.

2.1 General Notes about Methods

As you override methods in Java to implement a production component, keep the following in mind:

- Each production component can override the `OnInit()` and `OnTearDown()` methods.
- While native interoperability methods use one input argument and one output argument and return a status, the corresponding PEX methods take one input argument and return the output argument as a return value.
- All error handling for PEX methods are done with exceptions.
- For native interoperability methods that don't require persistent objects as input and output arguments, the corresponding PEX methods can also use arbitrary objects as arguments and return values. PEX utilizes forward proxy and reverse proxy of the Gateway to map the arbitrary object appropriately. Examples of such methods are ProcessInput, OnProcessInput and action methods of OutboundAdapters.
- For native interoperability methods that require persistent objects as arguments, i.e. the methods that send messages to other processes, the corresponding PEX methods can use PEX Messages as arguments and return values. Examples of such methods are SendRequestSync, SendRequestAsync, OnRequest, OnResponse and OnMessage.
- When overriding callback methods, you should not change the formal spec of the methods even if you have customized the message class. The argument types should remain as objects.

2.2 Setting Runtime Variables

The Java code of a custom production component can contain properties that will be set at runtime based on values defined in the Management Portal. At runtime, the values specified in the **remoteSettings** field of the built-in PEX component are assigned to the corresponding Java component. Values are defined in the format: `property=value`. More than one property/value pair can be specified as long as each pair is on its own line. No spaces are allowed.

For example, suppose your inbound adapter needs a username and password at runtime. In your Java code, you create string variables called `username` and `pw`. Then, in the Management Portal, you can define the following in the **remoteSettings** field of the business service that uses the adapter:

```
username=Employee1
pw=MySecret
```

2.3 Messaging

Within the PEX framework, most messages sent *between business hosts* are objects instantiated from a subclasses of `com.intersystems.enslib.pex.Message`. You simply add properties to your subclass, and then pass instantiated objects of the subclass using methods like `SendRequestAsync()` and `SendRequestSync()`. Within InterSystems IRIS, the Java message object corresponds to an object of class `EnsLib.PEX.Message`, which makes the message persistent and dynamic. By manipulating the object of type `EnsLib.PEX.Message`, you can reference any property in the Java object. Internally, the Java object is represented as JSON as it passes between business hosts, so viewing the messages in the Management Portal are displayed in JSON format.

Though you can use other message objects, they must still be persistent if they are being passed between business hosts. For example, Java objects of type IRISObject could be used if they were mapped to persistent objects in InterSystems IRIS. Trying to pass a non-persistent object as a message between business hosts results in a runtime error.

Objects sent from an inbound adapter to a business service are arbitrary and do not need to be persistent.

3

Inbound Adapters

Business services use inbound adapters to receive specific types of input data. You can write a custom inbound adapter that is used by an ObjectScript business service, or it can be used by a business service that is also written in Java. For general information related to all production components written in Java, see About Business Hosts and Adapters.

3.1 Java Class

An inbound adapter written in Java extends `com.intersystems.enslib.pex.InboundAdapter` .

At a minimum, your Java inbound adapter should override the `OnTask()` method. At runtime, the `OnTask()` method is called at the interval defined in the settings of the business service that is using the adapter. From within `OnTask()`, call `BusinessHost.ProcessInput()` to dispatch to the associated business service. This `ProcessInput()` method takes an arbitrary object as input and receives an arbitrary object as a return value. The input object becomes the input argument of the business service's `OnProcessInput()` method, and the return object comes from the output of the business service's `OnProcessInput()` method. These arbitrary objects do not need to be persistent within InterSystems IRIS.

The methods that can be overridden include:

OnInit()
OnTearDown()
OnTask()

3.2 Using the Inbound Adapter

In the Management Portal, a business service has a setting, **AdapterClassname**, that specifies the class of the inbound adapter associated with the business service. When the inbound adapter is written in Java, the business service's **AdapterClassname** setting must be `EnsLib.PEX.InboundAdapter`. In addition, the **remoteClassname** setting must be set to the name of the Java class of the inbound adapter.

4

Outbound Adapters

Business operations use outbound adapters to send out specific types of data from the production. You can write a custom outbound adapter that is used by an ObjectScript business operation, or it can be used by a business operation that is also written in Java. For general information related to all production components written in Java, see About Business Hosts and Adapters.

4.1 Java Class

An outbound adapter written in Java extends `com.intersystems.enslib.pex.OutboundAdapter` .

Within the outbound adapter's Java class, you can create all the methods you need to successfully send out data from the production. Each of these methods can be called from the business operation associated with the adapter.

The methods that can be overridden include:

OnInit()
OnTearDown()

4.2 Using the Outbound Adapter

In the Management Portal, a business operation has a setting, **AdapterClassname**, that specifies the class of the outbound adapter being used. When the outbound adapter is written in Java, the business service's **AdapterClassname** setting must be `EnsLib.PEX.OutboundAdapter`. In addition, the **remoteClassname** setting must be set to the name of the Java class of the outbound adapter.

5

Business Services

Business services connect with external systems and receive messages from them through an inbound adapter. For general information related to all production components written in Java, see About Business Hosts and Adapters.

5.1 Java Class

A business service written in Java extends `com.intersystems.enslib.BusinessService`.

The `OnProcessInput()` method takes an arbitrary object from the `ProcessInput()` method of the business service's adapter, and returns an arbitrary object. These arbitrary objects do not need to be persistent.

For information about sending messages between the business service and another business host, see Messaging.

The methods that can be overridden include:

OnInit()
OnTearDown()
OnProcessInput()
SendRequestAsync()
SendRequestSync()

5.2 Using the Business Service

When you want to integrate the Java business service into the production:

1. Open the production in the Management Portal and add a business service based on the PEX class `EnsLib.PEX.BusinessService`.
2. Define the **remoteClassname** setting of the new PEX business service as the Java class you created for the custom business service.
3. Use the **gatewayHost** and **gatewayPort** settings of the PEX business service to define the values of the Java Gateway.

For example, suppose you have a business service written in a Java class called `MyJavaBusinessService`. To incorporate this custom business service into the production, you go to the Management Portal and add a business service based on the

PEX class `EnsLib.PEX.BusinessService`. Once added, edit the **RemoteClassname** setting of the PEX business service to point to the Java classname `MyJavaBusinessService`.

6

Business Processes

Business processes allow you to define business logic, including routing and message transformation. Business processes receive messages from other business hosts in the production for processing.

For general information related to all production components written in Java, see About Business Hosts and Adapters.

6.1 Java Class

A business service written in Java extends `com.intersystems.enslib.BusinessProcess`.

For information about sending messages between the business process and another business host, see Messaging.

The methods that can be overridden include:

OnInit()
OnTearDown()
SendRequestAsync()
SendRequestSync()
OnRequest()
OnResponse()
OnComplete()

6.1.1 Persistent Properties

Within InterSystems IRIS, business processes are persistent objects. For the lifespan of a Business Process, properties of the Business Process are stored and accessible during each callbacks. Within the PEX framework, you can save properties of the business process class saved by using `@Persistent` annotation. Within InterSystems IRIS, persistent properties are saved in the corresponding instance of the business process. Persistent properties are restored before each callback and are saved after each callback. The @Persistent annotation only works for String, primitive types and their boxed types. The following is an example of using the @Persistent annotation:

```
@Persistent
public integer runningTotal = 0;

@Persistent
public string CurrentName = "temp";
```

6.2 Using the Business Process

When you want to integrate the Java business process into the production:

1. Open the production in the Management Portal and add a business service based on the PEX class `EnsLib.PEX.BusinessProcess`.
2. Define the **remoteClassname** setting of the new PEX business process as the Java class you created for the custom business process.
3. Use the **gatewayHost** and **gatewayPort** settings of the PEX business process to define the values of the Java Gateway.

For example, suppose you have a business process written in a Java class called `MyJavaBusinessProcess`. To incorporate this custom business process into the production, you go to the Management Portal and add a business process based on the PEX class `EnsLib.PEX.BusinessProcess`. Once added, edit the **RemoteClassname** setting of the PEX business process to point to the Java classname `MyJavaBusinessProcess`.

7

Business Operations

Business operations connect with external systems and send messages to them via an outbound adapter.

For general information related to all production components written in Java, see About Business Hosts and Adapters.

7.1 Java Class

A business operation written in Java extends `com.intersystems.enslib.BusinessOperation`.

For information about sending messages between the business operation and another business host, see Messaging.

At runtime, the `OnMessage()` method is called when the business operation receives a message from another business host. From within this method, the business operation can call any of the methods defined in the outbound adapter associated with the business operation.

Parameters for calls from a business operation to an outbound adapter are primitive types only, and do not need to be persistent.

The methods that can be overridden include:

OnInit()
OnTearDown()
OnMessage()
SendRequestAsync()
SendRequestSync()

7.2 Using the Business Operation

When you want to integrate the Java business operation into the production:

1. Open the production in the Management Portal and add a business operation based on the PEX class `EnsLib.PEX.BusinessOperation`.
2. Define the **remoteClassname** setting of the new PEX business operation as the Java class you created for the custom business operation.

3. Use the **gatewayHost** and **gatewayPort** settings of the PEX business operation to define the values of the Java Gateway.

For example, suppose you have a business operation written in a Java class called `MyJavaBusinessOperation`. To incorporate this custom business operation into the production, you go to the Management Portal and add a business operation based on the PEX class `EnsLib.PEX.BusinessOperation`. Once added, edit the **RemoteClassname** setting of the PEX business operation to point to the Java classname `MyJavaBusinessOperation`.

8

Running the Java Gateway

The PEX framework requires the Java Gateway to be running when the production is running. It is recommended that you manually start the Java Gateway from the Management Portal or InterSystems Terminal rather than adding EnsLib.JavaGateway.Service to the production. Complete the following tasks to ensure that the Gateway is properly configured to support the PEX framework:

- In the Management Portal, use the **gatewayHost** and **gatewayPort** settings of each PEX component to specify the host and port of the Gateway that will be running.
- Put the required jar files in the classpath of the Gateway.

8.1 Jar Files Required by the Gateway

The following Jar files must be on the classpath of the Java Gateway used for the PEX production:

- gson-2.8.5.jar, which is a serialization/deserialization library that converts Java objects into JSON and back. This jar file is located at `<install-dir>\dev\java\lib\gson\gson-2.8.5.jar`.
- Jar files that contain the subclasses you created for each production component. For example, if you wrote an inbound adapter in Java, you might have a jar file `MyJavaAdapter.jar`.

If you are using a dedicated Gateway for your PEX production, it makes sense to add these jar files to the classpath when starting the Gateway. Alternatively, you can specify these jar files for individual PEX components in the Management Portal. Every PEX component in a production has a setting **Gateway Extra CLASSPATH**. You can add the jar file that corresponds to the PEX component in this setting rather than specifying it when you start the Gateway, but don't forget to also add **gson-2.8.5.jar** to the classpath.

A

Java Business Hosts

A.1 Introduction to Java Business Hosts

Important: In previous releases of InterSystems IRIS before the PEX framework was available, Java Business Hosts provided a way to create new business services and business operations using Java without need for any ObjectScript coding. This feature is now deprecated; use the PEX framework instead.

You can use Java Business Hosts with the following kinds of messages:

- Plain text
- XML
- X12
- EDIFACT
- HL7(InterSystems IRIS for Health and HealthShare Health Connect only)
- ASTM (InterSystems IRIS for Health and HealthShare Health Connect only)

To develop a production using Java Business Services and Operations, you do the following:

- Write your Java code implementing the `com.intersystems.gateway.bh.BusinessService` and the `com.intersystems.gateway.bh.BusinessOperation` classes. These classes enable your Java code to send messages to a production and receive messages from a production.
- Compile your Java classes into jar files.
- Use the Java Business Host page in the Management Portal to add a EnsLib.JavaGateway.Initiator component to a production and then generate Java Business Services and Operations from your jar files.
- Add the Java Business Services and Operations to the production using the Production Configuration page in the Management Portal.
- Adjust any settings on your Java Business Services and Operations using the Product Configuration page.
- Add any other Business Services, Business Processes, and Business Operations to your production. For example, you can add a router Business Process or a Data Transformation to your production.
- Use the production infrastructure to manage and monitor your production. For example, you can examine messages and trace their path through the production.

Java Business Hosts provides an easy way to create business services and operations in Java. It uses the InterSystems IRIS™ Java Gateway to do this. Although it is more work to use the Java Gateway directly, it provides more options and capabilities than Java Business Hosts. For more information, see Using the Java Gateway.

A.2 Developing Java Business Services and Operations

This section describes how to implement the Java code for business services and business operations. To create a business service or business operation, you implement the following classes:

- `com.intersystems.gateway.bh.BusinessService`
- `com.intersystems.gateway.bh.BusinessOperation`

These classes are defined in the intersystems-gateway-3.0.0.jar file provided in the *install-dir*\dev\java\lib\JDK18 directory. In addition to the BusinessService and BusinessOperation classes, this jar file defines the `com.intersystems.gateway.bh.Production`, which provides access to the production and the Business Service.

For receiving messages from an external service, you implement a Java application that listens to messages and includes the Java class:

`com.intersystems.gateway.bh.BusinessService`

with the following methods:

- OnInit: — this method is called when the production starts or the business service is enabled. It typically starts a listener that will receive messages. The listener receives the messages from the external service and then sends them to the business service in the production by calling the method Production.SendRequest(). The production is passed in as an argument to OnInit. Your code should save it so that it can call SendRequest in the listener.
- OnTearDown: — this method is called when the production is stopped or the business service is disabled. It typically stops the listener.

To define settings in the Business Service, define a static string named SETTINGS, that has a string value of a comma separated list of settings name. For example, the following defines settings named MIN and MAX:

```
public static final String SETTINGS = "Min,Max";
```

For sending messages from the production to an external service, you implement a Java application, which includes the Java class:

`com.intersystems.gateway.bh.BusinessOperation`

with the following methods:

- OnInit — this method is called when the business operation starts. It typically initializes any structures needed by the OnMessage method. The production is passed in as an argument to OnInit.
- OnMessage — this method is called when the business operation receives a message. It is responsible for sending the message to the external service.
- OnTearDown — this method is called when the business operation ends. It typically releases any structures created by the OnInit method.

The production is provided as an argument to the BusinessService.and BusinessOperation OnInit method. It allows you to access settings on the Business Service and to set its status. The class `com.intersystems.gateway.bh.Production` has the following methods:

- SendRequest — Sends a request message to the target configuration item of the Business Service. This method is only available to the BusinessService. It is not available in the BusinessOperation.
- GetSetting — Gets the value for the specified Business Service or Business Operation setting.
- SetStatus — Sets the status of the Business Service or Business Operation configuration item and changes the color of the item on the Production Configuration page.
- LogMessage — Writes a message to the production log. You can use this to report errors or to help debug code.

For the reference documentation for these classes, see Javadocs Reference for Java Business Hosts Classes.

A.3 Generating and Configuring the Java Business Hosts

This section describes how to generate Java Business Hosts from jar files that contain one or more classes that implement `com.intersystems.gateway.bh.BusinessService` or `com.intersystems.gateway.bh.BusinessOperation`. To generate these hosts:

1. Add the Java Business Hosts initiator to a new or existing production.
2. If necessary, configure the settings on the initiator.
3. Use the Java Business Hosts Management Portal page to select a jar file.
4. Fill in the fields on the page.
5. Generate the host.

Once you have generated the hosts, you can add them to the production containing the initiator using the Production Configuration page, specify the settings for each host, and enable the hosts to run your production.

If when you select the jar file on the Java Business Hosts page you get the `The selected file does not contain any classes which can be imported as a java business host. Try selecting a different jar file.` error, the problem may be one of the following:

- Mismatch between the class name and the structure of the files in the jar archive. For example, if the class name is JavaHosts.JHBusinessService, but the jar file archive is organized so that the .class files are in the directory Output\JavaHosts\JHBusinessService, the extra level causes the initiator to not recognize the Java Business Hosts file.
- Missing jar files that are referenced by the Business Host jar file. Any jar file referenced in the Business Host's jar file must be specified in the initiator's **Class Path** setting.