Article
Heloisa Paiva · Sep 2, 2022 4m read

# Interoperability - Step to step on creating and connecting business hosts

I recently started to study interoperability and I found the official documentation very helpful in understanding how it works, though I still had some trouble implementing it myself. With the help I got from my coworkers, I managed to create a Demo of a system and learn through practice. Because of that, I decided to write this to help others with "getting their hands dirty" and share the help I got.

## Introduction

Firstly, let's briefly remember some concepts:

- Interoperability - this word's meaning is not as complex as its pronunciation - it is basically the name for the "magic" that brings and delivers all kinds of information from one system to another.
- Business hosts - if interoperability is the magic, the business hosts are the magician's top hat - there are the business services which recognize and receive information and send it as a message to business processes or business operations. The business operations perform the desired operations (as the name suggests) and deliver the message. The business processes control the flow of the messages: they define where the message goes (based on whatever you chose) and how it is passed on.
- Adapters - the adapters are some classes we can use to recognize and manipulate all kinds of information we may have to deal with. In practice, we put them as parameters and (optionally) properties to access its methods and properties.

# Preparing to build the production

It is easier to start simple - let's think about services and operations first - say you have a service that receives one kind of message that is easily recognized by the only operation we're working with.

Development is easier when the purpose of the production and its parts are very clear. If you'd like, it might be helpful to draw a diagram or write down the steps you want it to complete.

For example:

Start by asking "what do I need to do?" - in my Demo, I need to manipulate an SQL table - I give information such as a Title and an Author of a book and insert it on a table.

"So, what do I need the production to do?" - it has to receive a message containing the Title and Author and perform an SQL INSERT.

"Ok, how does it happen?" - the Business Service (BS) is going to receive the Title and Author, pass it on to the Business Operation (BO). The BO performs the SQL code.

"Now that I have the paths the information will follow, I need to understand the message. What is it?" - There are many ways I can send the data. I could send it in a File, or a REST application, even an email.

Let's pick the file to start simple. My BS will receive the file, read it and send its info to the BO. The BO performs the query.

# Getting to work

You can start by the part you're most confident in coding.

- The Business Service

```
Class Demo.Books.BS.FileService Extends Ens.BusinessService
{

Parameter ADAPTER = "EnsLib.File.InboundAdapter";

Method OnProcessInput(pInput As %Stream.Object, Output pOutput As %RegisteredObject) As %Status
  {
      Set tSC = $$$OK

      Try
      {
          // Reads the first line of the recieved File
          Set tLine = pInput.ReadLine()

          // Sets the properties of the request based on the data recieved
          Set tRequest = ##class(Demo.Books.BO.Register.Request).%New()
          Set tRequest.Title = $PIECE(tLine, ",", 1)
          Set tRequest.Author = $PIECE(tLine, ",", 2)

          // Sends to operation
          Set tSC = ..SendRequestSync("Demo.Books.BO.Operation", tRequest, .tResponse)

          Throw:$$$ISERR(tSC)
      }
      Catch (tEx)
      {
          Set:'$$$ISERR(tSC) tSC = tEx.AsStatus()
      }

      Quit tSC
  }

}
```

I started with the BS. Now it is clear to me that I need to implement the Request class so that it stores data and the Operation that it will be sent to.

- The Business Operation

Request

```
Class Demo.Books.BO.Register.Request Extends (Ens.Request, %XML.Adaptor)
{

Parameter RESPONSECLASSNAME = "Ens.Response";

Property Title As %String;

Property Author As %String;

}
```

The Request class will be as simple as this: I ' ll only use it to store some data. The %XML.Adaptor is meant to show the Request on the Management Portal for error management.

Operation

```
Class Demo.Books.BO.Operation Extends Ens.BusinessOperation
{

Property Adapter As EnsLib.SQL.OutboundAdapter;

Parameter ADAPTER = "EnsLib.SQL.OutboundAdapter";

Parameter INVOCATION = "Queue";

Method Register(pRequest As Demo.Books.BO.Register.Request, Output pResponse As Ens.Response) As %Status
 {
     // I could implement the method here, but to be more organized I created an Abstract Class for it.
     Quit ##class(Demo.Books.BO.Register.Method).Execute(##this, pRequest, .pResponse)
 }

// This map recognizes the type of message was sent in the request and executes the method specified.

XData MessageMap
 {
<MapItems>
 <MapItem MessageType = "Demo.Books.BO.Register.Request">
   <Method>Register</Method>
  </MapItem>
 </MapItems>
 }

}
```

The BO has a message map to give the adequate directions to each kind of message that arrives. For example, if in the BS, in the SendRequestSync method, in the Request argument, I had used a different type, such as “ Demo.Books.BO.SearchTable.Request, ” I could create another MapItem with this message type, referencing a Search method.

Method

```
Class Demo.Books.BO.Register.Method [ Abstract ]
{

ClassMethod Execute(pHost As Demo.Books.BO.Operation, pRequest As Demo.Books.BO.Register.Request, Output pResponse As Ens.Response) As %Status
 {
     Set tSC = $$$OK

     Try
     {
         Set tSC = pRequest.NewResponse(.pResponse)

         Throw:$$$ISERR(tSC)

         Set tSC = pHost.Adapter.ExecuteUpdate(.pRow, "INSERT books (title, author) VALUES (?,?)", pRequest.Title, pRequest.Author)

         Throw:$$$ISERR(tSC)
     }
     Catch (tEx)
     {
         Set:'$$$ISERR(tSC) tSC = tEx.AsStatus()
     }

     Quit tSC
 }

}
```

Here, you implement whatever the operation is supposed to do. It is better to implement the method in another class because after recompiling the BO you might need to restart the production to work with new changes.

# Settings on Management Portal

Finally, to get things working, follow:

Management Portal > Interoperability > List > Productions > New

The portal will create a class with the Production information.

Then, you add a Service with the class you created and set the File path (where you will put the input files) and a work path.

Also, add an operation with the class you created and specify its settings if necessary.

# Observations

- The Business Operation could receive a Synchronous Request, which means that the Service would only respond once the BO had retrieved its message, because the Service's response depends on the BO's response, for example if it had to perform a search in the table operation. Because the production only performs an INSERT operation, the BS only needs to send information and the BO INSERTs it; no response is needed, so we could have an Asynchronous request.
- It is outside the scope of this post to discuss specifications of adapters such as File and SQL adapters, this article is meant to give an overview on the coding and steps to better understand the practice.
- Feel free to contact me - I'd love to help in anyway I can!

#Innovatium #Interoperability #InterSystems IRIS