Article
Mihoko Iijima · Mar 5, 2021 7m read

[InterSystems IRIS for the First Time] Interoperability: Creating Components (Business Process)

This article is a continuation of this post**.**

In the previous article, we discussed business operations' creation from the components required for system integration.

In this article, you will learn how to create a business process that calls the two business operations you have defined in the sequence order.

- Production
- Message
- **Components**
  - Business Services
  - **Business Processes**
  - Business Operations (previous post)

The business process acts as the coordinator (command center) of the process.

The processing adjustments you may want to implement in the sample include the following:

Step 1: Provide the **city name** to an external Web API and request **weather information**.
Step 2: Register the result of the query (**weather information**) from Step 1 and the name of the purchased product received at the start of the production.

In the sample business process, we will wait for the answer to step 1 and adjust step 2 to operate.

In the process of waiting for a response (i.e., synchronization), for instance, what happens if step 1) doesn't respond for a few days?

If new messages are delivered to the business process while waiting for a response for a few days, the messages will not be dismissed since they are stored in a **queue**. However, the business process will not process new messages, and there will be a delay in the operation.

Note: Business processes and business operations have cues.

Therefore, in production, when there is a synchronous call, there are two ways for the business process to move: **A) to synchronize perfectly**, and **B) to save the state of the business process itself in the database and hand over the execution environment so that other processes can run while waiting for a response.**

A) How to synchronize perfectly:

While a synchronous call is being made, the business process's processing is ongoing, and waiting for the next message to be processed until all processing is completed.
➡This function is used when the order of processing needs to be guaranteed in the first-in-first-out method.

B) The method of saving the state of the business process itself in the database and hand over the execution environment so that other processes can run while waiting for a response is

When a synchronous call is made, the process saves its state in the database. When a response message is received, and it is time to process the message, it opens the database and executes the next process.
(IRIS will manage the storage and re-opening of business processes in the database).
➡ Used when it is acceptable to switch the processing order of messages (i.e., when it is allowed to process more and more different messages received while waiting for a response).

In the sample, B) is used.

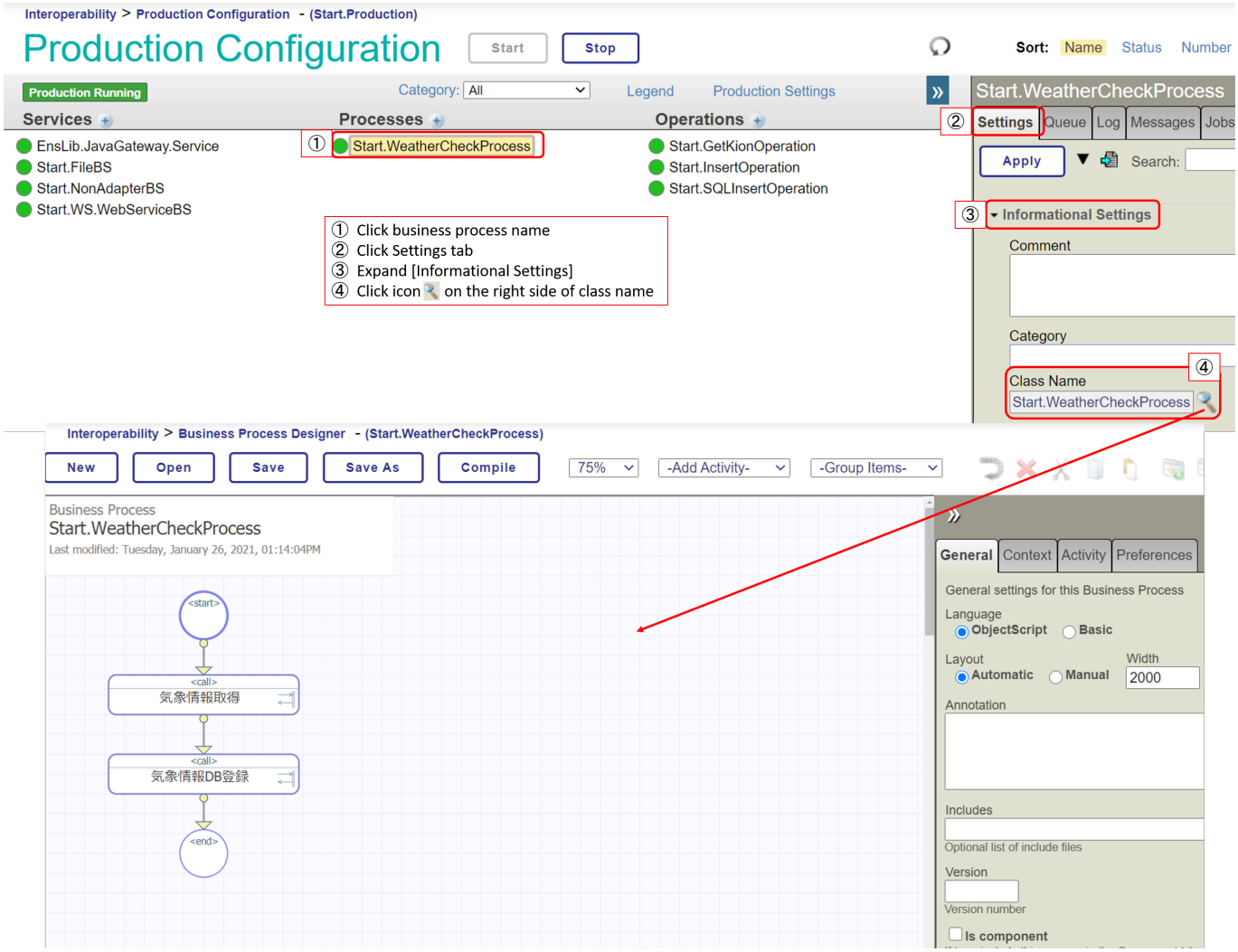
There are two types of editors for creating business processes: a Business Process Editor that allows you to place processing boxes (activities) and implement them while defining their execution, and a method for creating them using ObjectScript in Studio or VSCode.

If you use the Business Process Editor, you will use the call activity to invoke the component, but this activity is **implemented** in the B) way. **Of course, you can also implement the A)** method in the Business Process Editor, except that you will not use the call activity in that case (you will use the code activity).

In this section, I will explain how to create it.

If you use the Business Process Editor, you write them in the Management Portal.

You can also open the business process from the production configuration page. The figure below shows the procedure.



Icons like in this editor are called activities, and those marked with are activities that can invoke other components.

This symbol indicates that a response message will be returned (i.e., a synchronous call will be made). The activity defaults to the asynchronous call setting, which can be changed as needed.

Now let's look at business processes, which are components that are invoked upon receiving a request message, as well as business operations.

In the sample, the request message: It is set to start when it receives a Start.Request and does not return a response message.

In the business process, messages appear in various situations.

Request messages that are sent to business processes.

Request message (+ response message) to be sent when calling another component using the activity.

In the Business Process Editor, the names of the objects that store messages are clearly separated to be able to see which message was sent from which destination.

- request（basic requirements）

The message that triggered the start of the business process, in our example, is Start.Request (the message to be specified in the Request settings on the Context tab in the Business Process Editor)

- response（basic response）

Response message to return to the caller of the business process (not used in the sample) (message to be specified in the settings of the response in the context tab in the Business Process Editor)

- callrequest（request message）

Request message to be sent when calling the component determined by the activity.

- callresponse（response message）

Response message returned from the component specified by the activity.

callrequest and callresponse are objects that will be deleted when the call processing of the activity is completed.
All other objects will not disappear until the business process is finished.

Now comes the problem when callresponse disappears.

That's because, as you can see in this sample,
When calling a component, if you want to use the response result of a previously called component, the response message will be lost, and the information that was to be used in the next component will be erased.

It is a problem

What should we do?・・・・・

In such a case, you can use the context object.

The context object, like request/response, is an object that survives until the end of the business process.

Moreover, since context is a generic object, it can be defined in the process editor.

In addition to the context, the response object can also be used if it has a property that matches the inherited data's saving.

Now, let's go over the steps again.

Response message in the **light blue balloon**: Start.Response is an object that will be deleted when the process is finished.

Since we want to use the response message (Start.Response) that contains the weather information as the message to be sent to the next [Business Operation for DB Update], we have to implement the context object in a way that all the property values of the response message (Start.Response) can be assigned to it.

Then what is the setting for the context property?

The properties are defined in "Context Properties" in the Context tab of the Business Process Editor.

In this case, we would like to save all the properties of the response message (Start.Response) to the context object. Therefore, the property type specification is set to Start.Response.

Following that, check the settings in the activity.

The request and response messages have a button called ○○ Builder.

Clicking on this button will launch a line-drawing editor that allows you to specify what you want to register in the properties of each message.

After this, the business operation for requesting a database update (Start.SQLInsertOperation or Start.InsertOperation) is called in the same way with the activity, and you are all set.

(For more information, see Configuring for Business Processes).

Once you have completed the verification, you can test it. The testing method is the same as the one used for testing business operations (see this article).

The trace after the test is as follows:

Since the business process is the coordinator, we could see that it invoked the defined components sequentially, keeping the synchronous execution.

Note 1: The sample only deals with the call activity, but various other activities such as data transformation.

Note 2: Business processes created by ObjectScript alone, other than the Business Process Editor, inherit from the Ens.BusinessProcess class. If it is created in the Business Process Editor, it inherits from the Ens.BusinessProcessBPL class.

The business process is the coordinator of the system integration process.
The Business Process Editor provides the following types of variables for messages (request/response/callrequest/callreponse/context).
A business process created with the Business Process Editor can work in a way that does not delay other messages, even if there is synchronization in the component's calling.

In the next section, we will finally show you how to develop the last component: business services.

#Beginner #Business Process (BPL) #Interoperability #InterSystems IRIS #InterSystems IRIS for Health

Source URL:https://community.intersystems.com/post/intersystems-iris-first-time-interoperability-creating-components-business-process