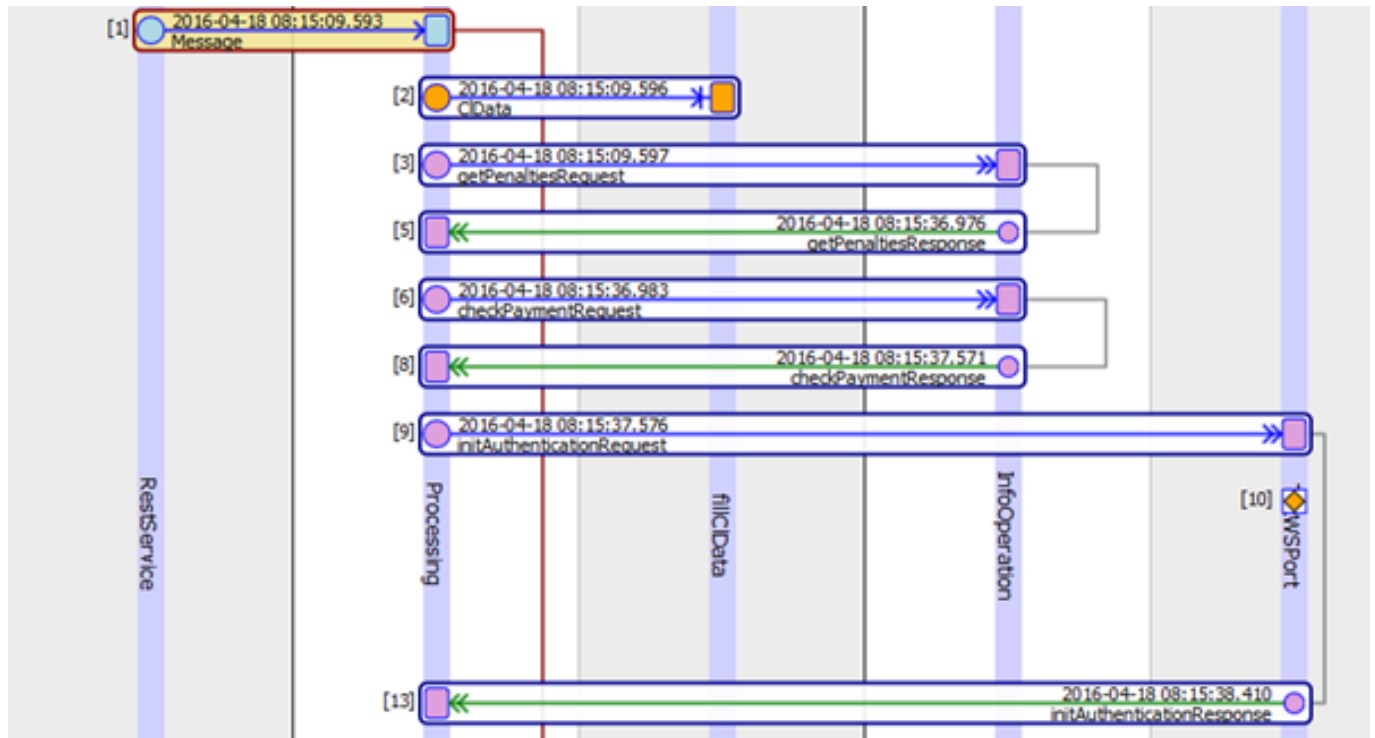


Article

[Tamara Lebedeva](#) · Jul 21, 2016 6m read

## How we learned to stop worrying and love InterSystems Ensemble



Preface: our small, but very ambitious company called “ Black Mushroom Studio ” came up with an idea for an e-commerce project and a mobile app that would let users pay for certain goods/services via a payment aggregator.

What we had initially: an Android app skeleton that, of course, liked communicating via HTTP and JSON, and a payment system with an API – web services with SOAP content.

Goal: make it all work together.

The following factors influenced the choice of the technology stack: speed of development and ability to quickly react to changes. The product was supposed to be an instant success. While competitors were still producing estimates, we wanted to launch the product. While our competitors were looking for the right developers, we were supposed to be counting our first

moneys. With this restricting factor in place, we still needed a serious approach to work, since it was all about investors ' money, and that is something that requires extra attention.

We could spend a lot of time talking about the advantages and disadvantages of specific technologies from specific vendors and the benefits of open source products. Having analyzed several products (which alone is worth of a separate article), we concluded that InterSystems Ensemble was the best choice for our needs.

Only one of our developers had practical experience of developing with Caché ObjectScript, but nobody knew Ensemble. However, we managed to implement the rather complex business logic of our product with Ensemble in just a couple of weeks.

What helped us:

1. Ensemble is a comprehensive product combining a DBMS, an application server, an enterprise service bus, a BMP system and a technology for developing analytical BI applications. There is no need to learn several solutions and integrate them.
2. Object-based storage model. If we want save an object to a database, we just save it to a database.
3. A very simple method of integration with external/internal systems based on various protocols thanks to an extendable library of adaptors.

## Top-level solution

A client sends a request with JSON content over the HTTP protocol to a server port. This port is listened to by Ensemble ' s “ black box ” . The client gets a synchronous response after the end of processing.

## What ' s inside

Using Ensemble, we implemented a production (an integration solution in Ensemble) consisting of three parts: business services, business processes, business operations (from now on, I will be using these terms without the “ business ” prefix for ease of reading).

A service in Ensemble is a component that allows you to receive requests over various protocols; a process is the applications ' s logic; an operation is a component that lets you send outgoing requests to external systems.

All interaction inside Ensemble is based on message queues. A message is an object of a class inherited from `Ens.Message` that makes it possible to use and transmit data from one part of a production to another.

The service in our case uses an HTTP adaptor inherited from `EnsLib.HTTP.InboundAdapter`, receives a request sent by a client, converts it into a “ message ” and submits it to the process. The business process responds with a “ message ” containing processing results and converts it into a response for the client.

## Here is how it looks in our solution:

```
Method OnProcessInput(pInput As %Library.AbstractStream, Output pOutput As %Stream.Object) As %Status  
  
{  
  
    Set jsonstr = pInput.Read()  
  
    // Let's convert it into a message  
  
    Set st = ##class(%ZEN.Auxiliary.jsonProvider).%ConvertJSONToObject(jsonstr,"invoices.Msg.Message",.tApplication)  
  
    Throw:$$$ISERR(st) ##class(%Exception.StatusException).CreateFromStatus(st)
```

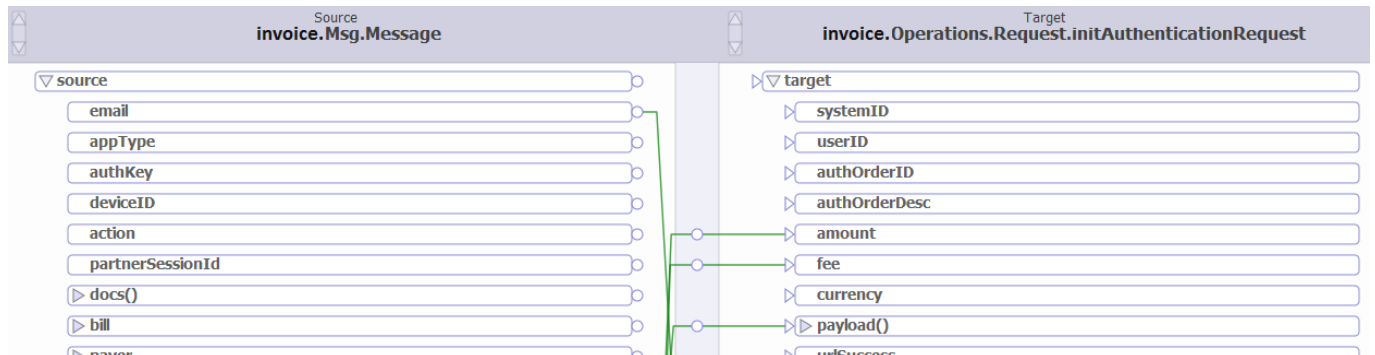
```
// Some logic for filling messages with data,  
  
// characteristic of our queries  
  
// Let's start a business process  
  
Set outApp=##class(invoices.Msg.Resp).%New()  
  
Set st =..SendRequestSync("Processing",tApplication,.outApp)  
  
Quit:$$$ISERR(st) st  
  
// Let's convert the response to json  
  
Set json=""  
  
Do ##class(invoices.Utills).ObjectToJSON(outApp,,,"aeloqu",.json)  
  
// Let's put json to the response  
  
Set pOutput=##class(%GlobalBinaryStream).%New()  
  
Do pOutput.SetAttribute("Content-Type","application/json")  
  
Do pOutput.Write(json)  
  
Quit st  
  
}
```

A business process is an implementation of the business logic of our application. It contains a sequence of actions that need to be performed to send a response to the client. For example: save/update the client's data, add a new traceable service, request a service status, pay for a service. Without an integrated platform, we'd have to write quite a lot of code.

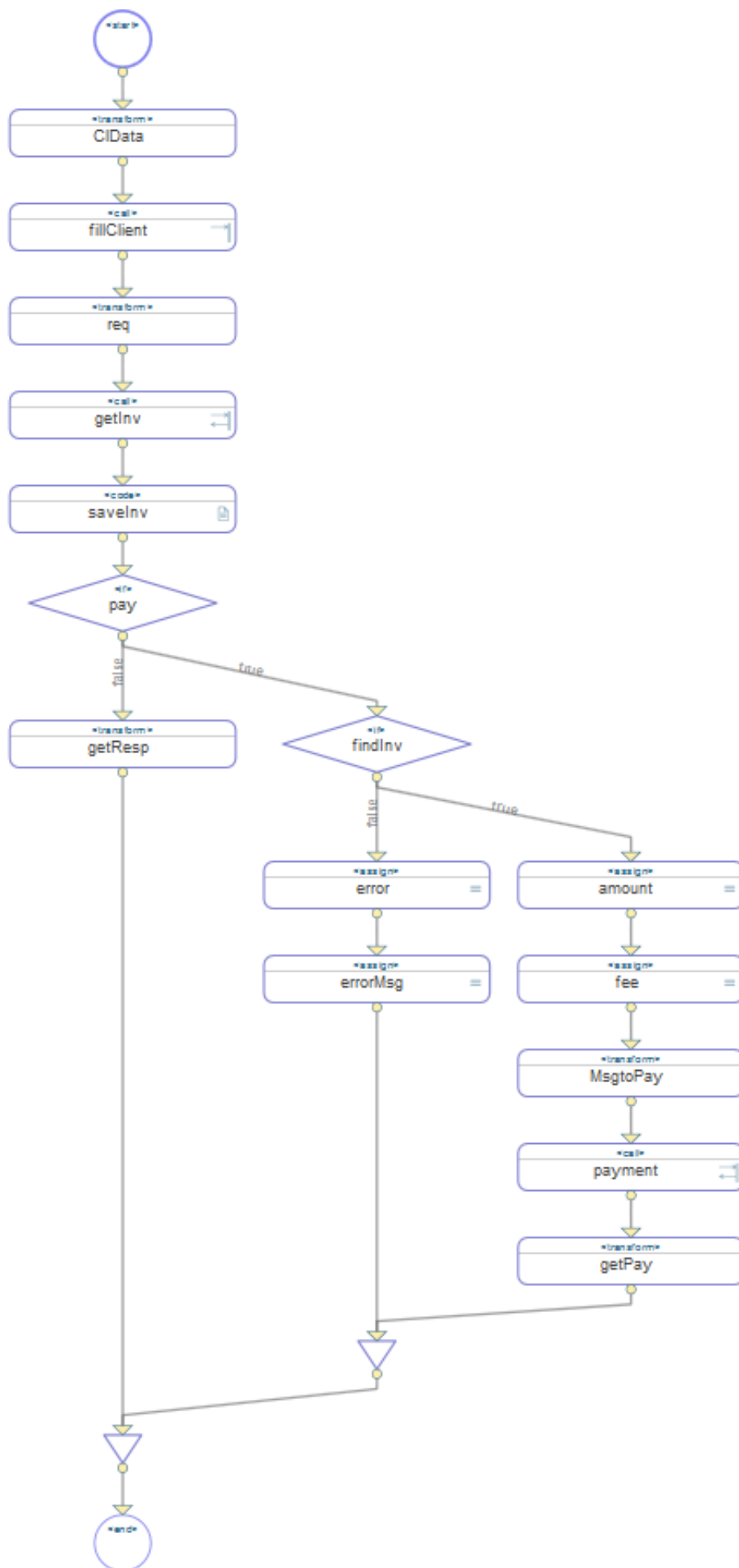
What we need to do in Ensemble: build a business process in a visual editor from different elements. Business processes are described in the Business

Process Language (BPL) language. Elements include simple (and not very) allocations, data conversion, code calls, synchronous and asynchronous process and operation calls (more on this below).

Data conversion is also very convenient and supports data mapping without having to write a line of code:

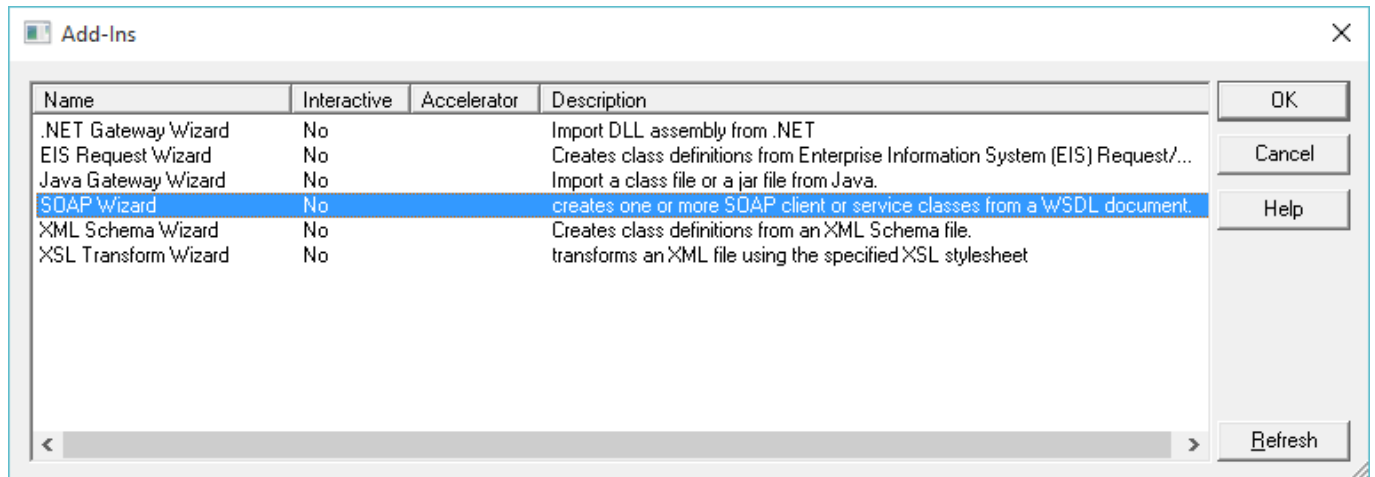


As the result, we got the following instead of a pile of code at a certain stage:

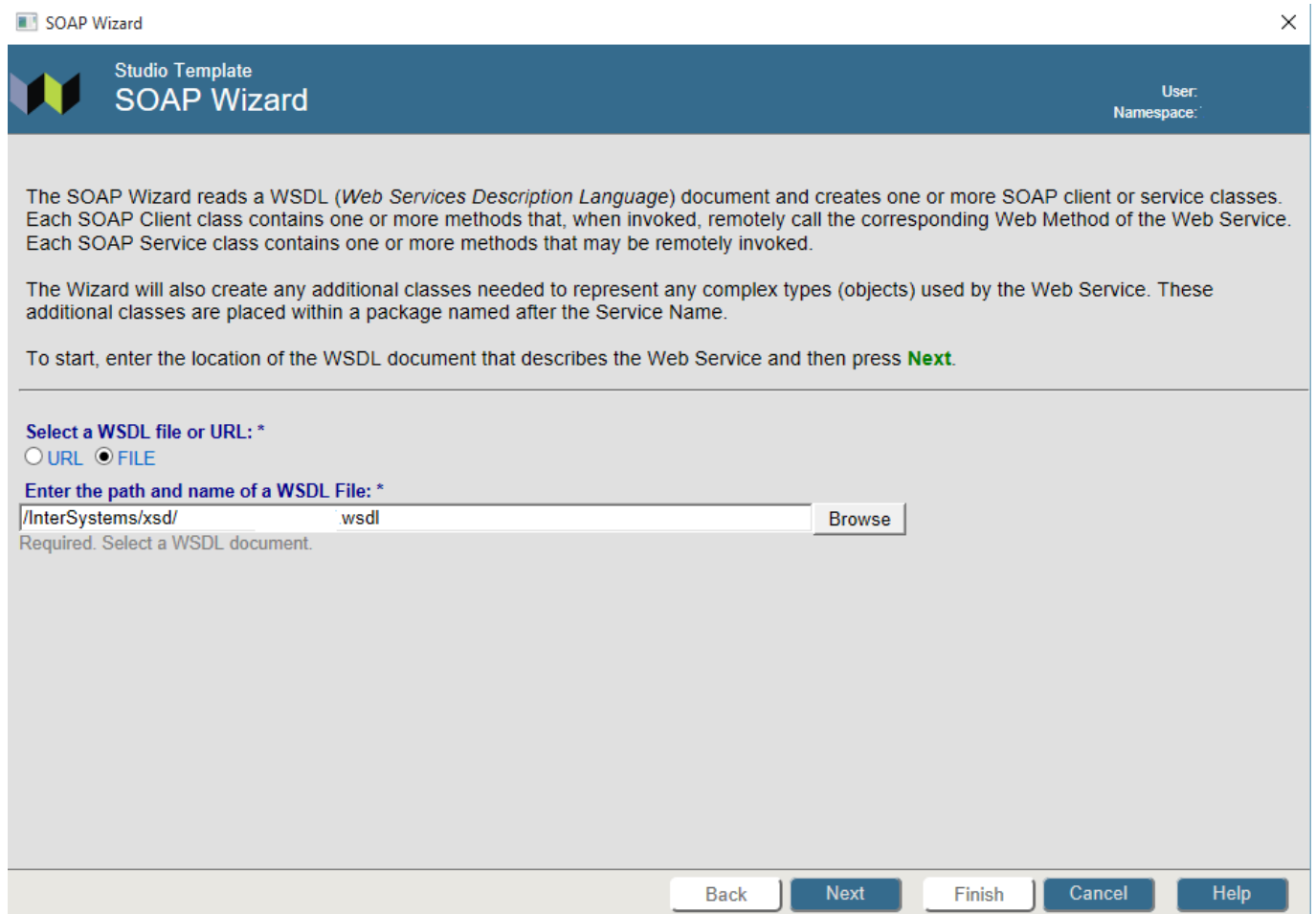


A few words about operations now. This entity allows us to send a request via some protocol to an external system.

How we did it: used a built-in studio extension to import the WSDL provided by the payment system from Caché Studio WSDL:



Let ' s specify the location of the WSDL:



SOAP Wizard

Studio Template  
SOAP Wizard

User:  
Namespace:

The SOAP Wizard reads a WSDL (*Web Services Description Language*) document and creates one or more SOAP client or service classes. Each SOAP Client class contains one or more methods that, when invoked, remotely call the corresponding Web Method of the Web Service. Each SOAP Service class contains one or more methods that may be remotely invoked.

The Wizard will also create any additional classes needed to represent any complex types (objects) used by the Web Service. These additional classes are placed within a package named after the Service Name.

To start, enter the location of the WSDL document that describes the Web Service and then press **Next**.

Select a WSDL file or URL: \*

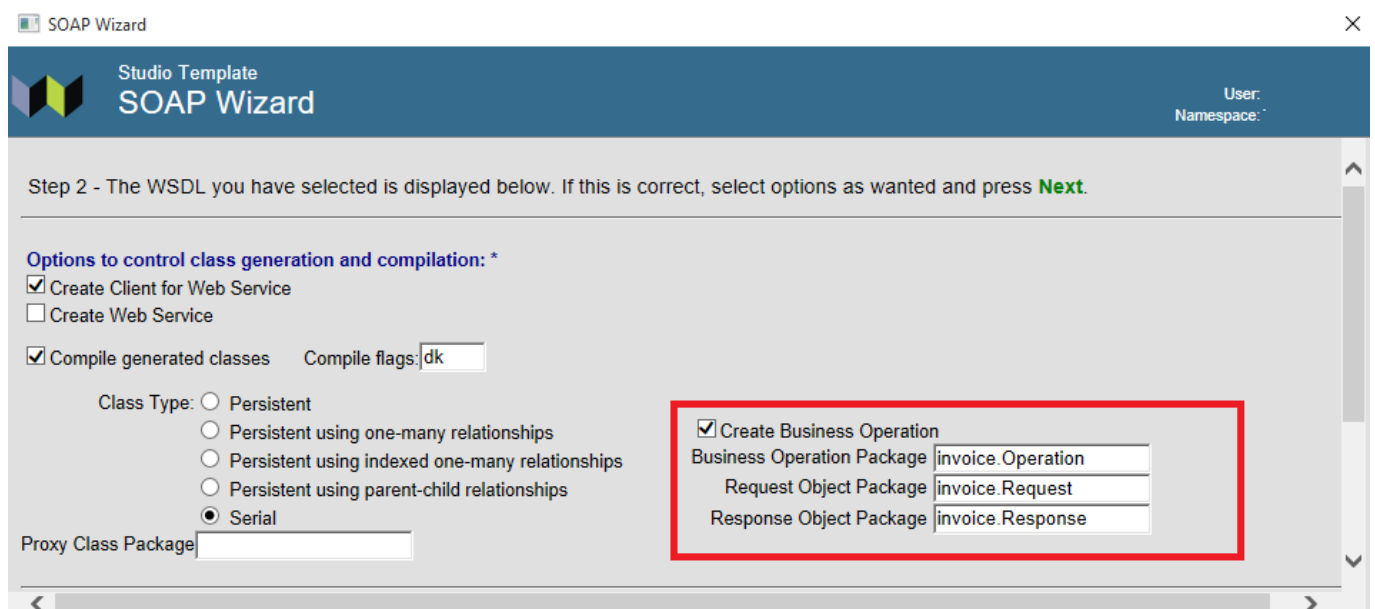
☐ URL ☒ FILE

Enter the path and name of a WSDL File: \*

/InterSystems/xsd/ .wsdl

Required. Select a WSDL document.

Let ' s tick the “ Create an operation ” box during import:



SOAP Wizard

Studio Template  
SOAP Wizard

User:  
Namespace:

Step 2 - The WSDL you have selected is displayed below. If this is correct, select options as wanted and press **Next**.

Options to control class generation and compilation: \*

☒ Create Client for Web Service  
☐ Create Web Service

☒ Compile generated classes Compile flags: dk

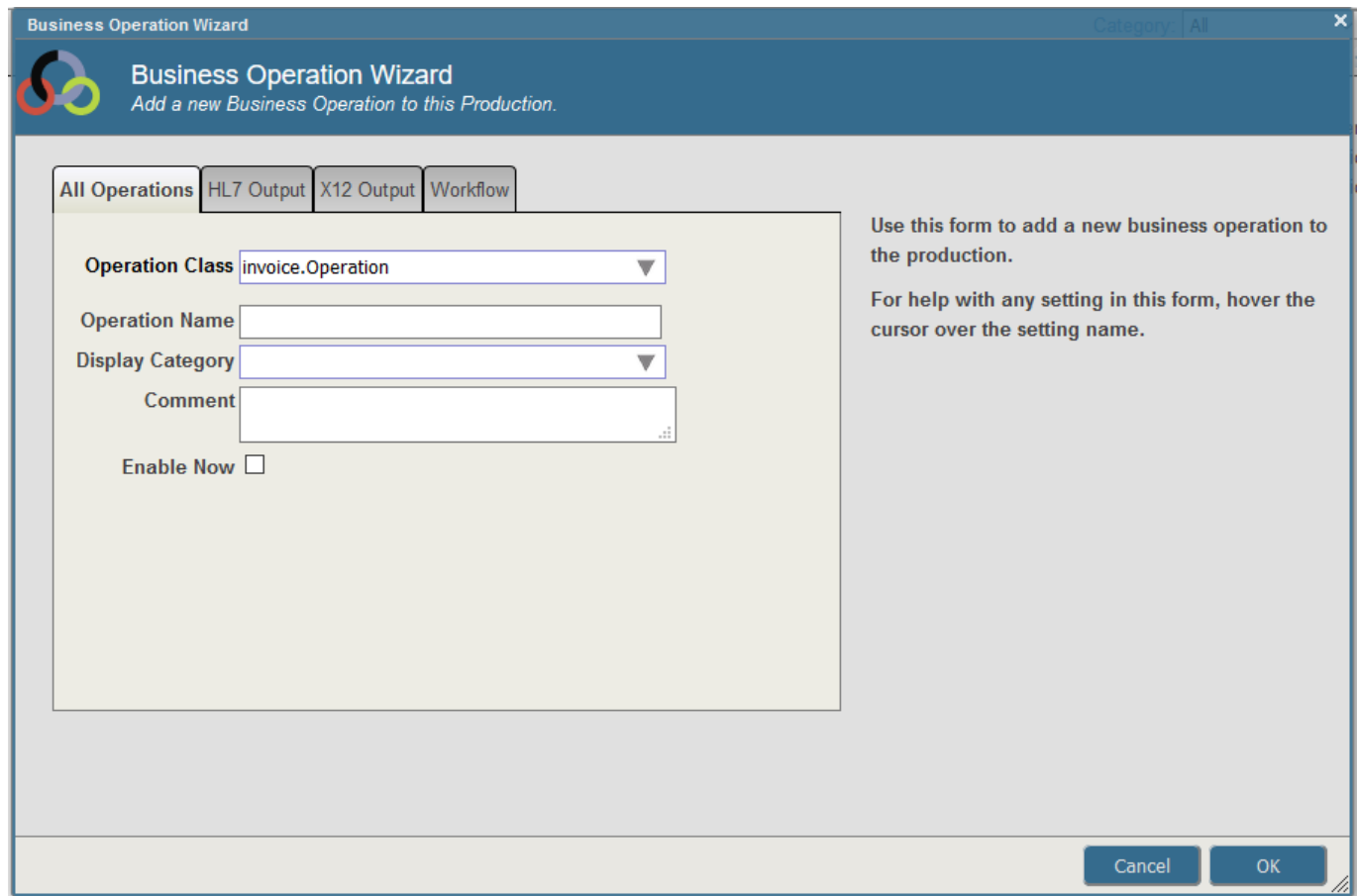
Class Type: ☐ Persistent  
☐ Persistent using one-many relationships  
☐ Persistent using indexed one-many relationships  
☐ Persistent using parent-child relationships  
☒ Serial

Proxy Class Package

☒ Create Business Operation  
Business Operation Package invoice.Operation  
Request Object Package invoice.Request  
Response Object Package invoice.Response

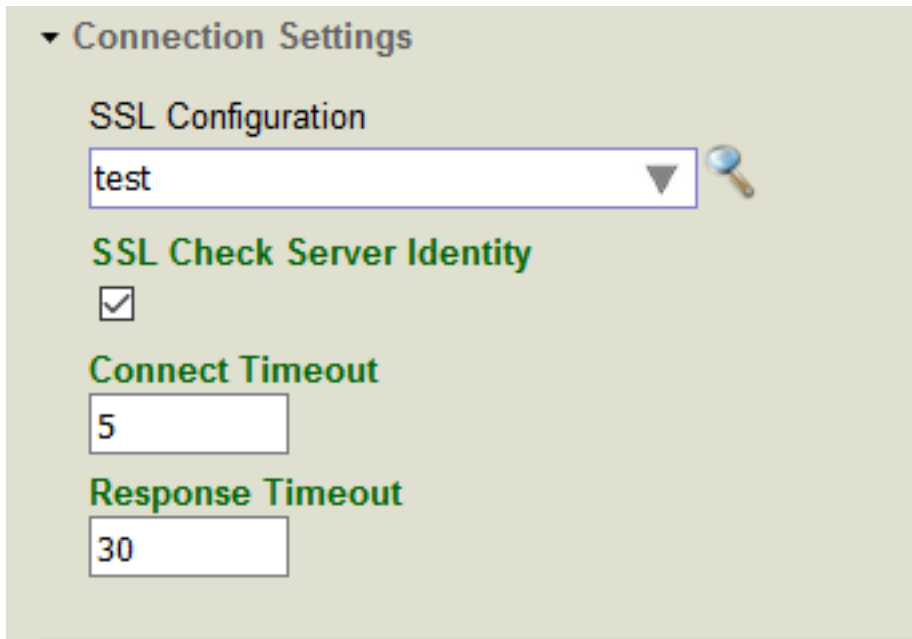


As the result, we get ready code for making requests to and processing responses from our payment system. Let ' s configure an operation in the portal and specify its class:



The screenshot shows the 'Business Operation Wizard' dialog box. The title bar includes the text 'Business Operation Wizard' and a close button. Below the title bar is a header area with the InterSystems logo and the text 'Business Operation Wizard' and 'Add a new Business Operation to this Production.' The main area contains a tabbed interface with four tabs: 'All Operations', 'HL7 Output', 'X12 Output', and 'Workflow'. The 'All Operations' tab is selected. Inside this tab, there is a form with the following fields: 'Operation Class' (a dropdown menu showing 'invoice.Operation'), 'Operation Name' (a text input field), 'Display Category' (a dropdown menu), 'Comment' (a text area), and 'Enable Now' (a checkbox). To the right of the form, there is instructional text: 'Use this form to add a new business operation to the production.' and 'For help with any setting in this form, hover the cursor over the setting name.' At the bottom right of the dialog, there are 'Cancel' and 'OK' buttons.

Let ' s now mount the SSL configuration (it should be created in advance through System Management Portal – System Administration – Security – SSL/TLS Configurations):



The screenshot shows the 'Connection Settings' dialog box in InterSystems Ensemble. It has a title bar with a dropdown arrow. Below the title bar, there are four sections: 'SSL Configuration' with a dropdown menu showing 'test' and a magnifying glass icon; 'SSL Check Server Identity' with a checked checkbox; 'Connect Timeout' with a text box containing '5'; and 'Response Timeout' with a text box containing '30'.

Done! All we need to do now is to call the operation from a business process. In our case, we have two such operations: for the information part and the payment part. In the end, we didn't have to write a single line of code for interacting with the payment system.

That's it, the integration is completed.

However, there is also a separate process that is used for sending PUSH notifications using built-in Ensemble tools, a separate process for obtaining SFTP registries from the payment system for receipt generation, a process for generating PDF receipts, but they all deserve a separate article.

As the result, we spent just a couple of weeks to implement all this (including the time needed to familiarize ourselves with the new technology).

Of course, this InterSystems product is not perfect (nothing is perfect). While working on our implementations, we faced a lot of difficulties, and the lack of documentation for Ensemble didn't help at all. However, in our case, the technology proved to be efficient and worked very well for our purposes. Kudos to the company for supporting young and ambitious developers and their readiness to help. We definitely plan to release new products based on this technology.

We have already launched an app based on this technology, and web version is under way.

Links: [Google Play](#) , [App Store](#)

[#Interoperability](#) [#Studio](#) [#Ensemble](#)

---

Source

URL: <https://community.intersystems.com/post/how-we-learned-stop-worrying-and-love-intersystems-ensemble>