Article
José Pereira · Jun 5, 2022 6m read

Open Exchange

# The Flow Editor application

# IRIS Megazord

In the first article about IRIS Megazord, Henrique explains what drove us to create such an application. It basic is a composition of these previous project which we did:

- History Monitor
- Message Viewer
- ZPM Explorer

But we also started the development of a new feature, called **Flow Editor**. In this article we are going to know more about it.

# IRIS Flow Editor

The aim of this feature is to test a new way to create IRIS Interoperability productions, as Henrique explained in the previous article.

In this way, users are presented to a graphical editor in which they can express information processing flows by drawing graphs. Each node in those graphs have one on these responsibilities:

- Grab data from a source
- Send data to a target
- Receive data, process it and forward it to another node

*Sample flow*

After defining a flow, users should ask for a IRIS Interoperability production generation. In this generation process, each of those nodes are mapped to a Business Service or a Business Operation.

*IRIS Interoperability production created from the previous sample flow*

We are planning to use Business Process in future.

In the next sections, we are going to go deeper in the architecture of how this mapping process is done.

## Nodes

As said before, each node in the flow is responsible for a task - grabbing, sending or processing data.

Nodes which grab data are mapped to Business Services. Nodes which send or process data are mapped to Business Operations.

All Business Services and Operations created by Flow Editor, must inherit from dc.irisflow.components.IrisFlowBusinessService and dc.irisflow.components.IrisFlowBusinessOperation, respectively. In short, these classes provide callbacks for creation and consumption of data in the flow, and define a target for the next Business Host that must be called.

*Target setting presented in all Business Hosts created by the Flow Editor*

The Flow Editor mapping process was designed to allow that any existing IRIS Interoperability Business Host (Services and Operations) could be adapted to be used. Such adaptation is done by creating new classes inheriting from the original Business Host and from the Flow Editor ones.

For instance, we had inherited classes from the IRIS Interoperability library to process files and emails. Classes available in the community are eligible as well, like the Telegram one created by Nikolay Soloviev.

Currently we had 9 nodes available:

- FromFHIRaaS: uses the dc.irisflow.components.fhiraas.FHIRaaSInboundAdapter Business Adapter (developed in this project) to send request to a **FHIRaaS account** in order to get FHIR resources and send them to is output
- FromTelegram: uses the Telegram.TelegramInboundAdapter Business Adapter to receive messages from a Telegram Bot sending them to a target
- ToTelegram: Sends to a Telegram Bot data from its input
- FromEmail(POP3): uses the EnsLib.EMail.InboundAdapter Business Adapter to grab emails and send them to its output
- ToEmail: uses the EnsLib.EMail.OutboundAdapter Business Adapter to send emails using its input in the email body
- FromFile: uses the EnsLib.File.PassthroughService Business Service to read files in a directory and send them to its output
- ToFile: uses the EnsLib.File.PassthroughOperation Business Operation to write its input in files
- ObjectScriptOperation: let users to define custom ObjectScript code for inputs come from another nodes and sends the processed data in its output
- SimpleEchoOperation: just add a prefix in its input and sends it for its output - for test only purposes

*Currently flow nodes*

## Generic messages

One of the simplifications that we would like to introduce in this project was the lack of need for message definitions. So we defined a generic message template, the dc.irisflow.components.GenericMessage class.

This class inherits from Ens.StreamContainer, which gives a generic stream storage for data.

So, all data grabbed, sent or processed by Business Hosts created from Flow Editor, uses data stored in a stream, in a property called Stream.

Besides the Stream property, the generic message also has another one called Context. This property was created to store any data which acts as a metadata in Flow nodes.

For instance, the ToTelegram node accepts a property in the Context called ChatId. If that property is present and with a valid Telegram chat id, then such value is used to identify the recipient of the message.

## Flow examples

Here we are going to see some animations showing some examples of flows created in the Flow Editor.

## FHIRaaS patient notification

In this project, we introduced the FHIRaaSService and the FHIRaaSInboundAdapter. These classes let users grab FHIR resources from a FHIRaaS account and process it in a flow.

In this example, a Patient resource is grabbed from an ID using the FromFHIRaaS node.

Then, this resource is sent to the ObjectScriptOperation. This node lets you to define custom ObjectScript code for process data came from its input and its context, in properties Expression and ContextExpression, respectively.

For this example, the following pieces of code were used for Expression and ContextExpression:

```
Set input = {}.%FromJSON(input)
Do ##class(Ens.Util.Trace).WriteTrace("user",$classname(),"","Patient ID: "_input.id)
Quit "Hi "_input.name.%Get(0).given.%Get(0)_"! This is a reminder for your appointmen
t!"
```

```
Set input = {}.%FromJSON(input)
Set record = $G(^MyDatabase(input.id))
If (record '= "") {
    Set email = $LG(record, 1)
    Set telegram = $LG(record, 2)
    Do ##class(Ens.Util.Trace).WriteTrace("user",$classname(),"",email)
    Do ##class(Ens.Util.Trace).WriteTrace("user",$classname(),"",telegram)

    Set st = ##class(Ens.Config.Credentials).GetCredentialsObj(.credObj, "", "Ens.Con
fig.Credentials", telegram)
    Set context.ChatId = credObj.Password
} Else {
    Do ##class(Ens.Util.Trace).WriteTrace("user", $classname(), "", "No record for pa
tient "_input.id)
}
Quit context
```

The Expression code extracts the Patient ID and Name from the FHIR resource sent to its input.

The CodeExpression also extracts the Patient ID but here it is used to query in a global the Telegram chat id stored in a IRIS Credential. After it retrieves the chat id, this code adds it to the Context property of the generated message sent to the output.

Finally, the processed message is sent to the ToTelegram node, which retrieves the text stored in the Stream property and sends it to the chat id stored in the ChatId property from the Context property from the generic message.

By this way we set up a simple flow which could be used to notify patients stored in a FHIRaaS account.

*FHIRaaS sample*

## Telegram echo

The first one is a simple application which grabs data coming from a Telegram Bot, adds a prefix to it and sends it back to the same Bot.

*Sample of a simple echo application using a Telegram Bot*

# Conclusion

In this article, a feature of IRIS Megazord, the Editor Flow was detailed.

As we saw, we can use any existing Business Hosts and Adapters created for IRIS Interoperability to be used in a different way for production creation. And new ones could be developed as well.

Also this is an experimental project, so your feedback and contribution are always welcome!

#Interoperability #InterSystems IRIS
Check the related application on InterSystems Open Exchange