Article
José Pereira · Sep 15, 2021 10m read

Implementing an IMAP Client in InterSystems IRIS - part II

In the first part we got a quick introduction on the IMAP protocol commands, now it's time to use IRIS and implement them and create our own IMAP client!

IRIS Email Framework

The IRIS platform has default interfaces and classes for working with email. Developers originally designed those artifacts for POP3 implementation. However, this doesn ’ t mean that we can ’ t use and extend these interfaces and classes to implement an IMAP client. So let ’ s talk about them:

- %Net.FetchMailProtocol: This is the base class for email retrieval. The IMAP client extends it.
- %Net.MailMessage: This is the MIME message. It extends %Net.MailMessagePart.
- %Net.MailMessagePart: This encapsulates a MIME message part for multipart messages. This class has an array for itself, enabling a tree representation of message subparts.
- %Net.MIMEReader: This utility class has methods to parse a message ’ s MIME content, generating a %Net.MIMEPart instance.
- %Net.MIMEPart: This encapsulates the message ’ s MIME parts and provides methods to get information about them.

Implementing an IMAP Client

In this section, we present implementation details about an IMAP client, an inbound interoperability adapter, and a simple production example. Note that, in favor of saving space, we won ’ t show most implementation methods. Instead, we link to each one ’ s full implementation details. You can find the complete source code on GitHub.

Creating a Basic IMAP Client

As we discussed before, IMAP is a plain text-based protocol over TCP. This means the base code to implement a client for such a protocol is a TCP client.

The IRIS platform provides standard ObjectScript commands to perform I/O operations: OPEN, USE, READ, WRITE, and CLOSE.

Here is a simple example of how to connect to the MS Outlook server, log in, then log out:

```
ClassMethod SimpleTest()
{
    // connection configuration
    SET dev = "|TCP|993"
    SET host = "outlook.office365.com"
    SET port = "993"
    SET mode = "C"
    SET sslConfig = "ISC.FeatureTracker.SSL.Config"
    SET timeout = 30
    // connection to MS Outlook IMAP server
    OPEN dev:(host:port:mode:/TLS=sslConfig):timeout
    THROW:('$TEST) ##class(%Exception.General).%New("Sorry, can't connect...")
    USE dev
    READ resp($INCREMENT(resp)):timeout
    WRITE "TAG1 LOGIN user@outlook.com password", !
    READ resp($INCREMENT(resp)):timeout
    WRITE "TAG2 LOGOUT", !
    READ resp($INCREMENT(resp)):timeout
    CLOSE dev
    // come back to default device (terminal) and prints responses
    USE 0
    ZWRITE resp
}
```

This is its output:

```
USER>d ##class(dc.Demo.Test).SimpleTest()
resp=3
resp(1)="* OK The Microsoft Exchange IMAP4 service is ready. [QwBQ..AA==]"_$c(13,10)
resp(2)="TAG1 OK LOGIN completed."_$c(13,10)
resp(3)="* BYE Microsoft Exchange Server IMAP4 server signing off."_$c(13,10)_"TAG2 O
K LOGOUT completed."_$c(13,10)
```

There are some highlights in this code:

- We set the mode variable to C, which is carriage return mode. This setting is mandatory for IMAP.
- The flag /TLS establishes a secure layer of communication (SSL). We must set this flag value to a valid SSL IRIS connection.
- The OPEN command initiates the connection.
- The special boolean variable $TEST returns 1 when a command with a timeout is successful or 0 if the timeout expires. In this example, if the OPEN command exceeds 30 seconds, the code throws an exception.
- After a connection is established successfully, the command USE owns the TCP device, redirecting all READ and WRITE commands to this device.
- The WRITE command issues commands to the IMAP server, and the READ command gets their output.
- To finish the connection, we must use the CLOSE command.
- After owning the device, all calls to READ and WRITE commands execute on the device specified in the dev variable, after using the USE dev command. To come back to the terminal and write to it again, you need to issue a USE 0 command first.

Each READ command has a limited buffer to store the server response. When the response size exceeds this limit, you need to issue another READ command to read the complete response. Of course, it ’ s possible to increase the buffer size, but a better approach is to be ready to deal with such a situation.

As we discussed before, IMAP requires a tag for each command. This tag is helpful to check if the code retrieved the complete response or if it needs to issue another READ command. In this case, we implement the ReadResponse method to ensure the code reads the whole message.

Implementing the %Net.FetchMailProtocol Interface for IMAP

The %Net.FetchMailProtocol abstract class abstracts email retrieval on the IRIS platform. We implement the following methods:

- Connect: This establishes a connection to the IMAP server and logs in a user.
- GetMailBoxStatus: This gets the size of the mailbox and how many messages are in it.
- GetSizeOfMessages: This gets the size of one or all messages identified by a message number.
- GetMessageUIDArray: This gets an array with one or all message UIDs in the inbox.
- GetMessageUID: This gets the UID corresponding to a message number.
- Fetch: This retrieves a message ’ s content, possibly multipart content, identified by a message number. It retrieves the message content encapsulated in a %Net.MailMessage object.
- FetchFromStream: This is the same as Fetch, but gets content from an encapsulated EML message content in a %BinaryStream object, instead of calling the IMAP server.
- FetchMessage: This is the same as Fetch, but returns specific message headers in ByRef variables.
- FetchMessageInfo: This retrieves only message headers and the text of the message.
- DeleteMessage: This adds a message to the deletion array.
- RollbackDeletes: This cleans up the deletion array.
- QuitAndCommit: This deletes all messages in the deletion array and disconnects from the IMAP server.
- QuitAndRollback: This cleans up the deletion array and disconnects from the IMAP server.
- Ping: This pings the IMAP server to keep the session alive.

First, we create a new class to implement the interface: dc.Demo.IMAP. This class inherits several properties, which we must set to establish a connection to the IMAP server.

We create a helper class as well: dc.Demo.IMAPHelper. This class parses methods for IMAP responses, gets all parts of a multipart message, and stores peripheral features, including a method to send commands and ensure the entire response is read.

The first method we implement is the Connect method. This method establishes a connection to the IMAP server using the configuration encapsulated in the class properties. It issues a login as well. This method uses the IRIS platform ’ s OPEN command to establish the connection to the IMAP server and the IMAP command LOGIN to authenticate to the server.

The next method we implement is GetMailBoxStatus. This method uses the SELECT command to select a mailbox and it brings some additional information as well, like how many messages are in the mailbox.

IMAP doesn ’ t have a ready-to-use command to get the size of all messages. Of course, it ’ s possible to iterate through all messages and sum their sizes. However, this strategy will probably cause slowness issues. So in this implementation, we don ’ t retrieve the size for all messages.

The next method is GetSizeOfMessages. This method gets the size of one or more messages in the inbox. When no message number is defined, this method throws an exception due to the same IMAP limitation we explained for the GetMailBoxStatus method. We use the IMAP command FETCH <messagenumber> (RFC822.SIZE) to retrieve a message size by its number.

The GetMessageUIDArray method comes next, which uses the IMAP commands SELECT and UID SEARCH [ALL ¦ <messagenumber>] and parses its response to get the UID array.

The next method is GetMessageUID. This method gets a UID for a defined message number and uses the same logic as the GetMessageUIDArray method.

Following this is the Fetch method. It uses the IMAP commands SELECT and FETCH <messagenumber> BODY to retrieve message content, which is coded in MIME format. Fortunately, the IRIS platform has a reader for MIME content, the %Net.MIMEReader class. This class gets the message in a stream and returns the parsed message in a %Net.MIMEPart object.

After getting the MIME content, the method creates a %Net.MailMessage object, fills it with data from the %Net.MIMEPart object, and returns it.

The MIME content is encapsulated in a %Net.MIMEPart object that maps into a %Net.MailMessagePart object through the GetMailMessageParts method in the dc.Demo.IMAPHelper class.

The next method is FetchFromStream. This method receives a stream object with an EML message and converts it to a %Net.MailMessage object. This method does not retrieve content from the server.

Following are the FetchMessage and FetchMessageInfo methods, which are special cases of the Fetch method.

The DeleteMessage method marks a message for deletion, whereas the RollbackDeletes method just cleans up the array of messages marked for deletion.

Next is the QuitAndCommit method. It disconnects from the IMAP server and calls the method CommitMarkedAsDeleted for message deletion.

The method QuitAndRollback just disconnects from the IMAP server and cleans up the array of messages marked for deletion.

The last method, Ping, issues a NOOP command to keep the IMAP session alive.

Implementing an Inbound Interoperability Adapter for IMAP

The base class for email interoperability inbound in the IRIS platform is EnsLib.EMail.InboundAdapter. This inbound adaptor requires these configurations:

- The email server host address
- The email server port
- A credential ID which stores the username and password for accessing the server
- An SSL configuration

This class was extended to create a new IMAP inbound adapter class: dc.Demo.IMAPInboundAdapter.

To use this new adapter, we set which mailbox to use in the Mailbox production parameter. Its default value is INBOX.

The implementation is simple, it just overrides the MailServer property and sets its type to dc.Demo.POP3ToIMAPAdapter IMAP client. This adapter maps the POP3 flow to the IMAP one, as the base adapter class was designed for POP3 commands.

Thus, this POP3 to IMAP adapter enables us to perform all the original inbound adapter logic using IMAP commands instead of POP3 commands.

In the dc.Demo.POP3ToIMAPAdapter class, we use the IMAP client IMAPClient of type dc.Demo.IMAP as a proxy for server communication. However, as dc.Demo.POP3ToIMAPAdapter extends %Net.POP3, it must override all abstract methods in %Net.FetchMailProtocol.

Also, we had to implement new methods that the %Net.POP3 client had implemented directly: ConnectPort and FetchMessageHeaders. In the same way, we created ConnectedGet and SSLConfigurationSet methods to set and get properties that %New.POP3 also implemented directly.

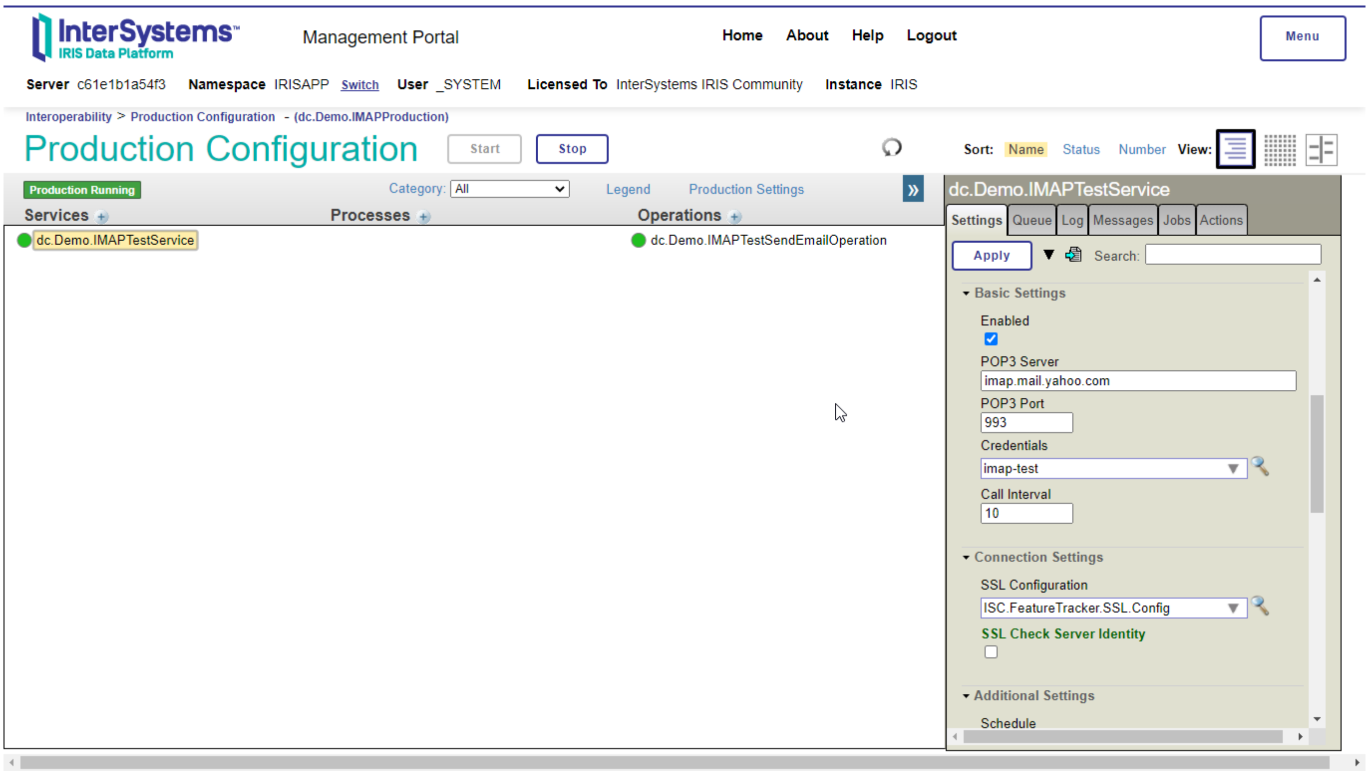
Setting up a Simple Production

To make all these classes work together, we set up a simple production. Check out Creating a Production to get more information about IRIS Interoperability productions.

This production includes a business service and a business operation, which uses the IMAP inbound adapter to check for new messages. This code was inspired by the Demo.Loan.FindRateProduction interoperability sample.

In short, this production:

- Uses the GetMessageUIDArray method to get all available messages in the configured mailbox
- Loops over them, tracing their output, fetched by the Fetch method
- Checks if each message subject matches a criterion — starting with "[IMAP test]"
- Responds to the sender if the message subject matches the criteria, otherwise ignores the message
- Deletes all of the messages so that it won ' t analyze them again



Server c61e1b1a54f3 Namespace IRISAPP Switch User _SYSTEM Licensed To InterSystems IRIS Community Instance IRIS

Interoperability > Production Configuration - (dc.Demo.IMAPProduction)

Production Configuration

Services: dc.Demo.IMAPTestService

Processes

Operations: dc.Demo.IMAPTestSendEmailOperation

dc.Demo.IMAPTestService

Basic Settings

Enabled: ✓

POP3 Server: imap.mail.yahoo.com

POP3 Port: 993

Credentials: imap-test

Call Interval: 10

Connection Settings

SSL Configuration: ISC.FeatureTracker.SSL.Config

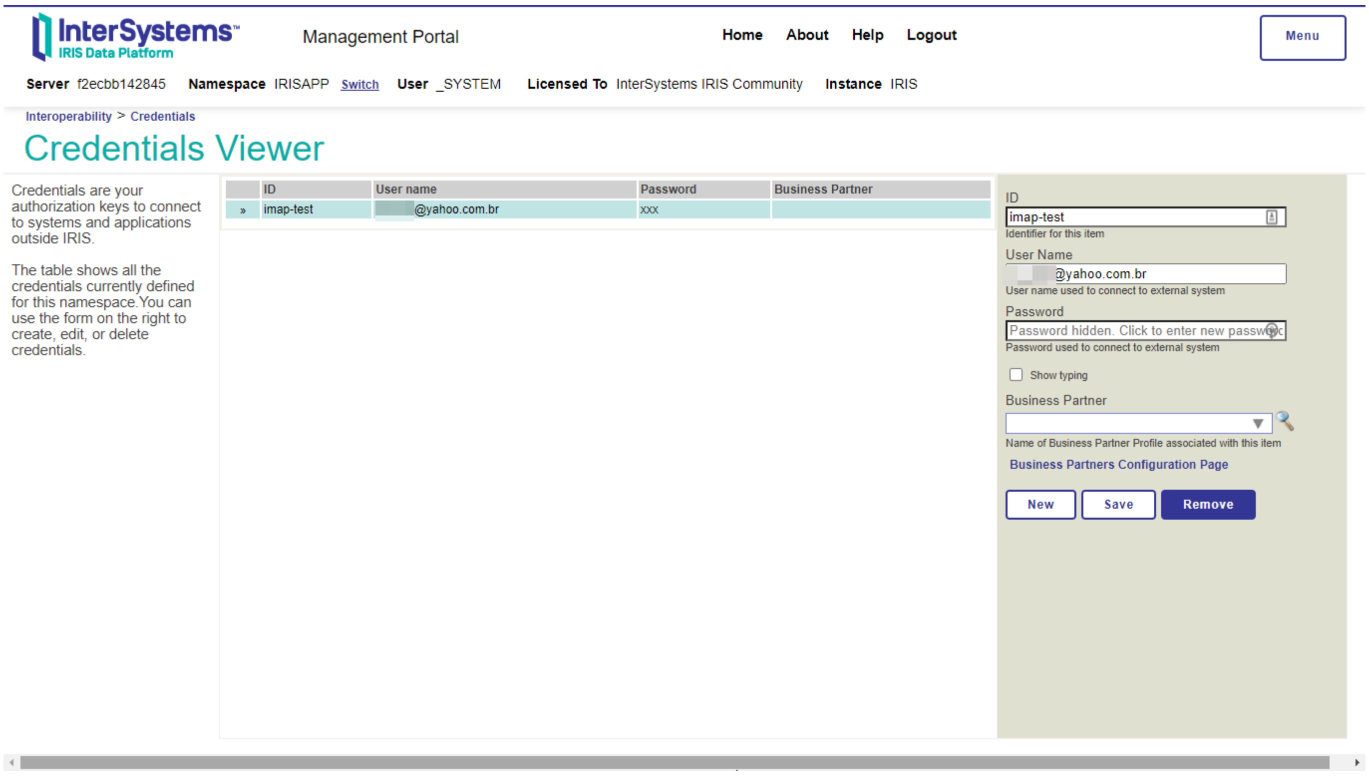
SSL Check Server Identity: ☐

Additional Settings

Schedule

In this example, we configure an IMAP server from Yahoo Mail imap.mail.yahoo.com, on port 993. We also use the default IRIS SSL configuration “ ISC FeatureTacker.SSL.Config ” .

Next, we configure a credential called imap-test containing a username and password, as follows:


InterSystems IRIS Data Platform
Management Portal
Home About Help Logout
Menu
Interoperability > Credentials
New Save Remove
Business Partners Configuration Page

Server f2ecbb142845 Namespace IRISAPP Switch User _SYSTEM Licensed To InterSystems IRIS Community Instance IRIS

Credentials Viewer

Credentials are your authorization keys to connect to systems and applications outside IRIS.

The table shows all the credentials currently defined for this namespace.You can use the form on the right to create, edit, or delete credentials.

	ID	User name	Password	Business Partner
»	imap-test	[redacted]@yahoo.com.br	xxx	

ID: imap-test
Identifier for this item

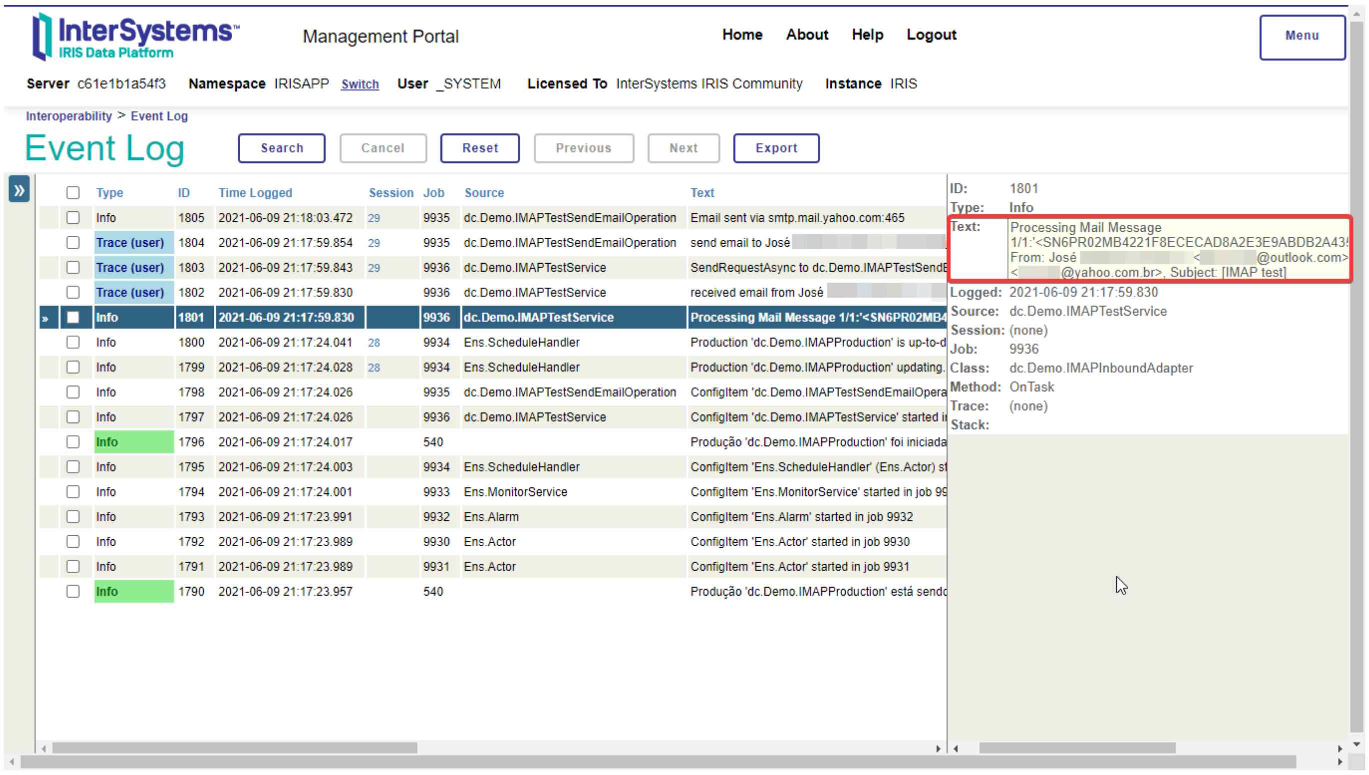
User Name: [redacted]@yahoo.com.br
User name used to connect to external system

Password: Password hidden. Click to enter new passw
Password used to connect to external system

☐ Show typing

Business Partner:
Name of Business Partner Profile associated with this item

As the image below shows, the production starts and keeps querying the IMAP server for new messages. When there are new messages, the inbound adapter grabs their information, like the header and subject, and lets production take further action based on this information.


InterSystems IRIS Data Platform
Management Portal
Home About Help Logout
Menu
Interoperability > Event Log
Search Cancel Reset Previous Next Export

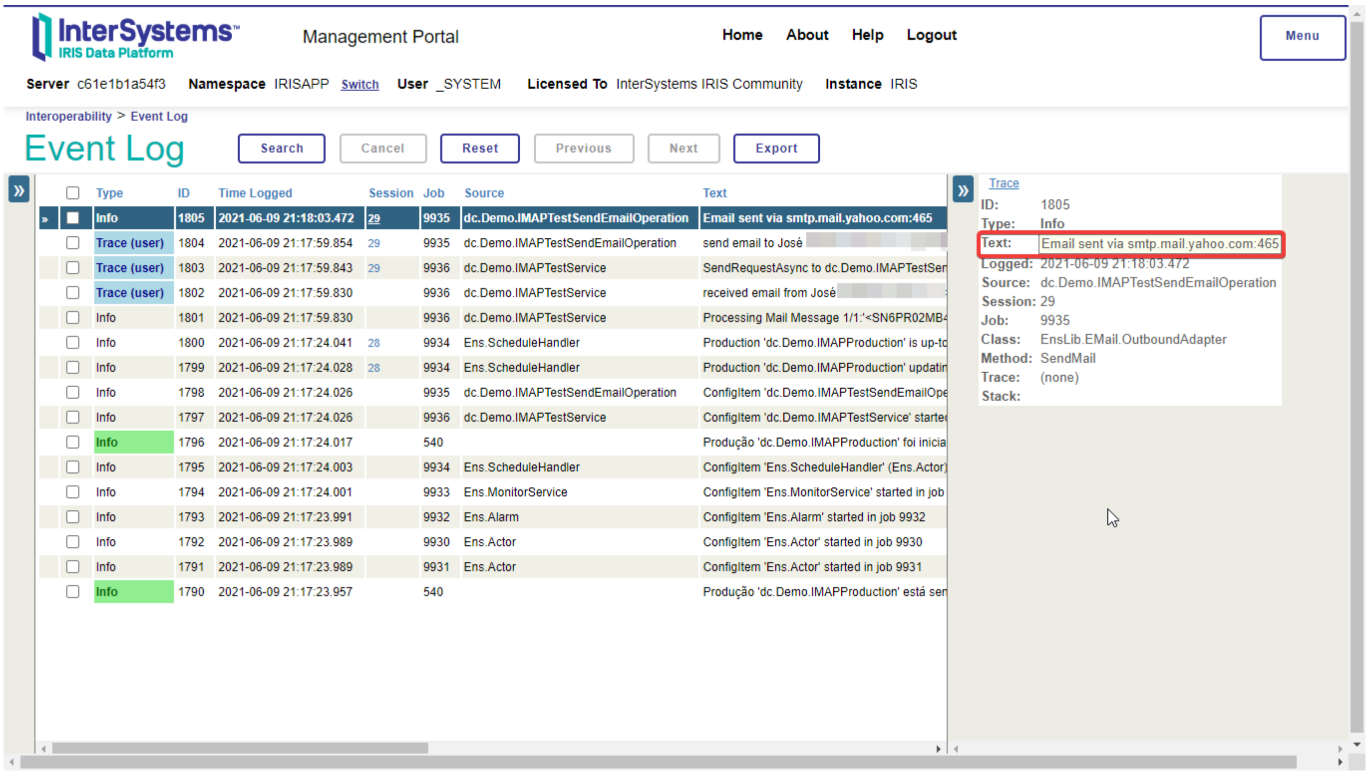
Server c61e1b1a54f3 Namespace IRISAPP Switch User _SYSTEM Licensed To InterSystems IRIS Community Instance IRIS

Event Log

	Type	ID	Time Logged	Session	Job	Source	Text
	Info	1805	2021-06-09 21:18:03.472	29	9935	dc.Demo.IMAPTestSendEmailOperation	Email sent via smtp.mail.yahoo.com:465
	Trace (user)	1804	2021-06-09 21:17:59.854	29	9935	dc.Demo.IMAPTestSendEmailOperation	send email to José [redacted]
	Trace (user)	1803	2021-06-09 21:17:59.843	29	9936	dc.Demo.IMAPTestService	SendRequestAsync to dc.Demo.IMAPTestSendE
	Trace (user)	1802	2021-06-09 21:17:59.830		9936	dc.Demo.IMAPTestService	received email from José [redacted]
»	Info	1801	2021-06-09 21:17:59.830		9936	dc.Demo.IMAPTestService	Processing Mail Message 1/1:'<SN6PR02MB4
	Info	1800	2021-06-09 21:17:24.041	28	9934	Ens.ScheduleHandler	Production 'dc.Demo.IMAPProduction' is up-to-d
	Info	1799	2021-06-09 21:17:24.028	28	9934	Ens.ScheduleHandler	Production 'dc.Demo.IMAPProduction' updating.
	Info	1798	2021-06-09 21:17:24.026		9935	dc.Demo.IMAPTestSendEmailOperation	ConfigItem 'dc.Demo.IMAPTestSendEmailOpera
	Info	1797	2021-06-09 21:17:24.026		9936	dc.Demo.IMAPTestService	ConfigItem 'dc.Demo.IMAPTestService' started i
	Info	1796	2021-06-09 21:17:24.017		540		Produção 'dc.Demo.IMAPProduction' foi iniciada
	Info	1795	2021-06-09 21:17:24.003		9934	Ens.ScheduleHandler	ConfigItem 'Ens.ScheduleHandler' (Ens.Actor) st
	Info	1794	2021-06-09 21:17:24.001		9933	Ens.MonitorService	ConfigItem 'Ens.MonitorService' started in job 99
	Info	1793	2021-06-09 21:17:23.991		9932	Ens.Alarm	ConfigItem 'Ens.Alarm' started in job 9932
	Info	1792	2021-06-09 21:17:23.989		9930	Ens.Actor	ConfigItem 'Ens.Actor' started in job 9930
	Info	1791	2021-06-09 21:17:23.989		9931	Ens.Actor	ConfigItem 'Ens.Actor' started in job 9931
	Info	1790	2021-06-09 21:17:23.957		540		Produção 'dc.Demo.IMAPProduction' está sendo

ID:	1801
Type:	Info
Text:	Processing Mail Message 1/1:'<SN6PR02MB4221F8ECECAD8A2E3E9ABDB2A43 From: José [redacted] <[redacted]@outlook.com> <[redacted]@yahoo.com.br>, Subject: [IMAP test]
Logged:	2021-06-09 21:17:59.830
Source:	dc.Demo.IMAPTestService
Session:	(none)
Job:	9936
Class:	dc.Demo.IMAPInboundAdapter
Method:	OnTask
Trace:	(none)
Stack:	

In this example, the production checks if the message subject starts with "[IMAP test]" and sends back a message to the sender.



Server c61e1b1a54f3 **Namespace** IRISAPP Switch **User** _SYSTEM **Licensed To** InterSystems IRIS Community **Instance** IRIS

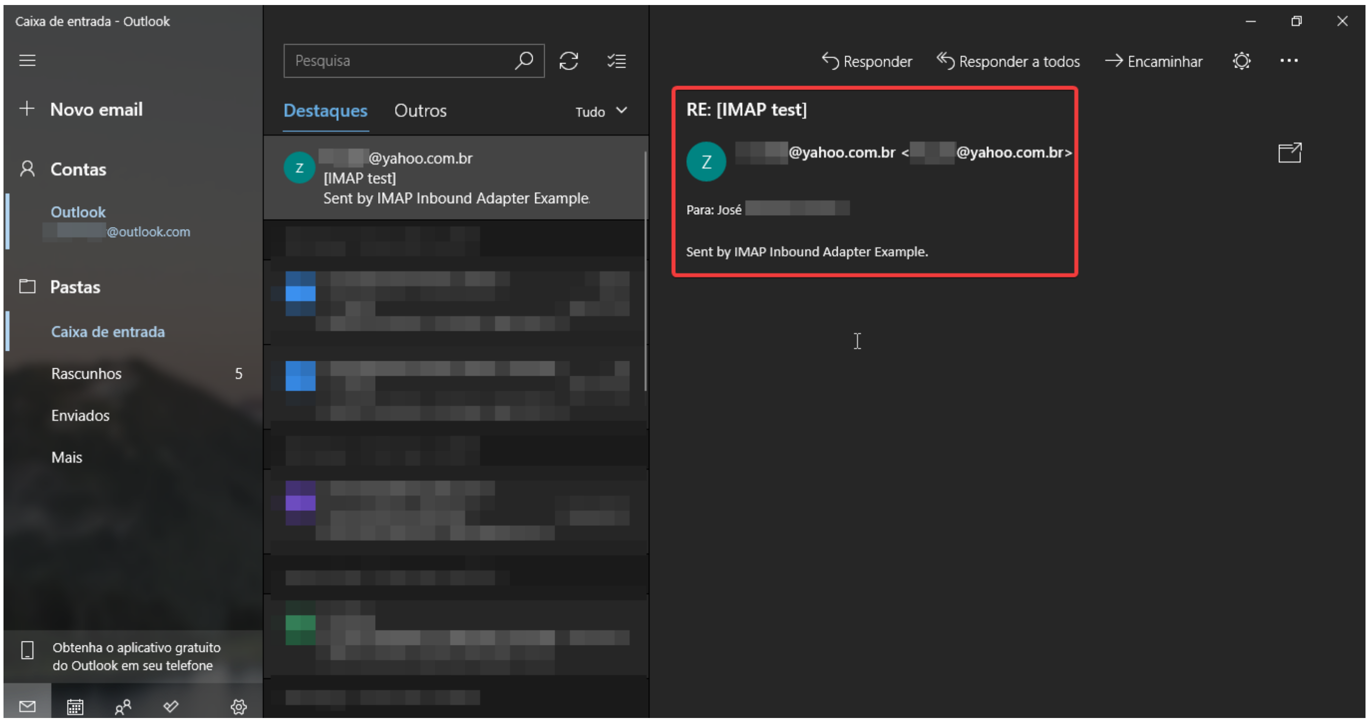
Interoperability > Event Log

Event Log

Type	ID	Time Logged	Session	Job	Source	Text
Info	1805	2021-06-09 21:18:03.472	29	9935	dc.Demo.IMAPTestSendEmailOperation	Email sent via smtp.mail.yahoo.com:465
Trace (user)	1804	2021-06-09 21:17:59.854	29	9935	dc.Demo.IMAPTestSendEmailOperation	send email to José [redacted]
Trace (user)	1803	2021-06-09 21:17:59.843	29	9936	dc.Demo.IMAPTestService	SendRequestAsync to dc.Demo.IMAPTestSer
Trace (user)	1802	2021-06-09 21:17:59.830		9936	dc.Demo.IMAPTestService	received email from José [redacted]
Info	1801	2021-06-09 21:17:59.830		9936	dc.Demo.IMAPTestService	Processing Mail Message 1/1:'<SN6PR02MB4
Info	1800	2021-06-09 21:17:24.041	28	9934	Ens.ScheduleHandler	Production 'dc.Demo.IMAPProduction' is up-to
Info	1799	2021-06-09 21:17:24.028	28	9934	Ens.ScheduleHandler	Production 'dc.Demo.IMAPProduction' updatin
Info	1798	2021-06-09 21:17:24.026		9935	dc.Demo.IMAPTestSendEmailOperation	ConfigItem 'dc.Demo.IMAPTestSendEmailOpe
Info	1797	2021-06-09 21:17:24.026		9936	dc.Demo.IMAPTestService	ConfigItem 'dc.Demo.IMAPTestService' starte
Info	1796	2021-06-09 21:17:24.017		540		Produção 'dc.Demo.IMAPProduction' foi inicia
Info	1795	2021-06-09 21:17:24.003		9934	Ens.ScheduleHandler	ConfigItem 'Ens.ScheduleHandler' (Ens.Actor)
Info	1794	2021-06-09 21:17:24.001		9933	Ens.MonitorService	ConfigItem 'Ens.MonitorService' started in job
Info	1793	2021-06-09 21:17:23.991		9932	Ens.Alarm	ConfigItem 'Ens.Alarm' started in job 9932
Info	1792	2021-06-09 21:17:23.989		9930	Ens.Actor	ConfigItem 'Ens.Actor' started in job 9930
Info	1791	2021-06-09 21:17:23.989		9931	Ens.Actor	ConfigItem 'Ens.Actor' started in job 9931
Info	1790	2021-06-09 21:17:23.957		540		Produção 'dc.Demo.IMAPProduction' está ser

Trace

ID:	1805
Type:	Info
Text:	Email sent via smtp.mail.yahoo.com:465
Logged:	2021-06-09 21:18:03.472
Source:	dc.Demo.IMAPTestSendEmailOperation
Session:	29
Job:	9935
Class:	EnsLib.EMail.OutboundAdapter
Method:	SendMail
Trace:	(none)
Stack:	



[redacted]@outlook.com

[redacted]@yahoo.com.br
[IMAP test]
Sent by IMAP Inbound Adapter Example.

[redacted]

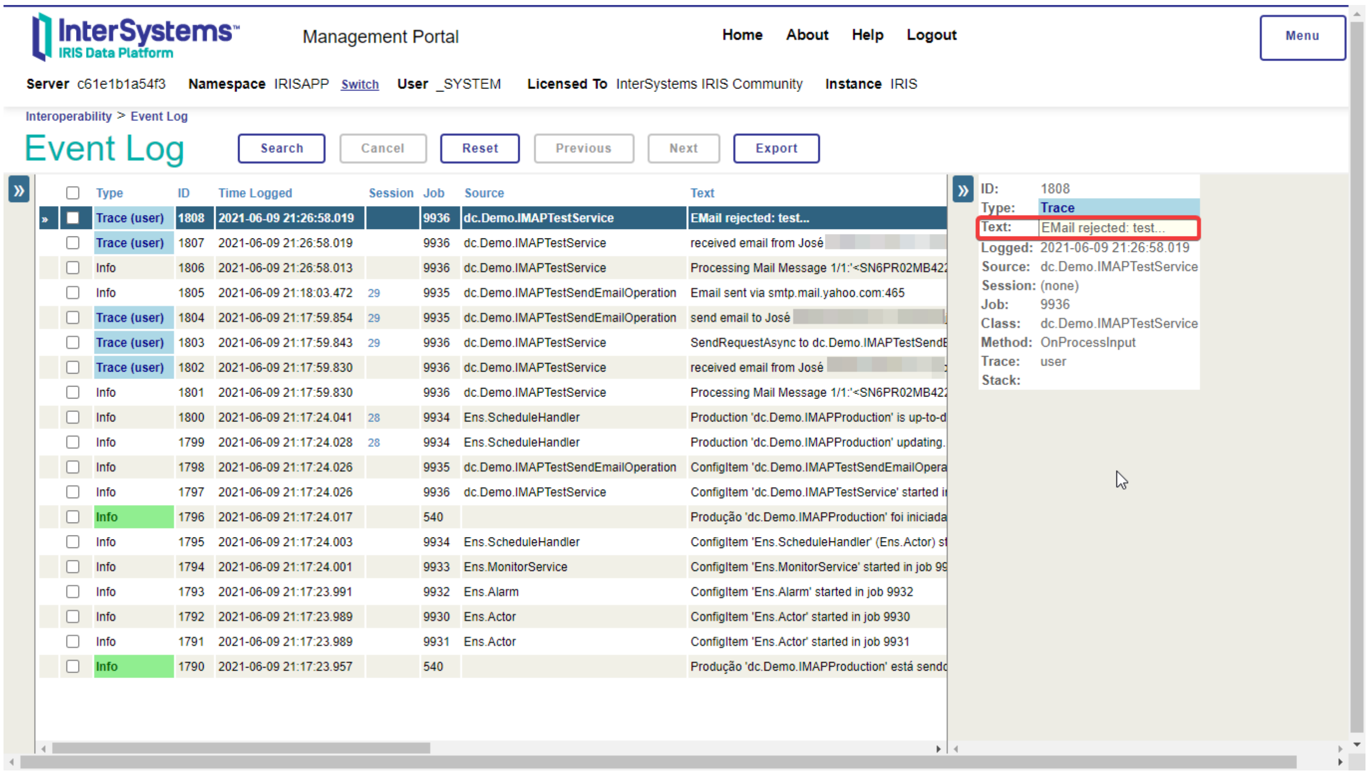
RE: [IMAP test]

[redacted]@yahoo.com.br <[redacted]@yahoo.com.br>

Para: José [redacted]

Sent by IMAP Inbound Adapter Example.

When a message doesn ’ t match the criteria, production just ignores it.


InterSystems IRIS Data Platform
Management Portal
Home About Help Logout
Menu
Search Cancel Reset Previous Next Export

Server c61e1b1a54f3 **Namespace** IRISAPP Switch **User** _SYSTEM **Licensed To** InterSystems IRIS Community **Instance** IRIS

Interoperability > Event Log

Event Log

Type	ID	Time Logged	Session	Job	Source	Text
Trace (user)	1808	2021-06-09 21:26:58.019		9936	dc.Demo.IMAPTestService	EMail rejected: test...
Trace (user)	1807	2021-06-09 21:26:58.019		9936	dc.Demo.IMAPTestService	received email from José [redacted]
Info	1806	2021-06-09 21:26:58.013		9936	dc.Demo.IMAPTestService	Processing Mail Message 1/1:'<SN6PR02MB422
Info	1805	2021-06-09 21:18:03.472	29	9935	dc.Demo.IMAPTestSendEmailOperation	Email sent via smtp.mail.yahoo.com:465
Trace (user)	1804	2021-06-09 21:17:59.854	29	9935	dc.Demo.IMAPTestSendEmailOperation	send email to José [redacted]
Trace (user)	1803	2021-06-09 21:17:59.843	29	9936	dc.Demo.IMAPTestService	SendRequestAsync to dc.Demo.IMAPTestSendE
Trace (user)	1802	2021-06-09 21:17:59.830		9936	dc.Demo.IMAPTestService	received email from José [redacted]
Info	1801	2021-06-09 21:17:59.830		9936	dc.Demo.IMAPTestService	Processing Mail Message 1/1:'<SN6PR02MB422
Info	1800	2021-06-09 21:17:24.041	28	9934	Ens.ScheduleHandler	Production 'dc.Demo.IMAPProduction' is up-to-d
Info	1799	2021-06-09 21:17:24.028	28	9934	Ens.ScheduleHandler	Production 'dc.Demo.IMAPProduction' updating.
Info	1798	2021-06-09 21:17:24.026		9935	dc.Demo.IMAPTestSendEmailOperation	ConfigItem 'dc.Demo.IMAPTestSendEmailOpera
Info	1797	2021-06-09 21:17:24.026		9936	dc.Demo.IMAPTestService	ConfigItem 'dc.Demo.IMAPTestService' started i
Info	1796	2021-06-09 21:17:24.017		540		Produção 'dc.Demo.IMAPProduction' foi iniciada
Info	1795	2021-06-09 21:17:24.003		9934	Ens.ScheduleHandler	ConfigItem 'Ens.ScheduleHandler' (Ens.Actor) st
Info	1794	2021-06-09 21:17:24.001		9933	Ens.MonitorService	ConfigItem 'Ens.MonitorService' started in job 99
Info	1793	2021-06-09 21:17:23.991		9932	Ens.Alarm	ConfigItem 'Ens.Alarm' started in job 9932
Info	1792	2021-06-09 21:17:23.989		9930	Ens.Actor	ConfigItem 'Ens.Actor' started in job 9930
Info	1791	2021-06-09 21:17:23.989		9931	Ens.Actor	ConfigItem 'Ens.Actor' started in job 9931
Info	1790	2021-06-09 21:17:23.957		540		Produção 'dc.Demo.IMAPProduction' está send

ID:	1808
Type:	Trace
Text:	EMail rejected: test...
Logged:	2021-06-09 21:26:58.019
Source:	dc.Demo.IMAPTestService
Session:	(none)
Job:	9936
Class:	dc.Demo.IMAPTestService
Method:	OnProcessInput
Trace:	user
Stack:	

Conclusion

In this article, we discussed an IMAP client implementation. First, we explored some essential background on IMAP and its main commands. Then, we detailed the implementation, covering the client itself and how to connect it to the IRIS platform. We also presented an extension to the default interoperability adapter to use IMAP, and a simple production example.

Now that you know more about IMAP and its settings and you know how to connect it to IRIS, you can set up email capabilities in your applications. To learn more about the IMAP topics we discussed here, explore the resources below.

Resources

- Atmail ’ š IMAP 101: Manual IMAP Sessions
- Fastmail ’ š Why is IMAP better than POP?
- IETF ’ š Internet Message Access Protocol
- IETF ’ š Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies
- InterSystems ’ I/O Devices and Commands
- InterSystems ’ Using the Email Inbound Adapter
- Nylas ’ Everything you need to know about IMAP

#InterSystems IRIS

Source URL: https://community.intersystems.com/post/implementing-imap-client-intersystems-iris-part-ii