

A woman with dark hair, wearing a light-colored shirt, is shown in profile from the chest up, looking down at a tablet computer she is holding with both hands. The background is a blurred cityscape with tall buildings. Overlaid on the lower left of the image is a semi-transparent financial chart with orange data points and lines, showing various numerical values.

# Billing and Collection 2.3.0

## User Guide

# TOC

Overview .....	6
Installing Billing and Collection .....	10
Prerequisites .....	10
Payments Management .....	15
Invoices .....	19
Invoices View .....	20
Invoice Form .....	22
Remove Installments from Invoices .....	24
Bank Statements .....	28
Bank Statements View .....	29
Uploading Bank Statements .....	30
Payments .....	36
Payments View .....	36
Insert Payment Form .....	38
Unallocated Payments .....	40
Unallocated Payments View .....	41
Unallocated Payments Form .....	42
Allocating Payments .....	45
Deallocating Payments .....	51
Outgoing Payments Operations .....	53
Outgoing Payment Requests .....	54
Outgoing Payment Requests View .....	55
Outgoing Payment Requests Form .....	56
Approving Outgoing Payment Requests .....	59
Outgoing Payments Instruction Files .....	65

OPI Files View and Form .....	65
Adding Payment Requests .....	67
Returning Payments .....	72
Direct Debit .....	78
Setting The Solution For DIDE Processing .....	79
Direct Debit Business Workflow .....	82
Direct Debit SEPA .....	86
External Reports .....	97
Direct Debit UK .....	106
Direct Debit UK Functionalities .....	115
<b>Configurations .....</b>	<b>130</b>
Flow Parameters and Scheduled Jobs .....	131
1 Flow Parameters .....	131
2 Scheduled Jobs .....	137
Import Bank Statements .....	140
Payment Group Insert Journey .....	140
Payment Group Journey .....	140
Business Workflow Configurations Actions .....	141
Server Side Script Libraries .....	142
Incoming Payments .....	157
Invoice Generation .....	158
Scheduled Job .....	158
Server Automation Scripts .....	158
Server Automation Script Libraries .....	159
Filtering Configurations .....	169
Automatic Allocation .....	173
Payment Allocation .....	173
Payment Deallocation .....	181
Manual Allocation .....	185

FTOS_PYMT_Payment_EditForm Journey .....	185
Server Automation Scripts .....	188
Server Automation Script Library .....	189
SEPA Direct Debit .....	195
Digital Journeys .....	195
On Demand Server Automation Scripts .....	197
Business Workflow Configuration Actions .....	198
Endpoints .....	200
Processors .....	200
Sequencers .....	200
Server Side Script Libraries .....	201
Scheduled Jobs .....	220
UK Direct Debit .....	221
Digital Journeys .....	221
On Demand Scripts .....	225
Business Workflow Configuration Actions .....	226
Endpoints .....	227
Processors .....	228
Server Side Script Libraries .....	231
Scheduled Jobs .....	248
<b>Outgoing Payments</b> .....	<b>249</b>
Outgoing Payments Admin .....	250
FTOS_PYMT_Payment_EditFormOutgoing Journey .....	250
FTOS_PYMT_OutgoingPaymentFileReadOnly Form Journey .....	252
On Demand Scripts .....	252
Endpoints .....	252
Server Side Script Libraries .....	253
Scheduled jobs .....	257
Outgoing Payments Allocation .....	257
Server Automation Scripts .....	257

Outgoing Payment Deallocation .....	259
Manual Outgoing Payment Requests .....	261
Form Client Side Scripts .....	262
Endpoints .....	264
Server Side Script Libraries .....	265
FTOS_GetAccountData .....	265
FTOS_OutgoingPaymentRequest .....	266
On Demand Scripts .....	271
<b>Billing And Collection Endpoints .....</b>	<b>271</b>
Generate Statements API .....	274
Generate Outgoing Payments API .....	286
Generate UK Direct Debit Mandate API .....	298
<b>Billing Notifications .....</b>	<b>302</b>
Server Side Script Libraries .....	302
Processors .....	305
Business Workflow Configuration Actions .....	305
On Demand Scripts .....	306
Scheduled jobs .....	307
Security Roles .....	307
Digital Assets .....	311
<b>Dashboards .....</b>	<b>313</b>
<b>Glossary .....</b>	<b>315</b>

# Overview

**Billing and Collection** is a highly customizable **FintechOS** solution that enables insurance companies to effectively speed up their quote-to-cash journey by automating routines unique to billing and collection processing, along with offering a series of digital journeys that flawlessly guide users through their manual operations - such as completing payment allocations or introducing a new payment request, in a timely and accurate manner.

With **Billing and Collection**, [invoicing is automatic only](#). **Invoices** are constantly generated into your system for all your insurance products that are in **Active** status. For billing and collection operations, you can use the solution in conjunction with online payment processors as well as bank payment orders. Furthermore, uploading a bank statement file for your bank accounts triggers the automatic parsing, sorting, and matching of the **Payment** data - contained by the file, with the **Invoices** or **Payment Requests** already registered into your system. For payments that respect some rules, such as providing the correct unique identifier for an installment or the number of the policy, the [allocation flow is also completely automated](#) - this way, you can understand faster what invoices have been paid, and which haven't.

On top of that, **Billing and Collection** offers insurers alternative ways to:

- increase safety when handling sensitive information - such as payment data, in comparison with the traditional, paper-based methods.
- implement the flexibility required to accommodate payment constraints along with payment updates, based on policy adjustments.
- achieve high proficiency in keeping track of the ever-growing payment-related data.
- manage their operations in a completely safeguarded and accurate manner.

Additionally, in order to accommodate the differences regarding the direct debit payments, the **Billing and Collection** solution has dedicated workflows for the **Single Euro Payments Area (SEPA)** and for the **UK** financial area.

When necessary, the **Billing and Collection** solution can be personalized in accordance with specific requirements from insurance companies. Thus, implementation time is shortened, while you can make sure that each component fulfills your business needs.

For example using **Billing and Collection** along with different **FintechOS Automation Processors** enables insurance companies to digitize workflows and improve accuracy, while also reducing the amount of time spent on other routine business operations.

The present guide helps you discover how to make the best use out of the **Billing and Collection** solution.

## Business Pain Points

**FintechOS** clients use the **Billing and Collection** solution to respond to different challenges related to:

- time-consuming routines;
- routines more prone to human error when done manually;
- complex recurring billing and collection scenarios;
- the sheer volume of collection activities;
- little time to identify and deal with discrepancies in collections.

## Billing and Collection Key Features

The solution has the following key features:

- The administration of **different payment requests** received from **different sources or systems** like claim payments, commissions payments, broker balance (credit) payments etc.
- Processing bank statements files in various formats - e.g. the MT940 format, used in SEPA area.
- Identifying the outgoing payments (including bank charges) in a bank statements file.
- The automatic generation of a payment instruction file based on the received payment request.
- The allocation of the confirmed outgoing payments (e.g. from a bank statement) to the specific payment request.

- The possibility to send notifications with the payment confirmation, and more.

## Billing and Collection Key Benefits

The benefits of using **Billing and Collection** are the following:

- works easy with volumes;
- processing payments from bank account statements;
- processing payments from online payment processors;
- automated and manual payment allocation and deallocation;
- scales from the simplest of billing and collection models to the most complex ones;
- manages one-time and low usage billing and collection scenarios and also the frequently recurring ones;
- speeds up billing and collection processes by automating routines;
- reduces the risk of human error while handling accounts;
- frees up time for where attention is really needed: selling more insurance!

## Billing and Collection Key Steps

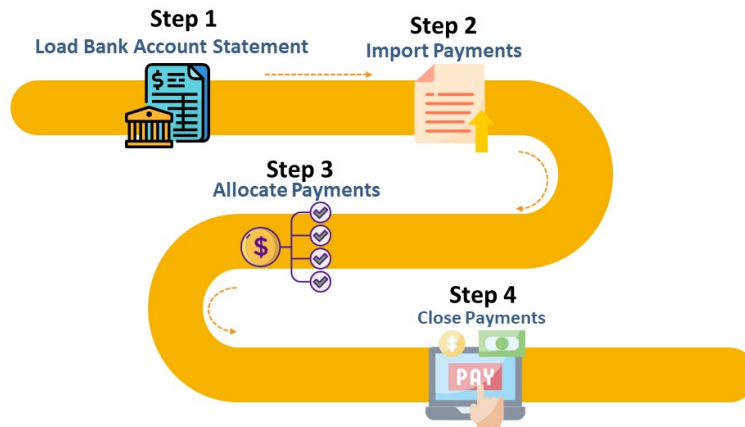
For incoming payments:

1. Load new statement for your bank accounts, obtained from your bank or from an online payments processor.
2. Import payment data. Allocation is automatic for payments with details filled-in correctly.
3. Deal with unallocated payments. Unallocated payments are returned



automatically after a configurable number of days.

4. After manual allocation, close payments.



For outgoing payments:

1. Load new statement for your bank accounts, obtained from your bank.
2. Import payment data.
3. Deal with outgoing payment requests. For recurring outgoing payment requests (such as broker payments) allocation is automatic and configurable.
4. After manual approval, close outgoing payment requests.

For more details about how you can administer payments with Billing and Collection, please consult the [Payments Management](#) page.

### HINT

Integrate **Billing and Collection** with more [FintechOS solutions for insurance](#) in order to make the best of process automation for your company, portfolios, products and clients!

# Installing Billing and Collection

Follow the instructions below to install and configure **Billing and Collection v2.3.0**.

## Prerequisites

Before installing or upgrading to **Billing and Collection v2.3.0**, make sure the following are already installed:

- **HPFI v22.1.0**
- **SySDigitalSolutionPackages v21.2.2301**
- **Core Insurance Master v2.3.0**

### **IMPORTANT!**

If you already have **Billing and Collection Import v2.2.0** installed, then you need to install **Billing And Collection Import Upgrade v2.3.0** instead.

Below you can find instructions for clean installing or upgrading to **Billing and Collection v2.3.0**.

## Billing and Collection v2.3.0 Clean Install

Install the following packages, in this exact order:

1. **Billing and Collection v2.3.0**
2. **Billing and Collection Import v2.3.0**

Follow the steps below:

1. In **Innovation Studio**, import and install the **Billing and Collection** digital solution by following the [standard procedure](#).
2. After installing the **Billing and Collection** digital solution, run the SQL scripts from **Billing and Collection** folder.
3. After installing the **Billing and Collection v2.3.0** digital solution, perform the following configurations:

3.1 Check System Parameter value:

3.1.1 - In **Innovation Studio**, set your context to **Core Insurance Master** digital asset as described in the [Editing Digital Assets](#) page.

3.1.2 - Go to **Main Menu > Admin > System Parameters**.

3.1.3 - Search **BillingUsed** parameter and open it for edit.

3.1.4 - Set the value to 1. Next, click **Save and close** to enable the use of **Billing and Collection** solution.

3.2 Add vault keys for **Portal** app-settings:

Attention!  
Add/modify the Vault keys for the Portal app-settings:  
Identify the following keys and add their values with your SMTP information:

```
{
  "baseUr1Api": "PORTALAPI_URL *",
  "clientApi": "yourClient",
  "userApi": "yourUserName",
  "passwordApi": "youUserPass",
  "SMTP:Port": "****",
  "SMTP:Host": "****",
  "SMTP:EnableSSL": "0",
  "SMTP:User": "****",
  "SMTP>Password": "****",
  "DefaultFromEmail": "****"
}
```

\* URL of the portal site using EBSDefaultAuthentication = EBS  
\*\*\* = your SMTP information

3.3 Add vault keys for **Innovation Studio** app-settings:

**Attention!**

Add/modify the Vault keys for the Studio app-settings:  
Identify the following keys and add their values:

```
{
  "SMTP:Port": "****",
  "SMTP:Host": "****",
  "SMTP:EnableSSL": "0",
  "SMTP:User": "****",
  "SMTP:Password": "****",
  "DefaultFromEmail": "****"
}
```

\*\*\*\* = your SMTP information

**3.4 Add vault keys for Job Server app-settings:****Attention!**

Add/modify the Vault keys for the Job server app-settings:  
Identify the following keys and add their values with your SMTP information:

```
{
  "UploadFolder": "yourPath:\Sites\UploadEBS",
  "AttachmentPath": "yourPath:\Sites\UploadEBS",
  "FileUploadWhiteList":
  ".pdf,.doc,.docx,.xls,.jpg,.jpeg,.xlsx,.dll,.ppt,.pptx,.txt,
  .png,.ttf,.xml",
  "baseUrlApi": "PORTALAPI_URL *",
  "clientApi": "yourClient",
  "userApi": "yourUserName",
  "passwordApi": "yourUserPass",
  "SMTP:Port": "****",
  "SMTP:Host": "****",
  "SMTP:EnableSSL": "0",
  "SMTP:User": "****",
  "SMTP:Password": "****",
  "DefaultFromEmail": "****",
}
```

\* URL of the portal site using EBSDefaultAuthentication = EBS  
\*\*\*\* = your SMTP information

**IMPORTANT!**

The **UploadFolder** and **AttachmentPath** keys are not needed for a job

server installed as a web app. Instead, use the standard configuration steps to allow the job server access to the blob storage used by the other sites (to the same UploadEBS folder).

4. Follow step 1 to import and install the **Billing and Collection Import v2.3.0**.

5. After installing the **Billing and Collection Import v2.3.0** digital solution, run the SQL scripts from the Billing and Collection **Import** folder.

6. After installing the **Billing and Collection Import v2.3.0** digital solution, perform the following configurations:

6.1 Approve all **Insurance Parameters** as described below:

6.1.1 - In the Portal, go to **Main Menu > Settings > Insurance Parameters**.

6.1.2 - Select a **Parameter** record and open it for edit.

6.1.3 - Use the status picker, from top left corner, to change the parameter status from **Draft** to **Approved**.

6.1.4 - Repeat for all **Insurance Parameters** in **Draft** status.

## Billing and Collection v2.3.0 Upgrade

Upgrade to the following packages, in this exact order:

1. **Billing and Collection v2.3.0**
2. **Billing and Collection Import Upgrade v2.3.0**

Follow the steps below:

1. In **Innovation Studio**, import and install the **Billing and Collection v2.3.0** digital solution by following the [standard procedure](#).

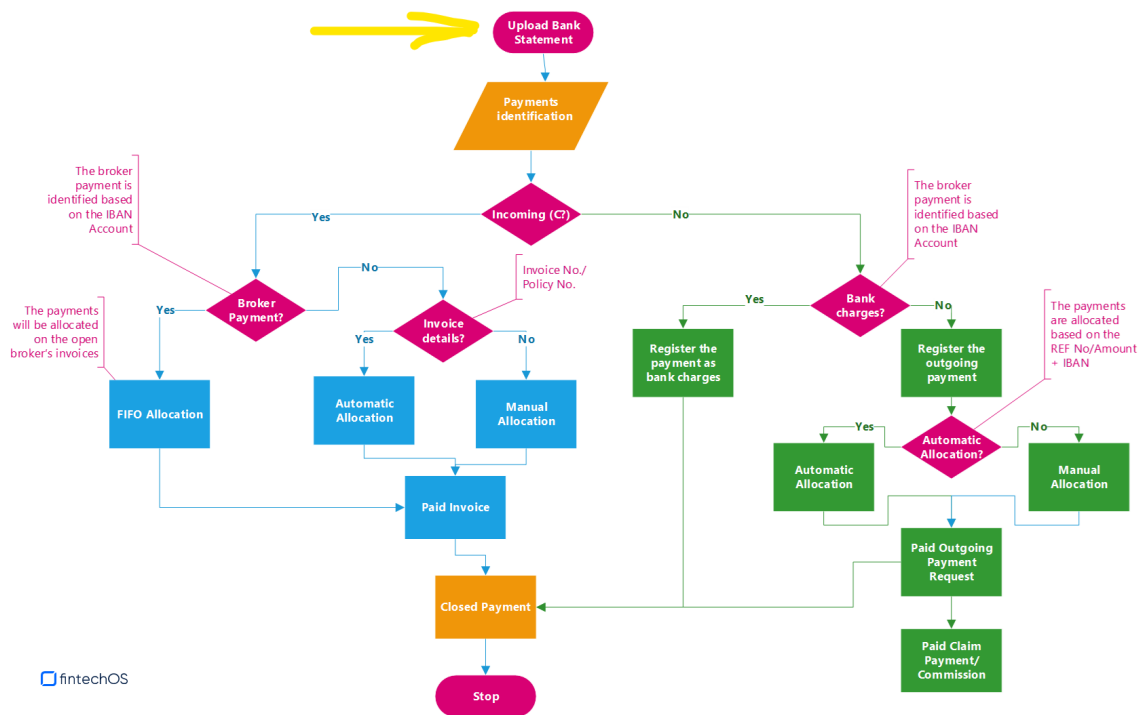
2. After installing the **Billing and Collection v2.3.0** digital solution, run the SQL scripts from **Billing and Collection** folder.

3. After installing the **Billing and Collection v2.3.0** digital solution, perform the configurations described at step 3, inside the [clean install](#) section.
4. Follow step 1 to import and install the **Billing and Collection Import Upgrade v2.3.0**.

# Payments Management

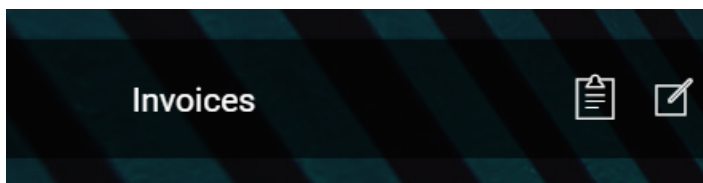
The **Billing and Collection** solution makes payment tracking effortless, it enables you to improve accuracy and completion time for your billing and collection routines and lets you process different types of payments, received from different sources or systems, manually or automatically.

Below, a diagram of the **Billing and Collection** main allocation flows:



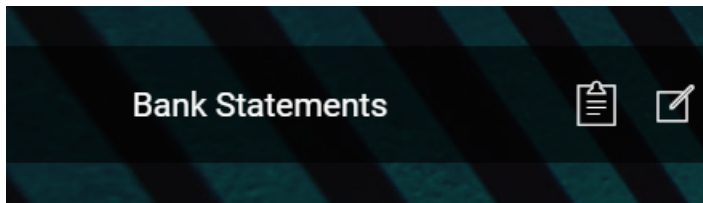
Here is how the solution works:

## Invoices



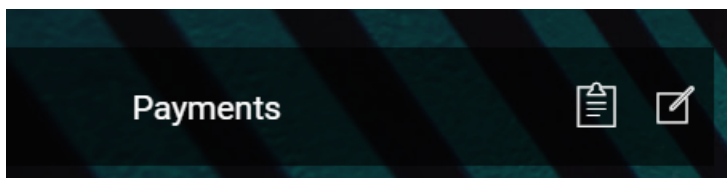
**Invoices** are automatically generated in the system for all your active products, according to the settings you configured for each product. For example a payment schedule is attached to a policy and, consequently, the invoices are generated in the system according to that particular schedule. The **Billing and Collection** automatic invoicing covers multiple scenarios and can scale from simple to complex ones. For more details about invoice settings at the product level, consult the [Insurance Product Factory](#) documentation and, for scheduling payments, consult the [Policy Admin](#) documentation.

## Bank Statements



The **Bank Statements** feature allows you to add payments in the system. Uploading the bank statements for your accounts in the system triggers the automatic parsing, sorting, and matching of the **Payment** data - contained by the file, with your **Invoices** or, depending on the case, with your **Payment Requests**. For payments that respect some rules - such as providing the correct unique identifier for an installment or the number of the policy, the allocation flow is completely automated. You can also add payment data from online payment processors - such as PayU. You can integrate **Billing and Collection** with any payment processor in order to complete the collection operations.

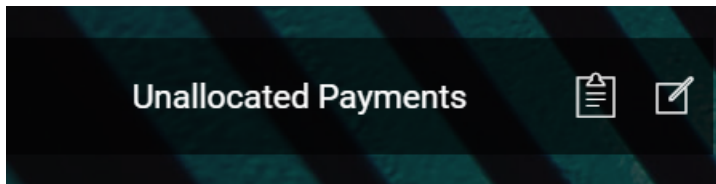
## Payments





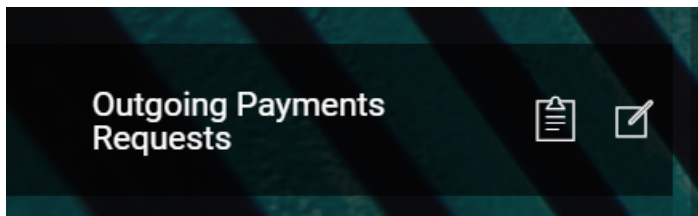
The **Payments** section offers an overview of **all your payments**, incoming and outgoing, with their current business statuses - such as **Unallocated**, **Partially Allocated**, and **Closed**. This view is automatically and continuously updated with new payment data, as it is progressively fed into the system automatically, or registered manually, and processed by the system. This is an all-inclusive view; yet, you can also search and sort your payments for easier processing. For example if you want to view all the **Incoming** payments, you can use the **Search by Payment Category** option and sort all your payment data accordingly.

## Unallocated Payments



**Unallocated Payments** is the section where you can see all incoming payments that the system was unable to allocate automatically - for example a policyholder made a payment without indicating the installment number or the policy number. You can either use the **Allocate** or the **Return** manual flow, in order to deal with the selected unallocated payment. Manual allocation can be done either **backward-looking** or **forward-looking**. You perform **backward-looking** manual allocation when you allocate a payment on an invoice (which is already issued in the system). Or you perform a **forward-looking** manual allocation when you allocate a payment on a future installment (since every policy has an agreed **installments schedule**, also registered in the system). The manual return flow integrates with the outgoing payments flow. For details, see the section below.

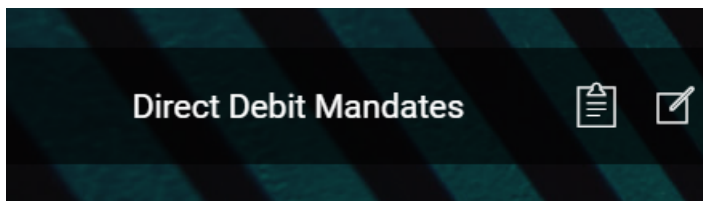
## Outgoing Payment Requests



The [Outgoing Payment Requests](#) functionality eases the management of payment requests received from different sources - such as when you **import bank statement files**, when you return an unallocated payment, when you receive a request through an API call or **manually introduce a request for payment** in the system. For some of your recurring outgoing payments scenarios - like paying brokers commissions, you can configure the system to automatically allocate the outgoing payments. For other scenarios, you use the manual flow that lets you propose, approve or decline outgoing payment requests according to your needs.

For more details about the functionality used to allocate outgoing payments, see also the [Outgoing Payments Instruction Files](#) page.

## Direct Debit Payments



The **Billing and Collection** **Direct Debit** functionality helps insurers to handle direct debit payment operations.

Once the policyholder agreed to pay the premiums by direct debit, the insurer initiates a direct debit activation procedure in order to notify the bank about the payment arrangement. After the mandate is activated by the bank, the insurer must regularly send the necessary payment instructions, in order for the bank to transfer the agreed premium amounts into insurer's accounts.

In order to accommodate the differences regarding the direct debit payments, the **Billing and Collection** solution has dedicated workflows for the **Single Euro Payments Area (SEPA)** and for the **UK** financial area. Each flow has an overview of the direct debit mandates registered in your system - with the newest mandate recorded in the system at the top, allowing you to search and sort the mandates for easier processing. Additionally, each flow has different menu items that display their respective functionalities and help you to handle direct debit payments processing according to SEPA or UK

regulations. The versioning functionality allows you to edit any mandate, and all updates are logged into the system. After editing, you must manually approve the new mandate version.

For direct debit payments that respect some rules (such as correct payment details, payment is due time), the process of billing and collecting is completely automated, and you can read about it on the [Direct Debit SEPA](#) and [Direct Debit UK](#) pages. For configuring either of the flows to be applied to your system, go to the [Direct Debit](#) page.

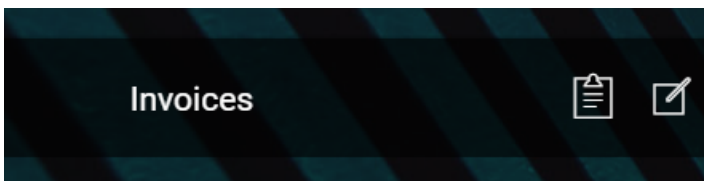
## Customer Notifications

**Customer Notifications** is a feature that allows you to automatically notify your customers about the status of their premium payments, according to their policy. The **Billing and Collection** solution allows you to send emails to the policyholder about the following payment events:

- the generation of the invoice for an installment (according to the payment schedule),
- follow up for unpaid invoices,
- confirmation of the premium payment.

When generating an invoice that contains multiple policies, the notification template (email or pdf) lists all the policies included in the invoice, and the total amount for that invoice. For more details about this functionality, consult the [Billing Notifications](#) page.

## Invoices



An invoice is a request for payment (or payments) based on a contract agreement.

For example the insurer provides the agreed coverage in exchange for premiums, paid by the insured. In this context, at certain dates, the insurer issues requests for the expected payments, towards the insured. Invoicing for a policy coverage is based on the policy installments schedule, previously agreed with the policyholder. Yet, the policy can be adjusted, as agreed by parties. Consequently, the invoicing must attune to the payment updates - either related to the frequency of installments, the changing of the payment method (for example from direct debit to online payment) or, if the case, the currency of the payments.

With the **Billing and Collection** solution, this scenario is covered: if there is an update on the policy related to payments, the invoices are issued according to this update.

**Invoices** are constantly generated into your system for all your insurance products that are in **Active** status. Once a new policy is issued into your system, installment records are automatically generated for that policy in accordance with the policy's appended installment schedule. Issuing an invoice for every installment on that policy is based on the [date parameters set at the product level](#) and the policy settings with regard to premium payments - such as the installment schedule.

All newly created invoice records can be found in the **Invoices** section, inside your **Billing and Collection** module.

**IMPORTANT!**

Invoicing with **Billing and Collection** is [automatic only](#).

## Invoices View

In your portal, in the **Invoices** section, you have an overview of the invoices generated into your system - with the newest invoice at the top. This is an all-inclusive view; yet, you also can search and sort your payments for easier processing. For example if you want to view all the invoices in **Unpaid** status, you can use the **Search by Business Status** option and sort all your invoices accordingly.

INVOICES LIST							
<input type="checkbox"/>	Invoice No.	Invoice Date	Contractor	Invoice Amount	Currency	Due Date	Business Status
<input type="checkbox"/>	REF0000532	13/09/2021		75.00	EUR	31/05/2022	Generated
<input type="checkbox"/>	REF0000530	13/09/2021		250.00	EUR	12/05/2022	Generated
<input type="checkbox"/>	REF0000531	13/09/2021		75.00	EUR	28/02/2022	CLOSED
<input type="checkbox"/>	REF0000529	13/09/2021		250.00	EUR	12/02/2022	Generated
<input type="checkbox"/>	REF0000400	10/08/2021		75.00	EUR	10/02/2022	CLOSED

5 10 20 1 2 3 4 5 ...

Follow the steps to view your invoices:

1. In your **FintechOS Portal**, navigate down the main menu of the **Billing and Collection** solution.
2. From the dropdown list, click **Invoices** to open the **Invoices List**.  
On the **Invoices List** page:
  1. To **inspect** a record from the grid, double-click it.
  2. To **edit** a record from the grid, double-click it and press **Edit**. The editing form allows you only to remove installments, from the selected invoice.

**NOTE**

You can only edit invoices in **Generated**, **Unpaid** or **OnGrace** status.

3. To **delete** a record from the grid, select it and click **Delete**, at the top right corner of the page.

**HINT**

You can export one or more records by pressing **Export**, at the top right corner of your screen.

## Invoice Form

An **invoice generation job** runs daily within the system, verifying all the installments on policies and their payment schedules, for all the insurance products that are in **Active** status. This job generates invoices for all the qualifying installments, taking into account specific billing settings, at policy and product level. When you double-click a record from the **Invoices** section, in order to manage an invoice, you launch the **Invoice Form**. This form allows you to remove any installment that was placed there incorrectly. It also allows you to see details about the invoice and the premiums paid, if the case.

Any **Invoice** contains the following details:

### The Header Section

The screenshot shows the 'Invoice' form header section with the following fields and values:

- Invoice No.: REF0000563
- Broker ID: [Dropdown]
- Broker Name: [Text]
- Invoice Date: 04/10/2021
- Invoice Reference: REF0000563
- Invoice Premium Amount: 25
- Total Taxes: 2.5
- Total Commission: 2.5
- Invoice Amount: 25
- Currency: EUR [Dropdown]
- Payment Type: Broker Collection
- Due Date: 14/10/2021
- Grace Limit Date: 14/10/2021
- Paid Amount: 0
- Unpaid Amount: 25
- Scheduled date: [Text]

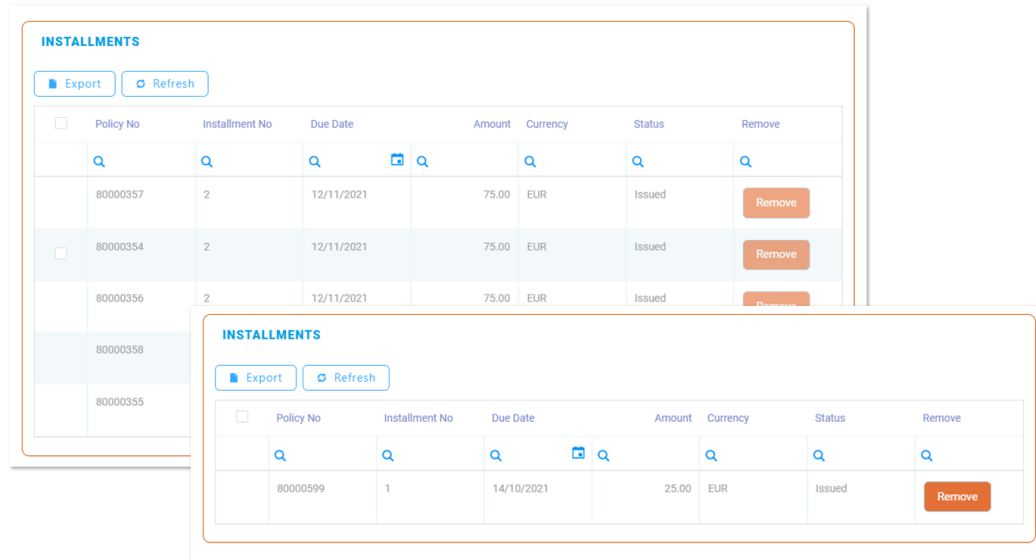
An 'Edit' button is located at the bottom right of the form.

Field Name	Description
Invoice No.	The invoice number.
Contractor	The contractor of the policy.
Broker ID	The Id of the broker - if the invoice is issued for a broker.
Broker Name	The name of the broker - if the invoice is issued for a broker.
Invoice Date	The date when the invoice is created in the system.
Invoice Reference	The invoice reference to be used in the payment flow.

Field Name	Description
Invoice Premium Amount	The invoice premium amount.
Total Taxes	The total taxes calculated for that invoice.
Total Commission	The total commission calculated for that invoice.
Invoice Amount	The total amount registered on the invoice.
Currency	The currency of the installments registered on the invoice.
Payment Type	The type of payment. It can have the following values: <b>Direct Debit</b> , <b>Payment Order</b> ("OP", or "bank transfer"), <b>PayU</b> , or <b>PayUOnTime</b> . The type of payment is set at the policy level.
Due Date	The due date for paying the current installment.
Grace Limit Date	The grace limit period - if any.
Paid Amount	The paid amount.
Unpaid Amount	The unpaid amount.
Scheduled date	The date scheduled for paying the installment, set at the policy level.

For more details about what kind of data is created inside the system, at the moment of issuing a new invoice, consult the **FTOS\_PYMT\_Statement** entity page.

## The Middle Section



The **Installments** section contains details about all the installments registered on that invoice. And there can be one or more installments listed inside this grid - for example for the case when an invoice is generated for a group policy. For the same case (of many installments on the same invoice), you also can search inside this grid by different keywords - such as **Policy No, Installment No, Due Date, or Status**.

Next to each installment record there is a **Remove** button that becomes active when you **Edit** the invoice. For details about editing, consult the [Remove Installments from Invoices](#) section, on this page.

## The Bottom Section

**PREMIUMS COLLECTED**

Export Refresh

Payment ID	Payment Regis...	Payer Name	Amount	Currency	Allocated Amount	Currency	Business Status
🔍	🔍	📅 🔍	🔍	🔍		🔍	🔍
No data							

The **Premiums Collected** section contains details about all the premiums collected on that invoice. And there can be one or more premiums listed inside this grid - for example for the case when an invoice is generated for a group policy. For the same case, of many premiums paid on the same invoice, you also can search inside the grid by different keywords - such as **Payment ID, Payment Registration Date, Payer Name or Business Status**.

More about payments in the [Payments](#) or [Payment Lifecycle](#) pages.

## Remove Installments from Invoices

There are cases when you need to remove an installment for an invoice. For example a policy is adjusted and a new installment schedule is agreed with the policyholder. Consequently, you need to remove the installment generated based on the outdated installment schedule.



**NOTE**

You can only remove installments from invoices in **Generated, Unpaid** or **OnGrace** status.

Follow the steps to remove an installment for an invoice:

## 1. Search and Open the Record

1. In your **FintechOS Portal**, navigate down the main menu of the **Billing and Collection** solution.
2. From the dropdown list, click **Invoices** to open the **Invoices List**.

On the **Invoices List** page, use the **Search** functionality to find your record. The picture below points to the **Search by Business Status** option but you can use some other variables for your search, also. See the picture for details.

Invoice No.	Invoice Date	Contractor	Invoice Amount	Currency	Due Date	Business Status
REF0000532	13/09/2021		75.00	EUR	31/05/2022	Generated
REF0000530	13/09/2021		250.00	EUR	12/05/2022	Generated
REF0000531	13/09/2021		75.00	EUR	28/02/2022	CLOSED
REF0000529	13/09/2021		250.00	EUR	12/02/2022	Generated
REF0000400	10/08/2021		75.00	EUR	10/02/2022	CLOSED

3. Double click the desired record in order to open it and look for the **Edit** button, inside the form. When the selected record opens, you can see the header of the invoice containing the invoice details. The details from the header section are highlighted in gray - since this information is not editable.

**Invoice**

Invoice No.	<input type="text" value="REF0000563"/>		
Broker ID	<input type="text" value=""/>	Broker Name	<input type="text" value=""/>
Invoice Date	<input type="text" value="04/10/2021"/>	Invoice Reference	<input type="text" value="REF0000563"/>
Invoice Premium Amount	<input type="text" value="25"/>	Total Taxes	<input type="text" value="2.5"/>
Total Commission	<input type="text" value="2.5"/>	Invoice Amount	<input type="text" value="25"/>
Currency	<input type="text" value="EUR"/>	Payment Type	<input type="text" value="Broker Collection"/>
Due Date	<input type="text" value="14/10/2021"/>	Grace Limit Date	<input type="text" value="14/10/2021"/>
Paid Amount	<input type="text" value="0"/>	Unpaid Amount	<input type="text" value="25"/>
Scheduled date	<input type="text" value=""/>	<input type="button" value="Edit"/>	

4. At this step, you can also see below the other two sections of the invoice - **Installments** and **Premiums Collected**.

**INSTALLMENTS**

<input type="checkbox"/>	Policy No	Installment No	Due Date	Amount	Currency	Status	Remove
<input type="checkbox"/>	80000599	1	14/10/2021	25.00	EUR	Issued	<input type="button" value="Remove"/>

**PREMIUMS COLLECTED**

Payment ID	Payment Regis...	Payer Name	Amount	Currency	Allocated Amount	Currency	Business Status
No data							

5. Continue your journey by pressing **Edit**. For details on how to continue, see also the next section.

## 2. Edit and Save Updates

When you press **Edit**, the **Remove** buttons become available. Inside the **Installments** grid, select the desired installment and press **Remove**, next to it. Repeat this step as many times as necessary.

Paid Amount  Unpaid Amount   
 Scheduled date

**INSTALLMENTS**

<input type="checkbox"/>	Policy No	Installment No	Due Date	Amount	Currency	Status	Remove
<input type="checkbox"/>	<input type="text" value="80000599"/>	<input type="text" value="1"/>	<input type="text" value="14/10/2021"/>	<input type="text" value="25.00"/>	<input type="text" value="EUR"/>	<input type="text" value="Issued"/>	<input type="button" value="Remove"/>

The system keeps a history of your updates: all the removed installments remain visible inside the **Installments** grid, highlighted in gray (see the pic below). The information about removal cannot be edited.

**NOTE**  
 The status of the invoice, from which the installment was removed, changes from **Paid** to **Draft**.

**INSTALLMENTS**

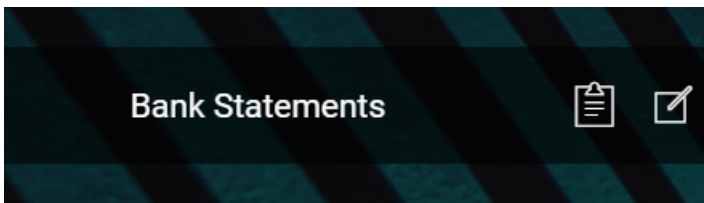
<input type="checkbox"/>	Policy No	Installment No	Due Date	Amount	Currency	Status	Remove
<input type="checkbox"/>	<input type="text" value="80000599"/>	<input type="text" value="1"/>	<input type="text" value="14/10/2021"/>	<input type="text" value="25.00"/>	<input type="text" value="EUR"/>	<input type="text" value="Cancelled"/>	<input type="button" value="Remove"/>

Once you finished removing the desired installments, save your updates by clicking **Save** - inside the form or by clicking **Save and close** - at the top right corner of your screen.

**HINT**  
 The **Billing and Collection** automatic invoicing covers multiple scenarios and can

scale from simple to complex ones. For more details about invoice settings at the product level, consult the [Insurance Product Factory](#) documentation and, also, consult the [Policy Admin](#) documentation, for setting scheduling payments.

## Bank Statements



The **Bank Statements** feature helps you add payment data into the system.

Uploading a bank statement file triggers the automatic parsing, sorting, and matching of the **Incoming Payments** data - contained by the file, with the **Invoices** already registered into your system. For payments that respect some rules - such as providing the correct unique identifier for an installment or the number of the policy, the [allocation flow is completely automated](#).

After import, you can find all the new payments into the general list, from the [Payments](#) section. You can recognize the automatically allocated payments by their **Closed** business status. If necessary, you can open a record to see the paid amount. More details about how the system does the sorting of the received payment data, in the [Incoming Payments](#) technical documentation.

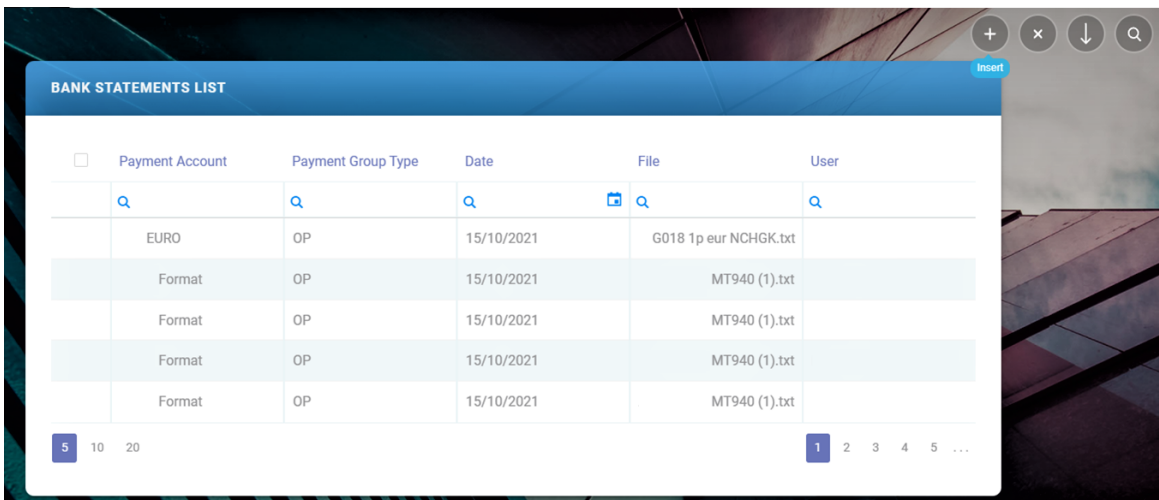
You also use this functionality to register **Outgoing Payments** in the system - namely, those requests for payments that your bank sends with the bank statement file. More details about how the system does the sorting of the received payment requests data, in the [Outgoing Payments](#) technical documentation. For some of your recurring outgoing payments scenarios - like paying the electricity bill, you can configure **Billing and Collection** to [automatically approve and allocate](#) those types of outgoing payments. For more details about managing Outgoing Payments with the **Billing and Collection** solution, consult the [Outgoing Payment Requests](#) page.

**IMPORTANT!**

You can delete **Bank Statements** that contain **Unallocated** payments only!

## Bank Statements View

In your portal, in the **Bank Statements** section, you have an overview of all your bank statements imported in the system - with the newest imported bank statement listed at the top.



<input type="checkbox"/>	Payment Account	Payment Group Type	Date	File	User
	EURO	OP	15/10/2021	G018 1p eur NCHGK.txt	
	Format	OP	15/10/2021	MT940 (1).txt	
	Format	OP	15/10/2021	MT940 (1).txt	
	Format	OP	15/10/2021	MT940 (1).txt	
	Format	OP	15/10/2021	MT940 (1).txt	

This general list also gives you the possibility to search and sort the imported bank statements for easier processing. For example if you want to view all the imported bank statements for a certain bank account of yours, you can use the **Search by Payment Account** option and sort all your records accordingly.

Follow the steps below to view your bank statements:

1. In your **FintechOS Portal**, navigate down the main menu of the **Billing and Collection** solution.
2. From the dropdown list, click **Bank Statements** to open the **Bank Statements List**.

On the **Bank Statements List** page:

- To inspect a record from the grid, double-click it.
- To add a new record, click **Insert**, at the top right corner of the page.
- To delete a record from the grid, select it and click **Delete**, at the top right corner of the page. Clicking **Delete** triggers the deletion of a bank statement and all the unallocated payments included in it. However, there is a specific validation for this user action: in order to delete it, a bank statement must contain only unallocated payments, otherwise if it contains allocated payments, the following error message is displayed: “This Bank Statement contains allocated payments and cannot be deleted!”.

**HINT**

You can export one or more records by pressing **Export**, at the top right corner of your screen.

## Uploading Bank Statements

**IMPORTANT!**

To import payments, you need to use the standard bank statement file issued by your bank, for your bank accounts. For example some banks use the **MT940** format file for issuing statements. Also, you use this same functionality to import standardized files from other payment processors - such as **PayU**. The **Billing and Collection** module can parse through different standardized payment data files.

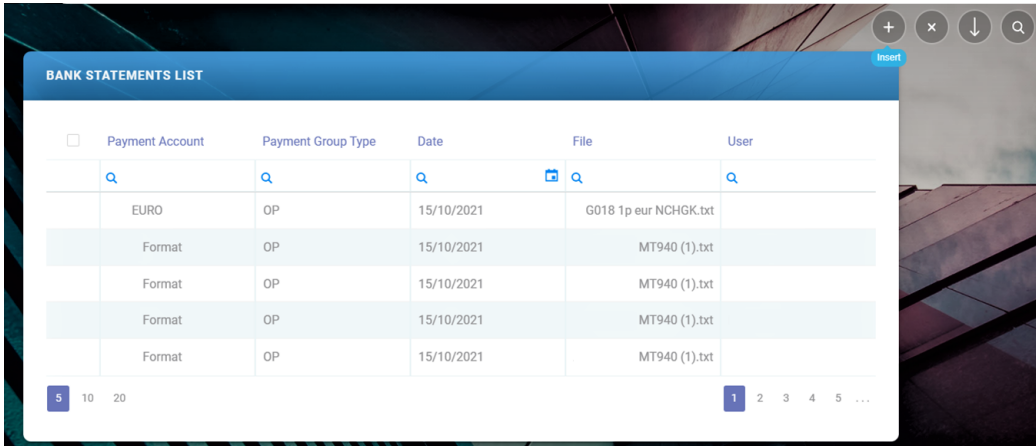
Follow the steps below to add payment data into your system:

### 1. Open the Billing and Collection Module

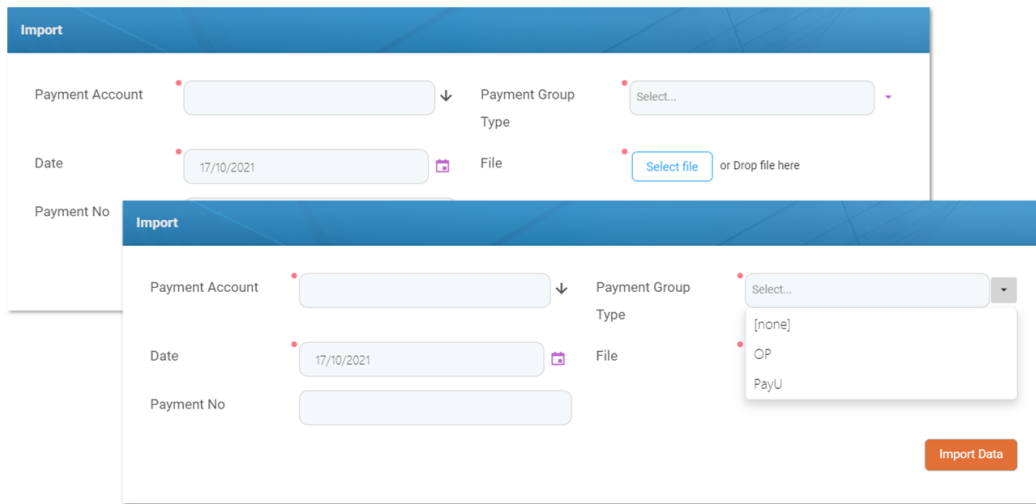
Once you obtained the standardized payments file from your bank...

In your **FintechOS Portal**, navigate to the **Bank Statements List** page.

On the **Bank Statements List** page, at the top right corner, click **Insert** to import the new file.



## 2. Import the Bank Statement File



Inside the **Import** pop-up, fill in the following fields:

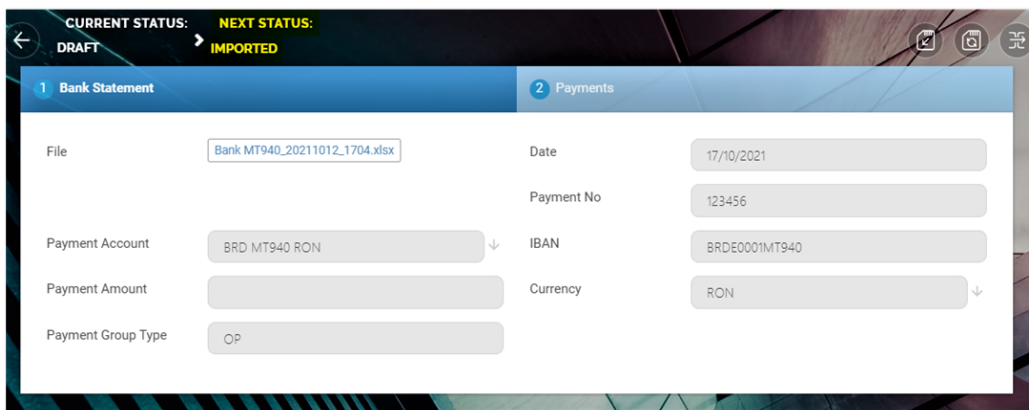
Field	Description
Payment Account	From the dropdown, select the bank account <b>standardized file type</b> and click <b>Ok</b> . The current option set values are: <b>BRD MT940 EUR, BRD MT940 RON, ING EURO, and ING RON</b> .
Payment Group Type	From the option set, select the group type for processing payments. You can choose between <b>OP</b> (bank payment) or <b>PayU</b> (for online payment processors).
Date	The date of the import - is automatically filled-in with the current date.

Field	Description
File	You can either <b>Add</b> or <b>Drop</b> the file containing the payment data.

Click **Import Data**. Click **Save and reload**, at the top right of your screen.

### 3. Inspect and Validate Payment Data

After clicking **Import Data**, the payment data view appears. Notice that the current status of your record is **Draft** and the next status is **Imported**. Proceed to check the details of your import.



#### Bank Statement Tab

This first tab is automatically populated with details, either introduced in the previous step or extracted from the file - such as the IBAN. You don't need to fill in any details here.

Go to the **Payments** tab.

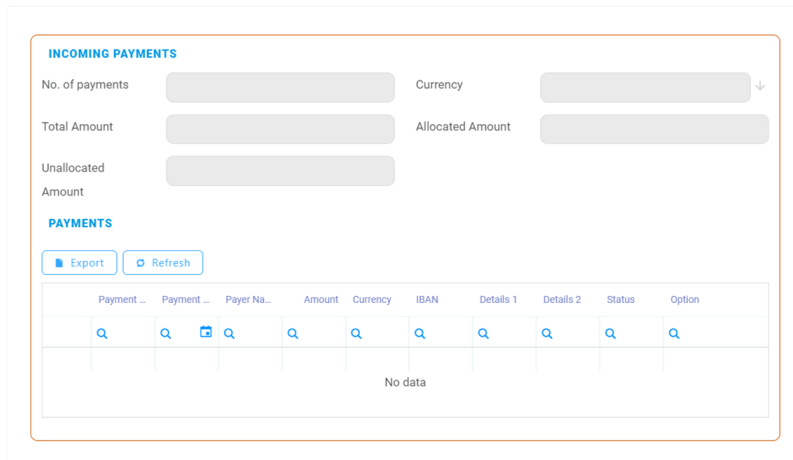
#### Payments Tab

This tab displays all the payments from the imported file. The payment data is organized as follows:

##### Incoming Payments

The header section displays the **Incoming Payments**:





Inside this section, the following fields are automatically populated with details extracted from the imported file. You don't need to fill in any details here.

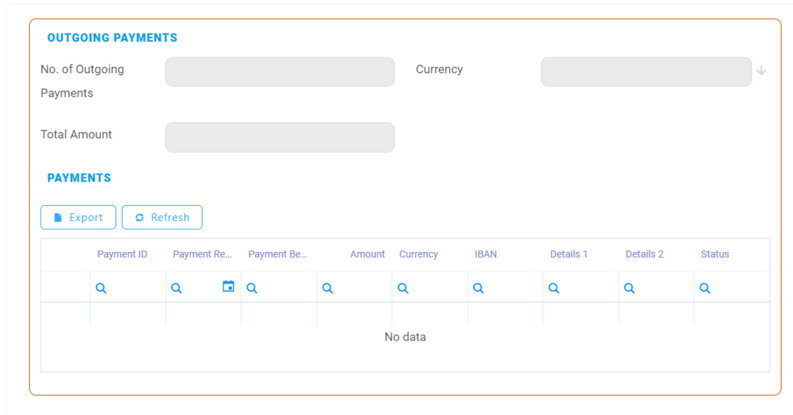
Field	Description
No. of Payments	The number of incoming payments.
Currency	The currency of the incoming payments.
Total Amount	The total incoming payment amount per that bank statement.
Allocated Amount	The total amount that the system managed to allocate automatically.
Unallocated Amount	The total amount that the system failed to allocate automatically.

At the bottom of the section, you can see the **Payments** grid - containing every incoming payment extracted from the imported file. For the case when there are many incoming payments, you can use the search functionality, inside this grid, to find a certain payment.

Next to each unallocated incoming payment there is a **Payment Return** button. For details about returning unallocated payments, consult the [Returning Payments](#) section of this guide.

# Outgoing Payments

The bottom section displays the **Outgoing Payments**:



Inside this section, the following fields are automatically populated with details extracted from the imported file. You don't need to fill in any details here.

Field	Description
No. of Outgoing Payments	The number of outgoing payments.
Currency	The currency of the outgoing payments.
Total Amount	The total outgoing payment amount per that bank statement.

At the bottom of the section, you can see the **Payments** grid - containing every outgoing payment extracted from the imported file. For the case when there are many outgoing payments, you can use the search functionality, inside this grid, to find a certain outgoing payment.

You notice that next to each unallocated incoming payment there is a Payment Return button. For details about returning unallocated payments, consult the Outgoing Payment Requests section of this guide.

# Update Business Status

## IMPORTANT!

In order for the payment data to be made available for processing in the system, be further available for manual allocation flows, payment return or reconciliation processes or any other payment operations inside the system, **you must validate it** by manually changing the business status from **Draft** to **Imported**.

The screenshot shows a software interface for entering bank statement data. At the top, it indicates the 'CURRENT STATUS' is 'DRAFT' and the 'NEXT STATUS' is 'IMPORTED'. The form is divided into two sections: '1 Bank Statement' and '2 Payments'. The 'Bank Statement' section includes fields for 'File' (Bank MT940\_20211012\_1704.xlsx), 'Payment Account' (BRD MT940 RON), 'Payment Amount', and 'Payment Group Type' (OP). The 'Payments' section includes fields for 'Date' (17/10/2021), 'Payment No' (123456), 'IBAN' (BRDE0001MT940), and 'Currency' (RON).

Once you finished checking the payment data, proceed to validate it. Click the **Next Status** option set, at the top left corner of the screen. A pop-up appears and you are asked: "Are you sure that you want to change the business status?" Click **Yes**.

After clicking **Yes**, you notice that the business status of the record is changed to **Imported**.

This screenshot shows the same 'Bank Statement' form as above, but the 'CURRENT STATUS' at the top left is now 'IMPORTED'. The form fields and their values remain the same, indicating that the data has been successfully updated to the 'Imported' status.

After the change of the record business status, the imported payment data is made available in the system - to be used by other processes.

**HINT**

For more information about the import rules and configurations, consult also the [Import Bank Statements](#) technical page.

## Payments

**Billing and Collection** allows you to add payments into the system by importing files received from the payment processors. The solution can be used in conjunction with different types of incoming or outgoing payments - such as bank payment orders, direct debit payments, credit card payments, or payments made through online processors. The bulk of payment processing is done automatically when you upload the statement files. For more details about how **Billing and Collection** handles payments, see the [Incoming Payments](#) or [Outgoing Payments](#) pages and also the [Payment Lifecycle](#) page.

## Payments View

In your portal, in the **Payments** section, you can find the repository of all the payments managed by the **Billing and Collection** solution.

The **Payments** section offers you an overview of all your incoming payments, in their current business statuses **Unallocated**, **Partially Allocated** and **Closed** (for fully allocated payments). This view is automatically and continuously updated with new data, as payments are regularly registered and processed by the system.

Double clicking on a payment record that is in **Unallocated** or **Partially Allocated** business status, opens the **Unallocated Payments Form** that lets you manually allocate, deallocate or return that payment. For example for an incoming payment

you can allocate the sum on a specific installment (partially or entirely) and for an outgoing payment you can approve the proposed payment request (depending on security roles).

For step by step instructions, consult also the [Unallocated Payments](#) or [Outgoing Payment Requests](#) pages.

PAYMENTS								
<input type="checkbox"/>	Payment ID	Payment Category	Payment Type	Payer Name	Payment Re...	Curre...	Amount	Business St...
	Q	Q	Q	Q	Q	Q	Q	Q
	1	Outgoing Payme...	Bank Charges		26/07/2021	EUR	2,000.00	Unallocated
	1	Incoming Payme...	Payment Order		25/04/2018	EUR	50,000.00	Unallocated
	1	Incoming Payme...	Payment Order		25/04/2018	EUR	50,000.00	Closed
	1	Incoming Payme...	Payment Order		25/04/2018	EUR	50,000.00	Closed
	1	Incoming Payme...	Payment Order		25/04/2018	EUR	50,000.00	Closed

5 10 20 1 2 3 4 5 ...

This is an all-inclusive view; yet, you also can search and sort your payments for easier processing. For example if you want to view all the **Closed** payments - that is payments that went through all the payment processing steps and reached their final business status, you can use the **Search by Business Status** option and sort all your payment data accordingly.

Follow the steps below to view your payments:

1. At the top left corner of your **FintechOS Portal**, click the main menu icon to open the main dropdown list.
2. From the main list, click **Billing and Collection**. A second dropdown opens.
3. Next, click **Payments** to go to the **Payments List**. This is the central repository for all payments existing in your system.

On the **Payments List** page:

- To inspect a record from the grid, double-click it.
- To add a new record, click **Insert**, at the top right corner of the page.

- To delete a record from the grid, select it and click **Delete**, at the top right corner of the page.

**HINT**

You can export one or more records by pressing **Export**, at the top right corner of your screen.

## Insert Payment Form

Even if rare, there are cases when you need to manually register a payment made by a customer. For this, you use the **Insert Payment Form** to add a new payment record, in the system.

**NOTE**

You can access this form from the **Payments** as well as the **Unallocated Payments** menus.

Follow the steps below to register a payment:

1. At the top left corner of your **FintechOS Portal**, click the main menu icon to open the main dropdown list.
2. From the main list, click **Billing and Collection**. A second dropdown opens.
3. Next, click **Payments** to go to the **Payments List**.
4. On the **Payments List** page, click **Insert**, at the top right corner of the page, to open the **Insert Payment Form**.

Below is an example of an insert payment Form:

Insert Payment

**PAYMENT**

Payment Type  Currency

Payment ID  Amount

Payment Registration Date

Payer Name  Comments

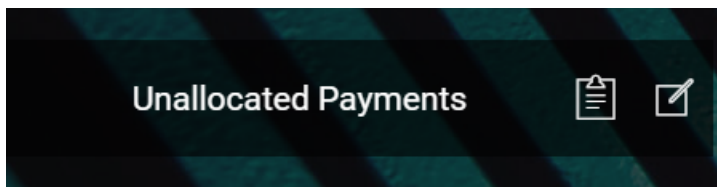
Broker

Proceed to insert the necessary details into the form, as follows:

Field Name	Description
Payment Type	Use the option set to select a type for the payment. The types are as follows: <ul style="list-style-type: none"> <li>• paymentOrder,</li> <li>• brokerPremiumPayment,</li> <li>• paymentUnallocated,</li> <li>• bankCharges,</li> <li>• outgoingPayment.</li> </ul>
Payment ID	Give a reference, or ID to the payment you register.
Payment Registration Date	Use the date picker to choose the date for your payment registration.
Payer Name	The payer name.
Broker	The broker name, or reference
Currency	The currency of the payment.
Amount	The amount that was paid.
Comments	Text area for comments.

Click **Save and close** to close the form **OR** click **Save and reload**. If the later, the **Unallocated Payments Form** opens and you can continue with the allocation flow - to allocate your freshly registered payment to an installment already existing in the system. For more details about the manual allocation process, consult the [Unallocated Payments](#) page.

## Unallocated Payments



**Unallocated Payments** is the section where you can see all incoming payments that the system was unable to allocate automatically - for example a policyholder made a payment without indicating the installment number or the policy number and the system does not have the necessary details in order to link (allocate) that paid amount to any registered installment.

This is also the place where a payment ends up when you don't allocate the payment's full amount on an invoice. You can either use the **Allocate** or the **Return** manual flow, in order to deal with a payment in **Unallocated** or **Partially Allocated** business status. More details to follow, below.

Even if rare, there are cases when you need to manually register a payment made by a customer. For this, you use the **Insert** button to add a new payment record, in the system. Once the payment registration is finished, upon hitting the **Save and reload** button, you are able to continue with the allocation flow. For more details about inserting a payment, go to the **Payments** page and scroll down to the [Insert Payment Form](#) section.

### **IMPORTANT!**

The details of an incoming payment record **are not editable**. You can only (partially or entirely) allocate, deallocate or return the payment. When the payment is fully allocated, the record changes its business status to **Closed**.



## Unallocated Payments View

In your portal, in the **Unallocated Payments** section, you have an overview of the unallocated payments registered into your system - with the newest unallocated payment at the top. In this list, the **Return** button next to every record offers a rapid way to initiate a payment return flow, for the selected payment. More details, in the Outgoing Payment Requests section.

UNALLOCATED PAYMENTS								
<input type="checkbox"/>	Payment ID	Payer Name	Payment Registrati...	Payment Type	Curre...	Amount	Business Status	Option
	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
	3		25/04/2018	Payment Order	EUR	907.50	Partially Allocated	<input type="button" value="Return"/>
	3		25/04/2018	Payment Order	EUR	907.50	Unallocated	<input type="button" value="Return"/>
	1		14/07/2021	Payment Order	RON	0.00	Unallocated	<input type="button" value="Return"/>
	1		15/04/2021	Payment Order	RON	1,000.00	Unallocated	<input type="button" value="Return"/>
	1		23/07/2021	Payment Order	EUR	95.00	Unallocated	<input type="button" value="Return"/>

5 10 20 1 2 3 4 5 ...

This is an all-inclusive view; yet, you also can search and sort your payments for easier processing. For example if you want to view all the payments in **Partially Allocated** status, you can use the **Search by Business Status** option and sort all your payments accordingly.

Follow the steps below to view your unallocated or partially allocated payments:

1. In your **FintechOS Portal**, navigate down the main menu of the **Billing and Collection** solution.
2. From the dropdown list, click **Unallocated Payments** to go to the **Unallocated Payments List**.

On the **Unallocated Payments List** page:

- To inspect a record from the grid, double-click it.
- To insert a record, click Insert, at the top right corner of your screen. Go to the [Insert Payment Form](#) section, for more details.
- To delete a record from the grid, select it and click **Delete**, at the top right corner of the page.

#### **HINT**

You can export one or more records by pressing **Export**, at the top right corner of your screen.

## Unallocated Payments Form

When you double-click a record from the **Unallocated Payments** section, in order to manage a payment, you launch the **Unallocated Payments Form**. This form allows you to partially or entirely allocate the selected payment, or to propose its return. It also allows you to deallocate amounts previously allocated - for example for the case when you allocated an amount on an invoice in error.

Manual allocation can be done either [backward-looking](#) - when you allocate a payment on an invoice (already existing in your system) or [forward-looking](#) - when you allocate a payment on a future installment (triggering the automatic issuing of the invoice that matches with the payment).

The **Unallocated Payments Form** contains the following sections:

### The Client Payment Section

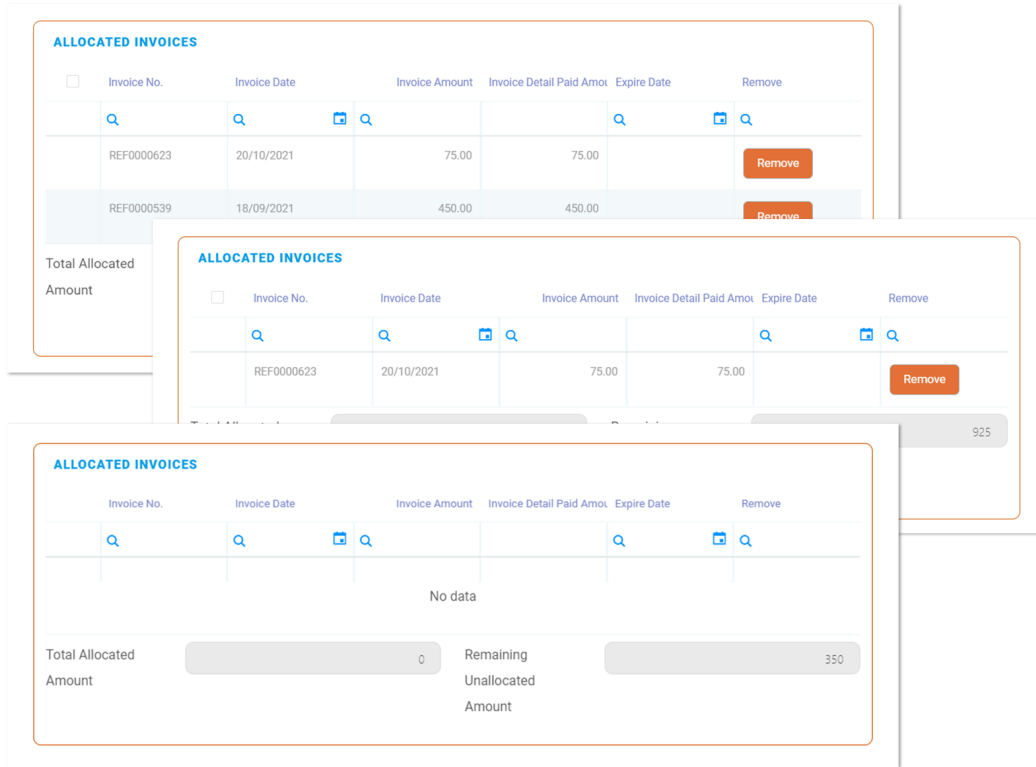
This is the header section. You use this section to see the **Client Payment** details, which are not editable, and to decide which manual flow you are going to launch - either the **Allocate** or the **Return** flow. You also use this section to check the status of the record, at the top left of your screen. Below, you can see the header section of a record in **Unallocated** business status.

## The Allocated Invoices Section

This is the middle section. Use this section to:

- see the **Invoices** that are currently **Allocated** on the selected payment. Depending on the situation, you can see none, one, or more invoices allocated on the selected payment.
- check the status of the **Total Allocated Amount** and the **Remaining Unallocated Amount** since their sums update dynamically as a consequence of your using the form to allocate or deallocate amounts from the payment.
- decide if you keep or **Remove** some of the allocated invoices - since there may be times when you need to use that **Remove** button; for example if you've allocated a payment to an invoice by mistake.

Below, you can see the middle section of three different records.



## The Outgoing Payments Section

This is the bottom section. You use this section to see any amount, from the selected payment, that was submitted to the outgoing payments flow when you pressed the **Return** button, in the header section. Depending on the situation, you can see none (when you don't press **Return**) or one draft outgoing payment request, on the selected payment. If needed, from here, you can jump directly into the manual Outgoing Payment Request flow, by double-clicking on the **Draft** request, inside the grid.

Below, you can see the bottom section of two different records.

The image shows two overlapping screenshots of a web application interface for 'OUTGOING PAYMENTS'. The top screenshot displays an empty table with the following columns: Payment No, Outgoing Payment T..., Payment Amount, Currency, Created On, and Business Status. Each column has a magnifying glass icon for search. Below the table, it says 'No data'. The bottom screenshot shows the same table with one row of data:

<input type="checkbox"/>	Payment No	Outgoing Payment T...	Payment Amount	Currency	Created On	Business Status
	REF0000571	Unallocated Payment	25.00	EUR	20/10/2021	Draft

## Allocating Payments

Take the below steps for allocating payments:

### 1. Launch the Unallocated Payments Form

In order to deal with a payment in **Unallocated** or **Partially Allocated** business status, you must go to the **Unallocated Payments List** page and double-click the desired record to open it.

UNALLOCATED PAYMENTS								
<input type="checkbox"/>	Payment ID	Payer Name	Payment Registrati...	Payment Type	Curre...	Amount	Business Status	Option
	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
	3		25/04/2018	Payment Order	EUR	907.50	Partially Allocated	<input type="button" value="Return"/>
	3		25/04/2018	Payment Order	EUR	907.50	Unallocated	<input type="button" value="Return"/>
	1		14/07/2021	Payment Order	RON	0.00	Unallocated	<input type="button" value="Return"/>
	1		15/04/2021	Payment Order	RON	1,000.00	Unallocated	<input type="button" value="Return"/>
	1		23/07/2021	Payment Order	EUR	95.00	Unallocated	<input type="button" value="Return"/>

5 10 20 1 2 3 4 5 ...

This launches the **Unallocated Payments Form**.

Check the status of your record, at the top left of your screen.

Inside the **Unallocated Payments Form**, find the **Allocate** button and click it.

## 2. Search for Unpaid Invoices

Clicking **Allocate** activates the **Search** grid. Scroll down to find this grid, below the **Outgoing Payments Section**. Use the **Search** functionality to sort through possible results. You might want to search by **First Name**, or by **Date**, or by **Amount**, etc. Once finished, press **Search Invoice**.

**INVOICE SEARCH**

Start Due Date From  Start Due Date To

Last Name  Amount

First Name  Payment Type

Policy No.  Invoices No.

Depending on your search terms, you can see none, one or more **Invoices** displayed by the system. Under the **Invoices** list, a list of **Installments** (scheduled at future dates) is also displayed - for the case when you want to allocate the selected payment to an **Installment** (partially or entirely).

Below, you can see results from two different searches.

**INVOICES LIST**

Invoice No	Invoice Date	Contractor	Invoice Amou...	Unpaid Amou...	Currency	Invoice Refer...	Due Date	Option
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
REF0000564	04/10/2021		60.47	40.57	EUR	REF0000564	04/10/2021	<input type="button" value="Add"/>
REF0000498	30/01/...							
REF0000399	10/01/...							
REF0000398	10/01/...							
REF0000392	06/01/...							

**INVOICES LIST**

Invoice No	Invoice Date	Contractor	Invoice Amou...	Unpaid Amou...	Currency	Invoice Refer...	Due Date	Option
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
REF0000564	04/10/2021		60.47	40.57	EUR	REF0000564	04/10/2021	<input type="button" value="Add"/>

**INSTALLMENTS LIST**

Policy No.	Installment No.	Due Date	Amount	Currency	Allocation Amount	Option
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
BRDAG EMB80000...	2	11/10/2021	200	EUR		<input type="button" value="Add"/>
80000200	2	16/10/2021	75	EUR		<input type="button" value="Add"/>
BRDAG EMB80000...	2	17/10/2021	200	EUR		<input type="button" value="Add"/>
BRDAG EMB80000...	2	17/10/2021	200	EUR		<input type="button" value="Add"/>

The **Invoices** and the **Installments** lists have their own **Search** functionality to help you sort through the data.

However, there are situations when you cannot work with the current results and you need to refine your main search. If so, go back to the **Search** grid and introduce some other search terms. This time, you might want to try other search options like **Payment Type** or to set other **Dates** - for example so that the system displays fewer results. You can repeat the search step until you are satisfied with the results. When so, continue to the next step.

### 3. Click Add to Allocate

In the search results list, next to every **Invoice** or **Installment** you can see the **Add** button which offers a rapid way to (partially or entirely) match invoices or installments with the current unallocated payment. Click **Add** to allocate the payment amount on the desired invoice or installment.

Below is an example of some allocation options for a payment amount.

INVOICES LIST									
Invoice No	Invoice Date	Contractor	Invoice Amou...	Unpaid Amou...	Currency	Invoice Refer...	Due Date	Option	
REF0000564	04/10/2021		60.47	40.57	EUR	REF0000564	04/10/2021		<b>Add</b>

INSTALLMENTS LIST									
Policy No.	Installment No.	Due Date	Amount	Currency	Allocation Amount	Option			
BRDAG EMB80000...	2	11/10/2021	200	EUR					<b>Add</b>
80000200	2	16/10/2021	75	EUR					<b>Add</b>
BRDAG EMB80000...	2	17/10/2021	200	EUR					<b>Add</b>
BRDAG EMB80000...	2	17/10/2021	200	EUR					<b>Add</b>

When you click **Add**, next to an item from the list, the selected invoice or installment shows up in the upper **Allocated Invoices** grid. You can scroll up and check the status of the **Total Allocated Amount** and the **Remaining Unallocated Amount** of the current payment inside this grid.

Below is an example of many allocations inside the **Allocated Invoices** grid.



ALLOCATED INVOICES						
<input type="checkbox"/>	Invoice No.	Invoice Date	Invoice Amount	Invoice Detail Paid Amou	Expire Date	Remove
<input type="checkbox"/>	REF0000464	18/08/2021	75.00	75.00		<input type="button" value="Remove"/>
<input type="checkbox"/>	REF0000411	12/08/2021	250.00	250.00		<input type="button" value="Remove"/>
<input type="checkbox"/>	REF0000503	31/08/2021	135.79	135.79		<input type="button" value="Remove"/>
<input type="checkbox"/>	REF0000478	18/08/2021	75.00	75.00		<input type="button" value="Remove"/>
<input type="checkbox"/>	REF0000626	20/10/2021	41.25	41.25		<input type="button" value="Remove"/>

5 10 20

Total Allocated Amount 2,657.04

1 2 3 4

Remaining Unallocated Amount 47,342.96

There are cases when you can't allocate the entire amount with only one **Add**. For example some policyholder made a payment that covers multiple invoices and you need to allocate different amounts on those different invoices, or maybe that payment covers also a future installment. So, if the payment is only partially allocated and you need to further allocate the rest of the amount, you can repeat this step as many times as necessary.

**IMPORTANT!** If you need to refine you search again, you must scroll-up and press the **Validate** button first, to save your allocations and then, press **Allocate**, again, to launch a new **Search**.

Click **Add** to allocate until you finished making your allocations. Once finished, scroll up.

## 4. Validate Your Allocations

In the header section, click **Validate**. When you click **Validate**, you can see, at the top left of your screen, the that the business status of the current record changed from **Unallocated** to **Closed**, if you allocated the entire payment amount.

Below is an example of a record in **Closed** status.

The screenshot shows a 'Client Payment' form with the following fields and values:

Payment ID	2	Payer Name	Crimtech Srl
Payment Registration Date	01/09/2020	Bank Reference	ING
Payment Date	08/07/2021	Business Status	Closed
Amount	1,000	Currency	EUR
Comments			

At the bottom of the form, there are three buttons: **Return**, **Allocate**, and **Validate**. The 'CURRENT STATUS' is indicated as **CLOSED** in the top left corner.

If, by chance, you did not allocate all the amount, you can validate your current allocations and leave the rest of the amount to be sent back to the payer. This is done by pressing the **Return** button to initiate the return flow yourself. For more details, consult the page about [Returning Payments](#).

Below is an example of a record in **Partially Allocated** status.

The screenshot shows a 'Client Payment' form with the following fields and values:

Payment ID	2	Payer Name	Crimtech Srl
Payment Registration Date	01/09/2020	Bank Reference	ING
Payment Date	08/07/2021	Business Status	Partially Allocated
Amount	1,000	Currency	EUR
Comments			

At the bottom of the form, there are three buttons: **Return**, **Allocate**, and **Validate**. The 'CURRENT STATUS' is indicated as **PARTIALLY ALLOCATED** in the top left corner.

If, by chance, while making your final check on the payment, you spot an allocation error - for example an amount allocated to the wrong invoice or installment, you can deallocate that amount by using the **Remove** button.

ALLOCATED INVOICES						
<input type="checkbox"/>	Invoice No.	Invoice Date	Invoice Amount	Invoice Detail Paid Amou	Expire Date	Remove
<input type="checkbox"/>	REF0000464	18/08/2021	75.00	75.00		<input type="checkbox"/> Remove
<input type="checkbox"/>	REF0000411	12/08/2021	250.00	250.00		<input type="checkbox"/> Remove
<input type="checkbox"/>	REF0000503	31/08/2021	135.79	135.79		<input type="checkbox"/> Remove
<input type="checkbox"/>	REF0000478	18/08/2021	75.00	75.00		<input type="checkbox"/> Remove
<input type="checkbox"/>	REF0000626	20/10/2021	41.25	41.25		<input type="checkbox"/> Remove

5 10 20 1 2 3 4

Total Allocated Amount	2,657.04	Remaining Unallocated Amount	47,342.96
------------------------	----------	------------------------------	-----------

Remember that, after using the **Remove** button, the **Total Allocated Amount** and the **Remaining Unallocated Amount** update their statuses dynamically, as a consequence of your using the form to allocate or deallocate amounts from the payment. For more details, consult the page about [Deallocating Payments](#).

Once finished, press **Validate** and then press **Save&Close**, at the top right corner of your screen, to close the form.

## Deallocating Payments

There are cases when you need to deallocate payments from invoices. For example when an allocation error happens or a customer asks for a refund on a payment made for a future installment, and so on.

**IMPORTANT!** The system does not allow you to deallocate payments from records in **Closed** status. You can only make these kind of changes on records in **Unallocated** or **Partially Allocated** status.

By using the **Remove** button, you can easily deallocate any amount when necessary. For doing so, take the below steps:

1. At the top left corner of your **FintechOS Portal**, click the main menu icon to open the main dropdown list.
2. From the main list, click **Billing and Collection**. A second dropdown opens.
3. Next, click **Payments** to go to the **Payments List**.
4. Inside the **Payments List** page, use the search functionality to find the desired payment record.
5. Double click the record to open it and go to the **Allocated Invoices** grid.

Below is an example of multiple allocated invoices listed inside the **Allocated Invoices** grid.

<input type="checkbox"/>	Invoice No.	Invoice Date	Invoice Amount	Invoice Detail Paid Amou	Expire Date	Remove
<input type="checkbox"/>	REF0000464	18/08/2021	75.00	75.00		Remove
<input type="checkbox"/>	REF0000411	12/08/2021	250.00	250.00		Remove
<input type="checkbox"/>	REF0000503	31/08/2021	135.79	135.79		Remove
<input type="checkbox"/>	REF0000478	18/08/2021	75.00	75.00		Remove
<input type="checkbox"/>	REF0000626	20/10/2021	41.25	41.25		Remove

5 10 20

1 2 3 4

Total Allocated Amount: 2,657.04

Remaining Unallocated Amount: 47,342.96

6. Inside the grid, locate the invoice that you want to deallocate and click the **Remove** button next to it.
7. Repeat the previous step as many times as necessary.
8. Click **Validate** to save your updates on the current payment record.

After using the **Remove** button, the **Total Allocated Amount** and the **Remaining Unallocated Amount** update their statuses dynamically, as a consequence of your using the form to allocate or deallocate amounts from the payment.

**NOTE**

The status of the payment changes from **Partially Allocated** to **Unallocated** if you remove all the items inside the **Allocated Invoices** grid.

## Outgoing Payments Operations

The Billing and Collection solution helps you manage the payment requests received from different sources - such as when you **import bank statement files**, when you return an unallocated payment or **manually introduce a request for payment** in the system.

Check the following pages to see how this functionality works:

[Outgoing Payment Requests](#) - for details about the outgoing payment requests **View** and **Form**, and for the payment requests approval flow.

[Adding Payment Requests](#) - for details about how to register your payment request.

[Returning Payments](#) - for details about the payment return flow.

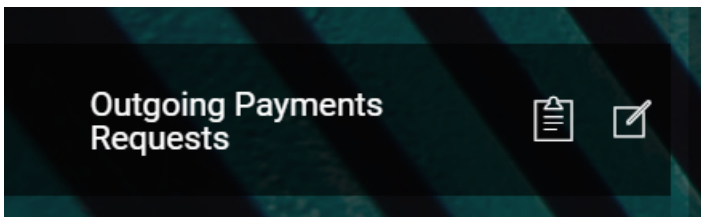
[Outgoing Payments Instruction Files](#) - for details about the instruction files generated by the system, and used to allocate outgoing payments.

For more technical details, see also the [Generate Outgoing Payments API](#), and [Outgoing Payments](#) pages.

**HINT**

You can easily find your **Draft** payment request in the [Outgoing Payment Requests List](#) page by using the **Search by Status** option.

## Outgoing Payment Requests



The **Outgoing Payment Requests** is the section where you find all the outgoing payments registered into your system.

For some of your recurring outgoing payments scenarios, you can configure **Billing and Collection** to [automatically approve and allocate](#) those types of outgoing payments. For other scenarios, you have an outgoing payment request [manual approval flow](#) that lets you approve or decline such requests, according to your needs. For more details about the functionality used to allocate outgoing payments, see also the [Outgoing Payments Instruction Files](#) page.

With **Billing and Collection** you can effortlessly manage payment requests received from different sources or systems - such as Claim payments, Commission, Broker Balance (Credit), etc. See the examples below:

Payment requests are automatically registered into your system when you use the **Bank Statements** functionality to upload your bank statement files. Your import triggers the automatic parsing, sorting and matching of the **Payment** data - contained by the file, with the **Outgoing Payments** already configured or scheduled by you - such as broker commissions. For payment requests that respect some rules - such as providing the correct unique identifier for a broker or the number of the policy under brokerage, the [allocation flow for the outgoing payment is completely automated](#). After import, you can find all the new outgoing payments into the **Outgoing Payment Requests** section. You can recognize the automatically allocated outgoing payments by their **Paid** business status. If necessary, you can open a record to see the paid amount. More details about how the system does the sorting of the received payment data, in the [Import Bank Statements](#) documentation.

Another example is the receiving of payments request through the [Generate Outgoing Payments API](#). For example, an insurer uses an external Claims system that calls the Generate Outgoing Payments API to register a new request for a claim payment. Once received, the payment request is processed by the **Billing and Collection** module and

automatically registered into the system, in **Draft** status. Further, you use the [manual approval flow](#) to approve, or decline the request, accordingly. If the payment request received through the API is already in **Approved** business status, the system completes the payment flow, automatically.

Yet another example is when you decide the return of an unallocated incoming payment. You can return the whole amount or the amount remained after the allocation operation. The return of **Unallocated Payments** can be done [automatically](#), after a configurable number of days. However, for the case of a **Partially Allocated** payment, you need to initiate the return flow by yourself. For more details, see the [Returning Payments](#) section. When you press the **Return** button, you create a **Draft** payment request that is automatically populated with the necessary data and transitioned by the system into the **Outgoing Payment Requests** section.

Finally, this is also the place where you go in order to manually insert a request for an outgoing payment into the system, when the case.

## Outgoing Payment Requests View

In your portal, the **Outgoing Payment Requests** section is continuously updated, with payment request data coming from different sources, as shown in the examples above. This section offers you an overview of the payment requests registered into your system, with the newest record at the top.

OUTGOING PAYMENTS REQUESTS								
<input type="checkbox"/>	Payment No	Reference No	Payment type	Amount	Currency	Payment Be...	Paid Amount	Status
	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
	REF0000575	REF0000575	Unallocated...	294.24	EUR		0.00	Draft
	REF0000573	REF0000573	Unallocated...	75.00	EUR		0.00	Draft
	REF0000572	REF0000572	Unallocated...	70.00	RON		0.00	Draft
	REF0000571	REF0000571	Unallocated...	25.00	EUR		0.00	Approved
	REF0000570	REF0000570	Unallocated...	50,000.00	EUR		0.00	Cancelled

5 10 20 1 2 3 4 5 ...

This is an all-inclusive view; yet, you also can search and sort your records for easier processing. For example if you want to view all the payment requests in **Draft** status, you can use the **Search by Status** option and sort all your records accordingly.

Follow the steps to view your outgoing payments:

1. In your **FintechOS Portal**, navigate down the main menu of the **Billing and Collection** solution.
2. From the dropdown list, click **Outgoing Payment Requests** to open the **Outgoing Payment Requests** list, at the left.

On the **Outgoing Payment Requests** list page:

- To inspect a record from the grid, double-click it. Depending on security roles, you can **Approve** or **Cancel** a payment request.
- To add a record, at the top right corner, click **Insert**. For details, see the [Adding Payment Requests](#) page.
- To delete a record from the list, select it and click **Delete**, at the top right corner of the page.

#### HINT

You can export one or more records by pressing **Export**, at the top right corner of your screen.

## Outgoing Payment Requests Form

When you double-click a record, in order to manage an outgoing payment, you launch the **Outgoing Payment Request Form**. This form allows you to propose, approve or, if the case, cancel a payment request.

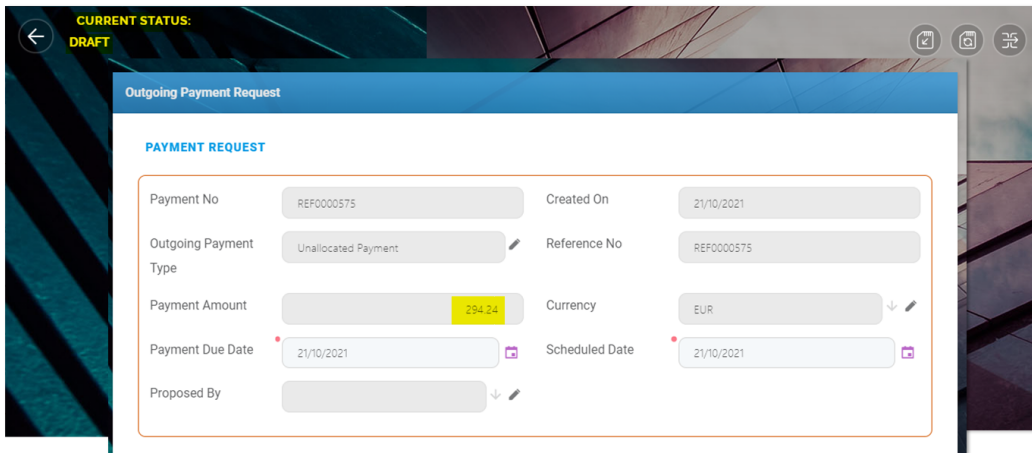
The **Outgoing Payment Request Form** contains the following sections:

## The Payment Request Section

This is the header section. You use this section to insert or to inspect the **Outgoing Payment** details. Depending on the case, the fields are editable or not. For example for [returning a payment](#), the details in this section are already filled in by the system and you only need to use the option sets to select the **Payment Due Date** and the **Scheduled Date** for payment. You also use this section to check the status of the record, at the top left of your screen.

Below, you can see the header section of a record in **Draft** business status.

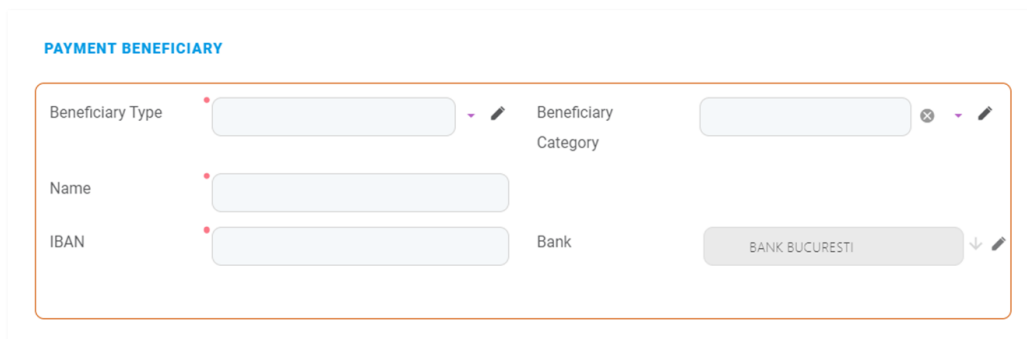




## The Payment Beneficiary Section

This is the middle section, which is editable. You use this section to fill in details about the **Payment Beneficiary** for the selected outgoing payment. The option sets make it easier for you to complete the **Beneficiary Type** and the **Beneficiary Category** fields. For more details about the option set values, see the table below. Depending on the type of outgoing payment, it is possible to see the **Bank** field already completed and displayed in a non-editable form. For example this is the case of a return payment - where the system already knows which bank sent the payment.

Below, you can see the middle section of an outgoing payment record, with **Bank** details already filled in.



Inside the **Payment Beneficiary** section, the following fields are editable:

Field Name	Description
Beneficiary Type	<p>The payment beneficiary type. The types are as follows:</p> <ul style="list-style-type: none"> <li>• Payer</li> <li>• Insured</li> <li>• Contractor</li> <li>• Policy Beneficiary</li> <li>• Broker</li> <li>• Service Provider</li> <li>• Other</li> </ul>
Beneficiary Category	<p>The beneficiary category. The category types are as follows:</p> <ul style="list-style-type: none"> <li>• Other</li> <li>• Individual Person</li> <li>• Legal Person</li> </ul>
Name	The name of the payment beneficiary.
IBAN	The IBAN of the beneficiary.
Bank	It is auto-completed, based on the inserted IBAN.

## The Payer Details Section

This is the bottom section and it contains your company's details. Here, most of the fields are **auto-filled** and **non-editable**. However, in the **Comments** field, you can insert any relevant information to be considered by the user that is going to approve your request for payment. For example, you can write "This payment was allocated only partially and we need to return the rest of the amount." or "This is a payment return." or "Please, approve this return by noon. Thank you!".

Below, you can see the bottom section of an outgoing payment record, in **Draft** status.

This section also contains the **Cancel payment return** and **Propose payment return** buttons. You use the buttons in order to either submit your request to the manual approval flow or cancel the request. When you click the canceling button, the status of the record changes to **Cancelled**. Respectively, when you click the proposal button, the status of the record changes to **Proposed**. For more details about the approval, see below.

### HINT

Once registered into the system, the **referenceNo** attribute for any outgoing payment request is automatically populated by using a **sequencer**.

## Approving Outgoing Payment Requests

See details about how to find your desired record in the [Outgoing Payment Requests View](#) section, at the top of this page. If your record is in **Draft** status, you must first make a proposal for an outgoing payment. If your record is in **Proposed** status already, go to the [Launch the Approval Flow](#) step, below.

Take the below steps for approving payment requests:

### 1. Launch the Proposal Flow

In order to approve an outgoing payment, you must go to the **Outgoing Payment Requests List** page and double-click the desired record to open it. You can find more details about the steps to reach this list in the [Outgoing Payment Requests View](#) section.

Below, you can see an example of an outgoing payment record, in **Draft** status.

OUTGOING PAYMENTS REQUESTS								
<input type="checkbox"/>	Payment No	Reference No	Payment type	Amount	Currency	Payment Be...	Paid Amount	Status
	<input type="text" value="REF0000575"/>	<input type="text" value="REF0000575"/>	<input type="text" value="Unallocated..."/>	<input type="text" value="294.24"/>	<input type="text" value="EUR"/>		<input type="text" value="0.00"/>	<input type="text" value="Draft"/>
	<input type="text" value="REF0000573"/>	<input type="text" value="REF0000573"/>	<input type="text" value="Unallocated..."/>	<input type="text" value="75.00"/>	<input type="text" value="EUR"/>		<input type="text" value="0.00"/>	<input type="text" value="Draft"/>
	<input type="text" value="REF0000572"/>	<input type="text" value="REF0000572"/>	<input type="text" value="Unallocated..."/>	<input type="text" value="70.00"/>	<input type="text" value="RON"/>		<input type="text" value="0.00"/>	<input type="text" value="Draft"/>
	<input type="text" value="REF0000571"/>	<input type="text" value="REF0000571"/>	<input type="text" value="Unallocated..."/>	<input type="text" value="25.00"/>	<input type="text" value="EUR"/>		<input type="text" value="0.00"/>	<input type="text" value="Approved"/>
	<input type="text" value="REF0000570"/>	<input type="text" value="REF0000570"/>	<input type="text" value="Unallocated..."/>	<input type="text" value="50,000.00"/>	<input type="text" value="EUR"/>		<input type="text" value="0.00"/>	<input type="text" value="Cancelled"/>

5 10 20 1 2 3 4 5 ...

Double click the selected record to launch the **Outgoing Payment Requests Form**.

**CURRENT STATUS:**  
DRAFT

**Outgoing Payment Request**

**PAYMENT REQUEST**

Payment No: REF0000575      Created On: 21/10/2021

Outgoing Payment Type: Unallocated Payment      Reference No: REF0000575

Payment Amount: 294.24      Currency: EUR

Payment Due Date: 21/10/2021      Scheduled Date: 21/10/2021

Proposed By: [User Name]

You can check the status of your record, at the top left of your screen.

Inside the **Outgoing Payment Requests Form**, fill in the necessary details as described in [The Payment Beneficiary](#) section.

The screenshot shows the 'Outgoing Payment Request' form. The 'PAYMENT REQUEST' section includes fields for Payment No (REF0000596), Created On (23/10/2021), Outgoing Payment Type (Unallocated Payment), and Reference No (REF0000596). The 'PAYMENT BENEFICIARY' section includes Beneficiary Type ([none]), Beneficiary Category (Other), Name, IBAN, and Bank (BANK BUCURESTI CENTRALA). The 'PAYER DETAILS' section includes Name, IBAN, Bank (BANK BUCURESTI CENTRALA), and Comments. At the bottom, there are two buttons: 'Cancel payment return' and 'Propose payment return'.

Once finished, move to the next step.

## 2. Propose Your Payment Request

At the bottom of the form, locate the **Propose Payment Return** button and click it.

Below, you can see the bottom section of an outgoing payment record, in **Draft** status.

The screenshot shows the 'PAYER DETAILS' section of an outgoing payment record in Draft status. It includes fields for Name, IBAN, Bank (BANK BUCURESTI CENTRALA), and Comments. At the bottom, there are two buttons: 'Cancel payment return' and 'Propose payment return'.

When you click **Propose Payment Return**, the selected record is submitted to the manual outgoing payment request approval flow.

You can check the status of your record, at the top left of your screen. Your record is currently in **Proposed** status, as in the example below.

The screenshot shows a user interface for an 'Outgoing Payment Request' form. At the top left, a dark box displays 'CURRENT STATUS: PROPOSED' in yellow text. The form has two tabs: '1 Outgoing Payment Request' (active) and '2 Approval'. Below the tabs, the form is titled 'PAYMENT REQUEST' and contains several input fields:

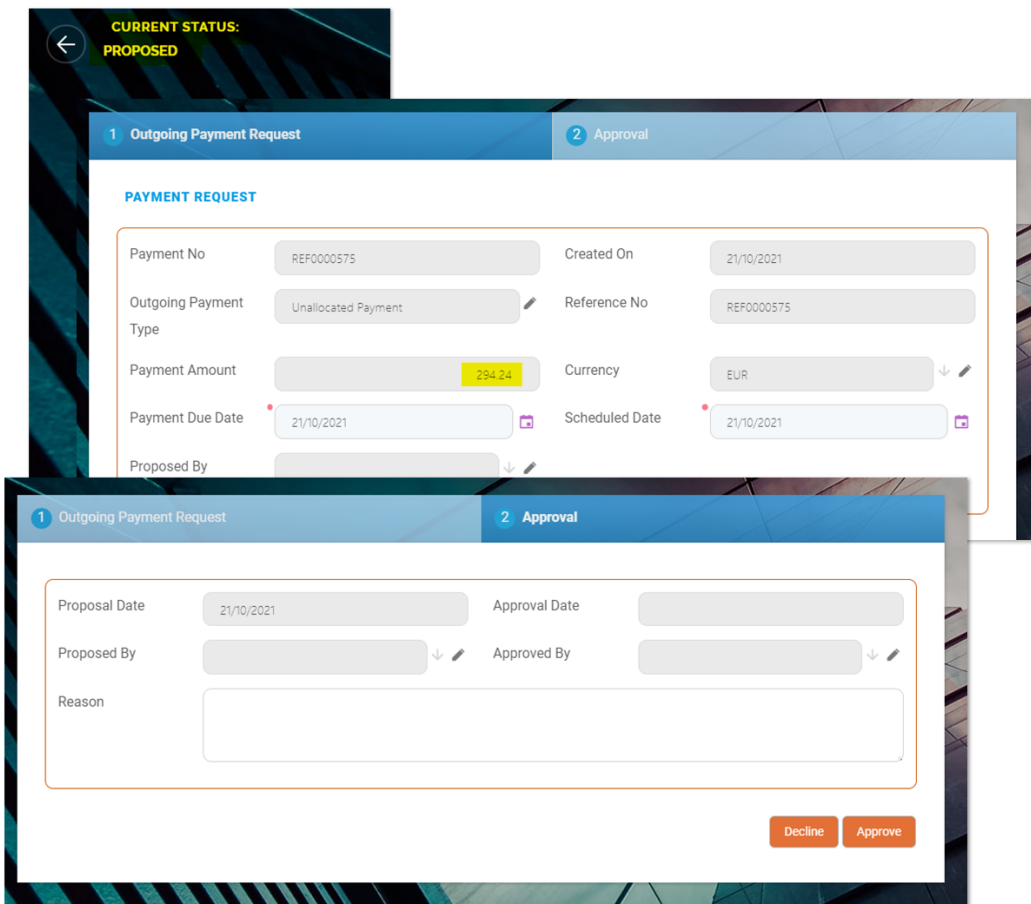
Payment No	REF0000575	Created On	21/10/2021
Outgoing Payment Type	Unallocated Payment	Reference No	REF0000575
Payment Amount	254.24	Currency	EUR
Payment Due Date	21/10/2021	Scheduled Date	21/10/2021
Proposed By			

Continue to the next step.

### 3. Launch the Approval Flow

After pressing the **Propose Payment Return** button, the second tab of the **Outgoing Payment Request** form becomes available. In the first tab, you can see the details of the request and in the second tab, you can use the buttons to either **Approve** or **Decline** the request. Continue to the next step.

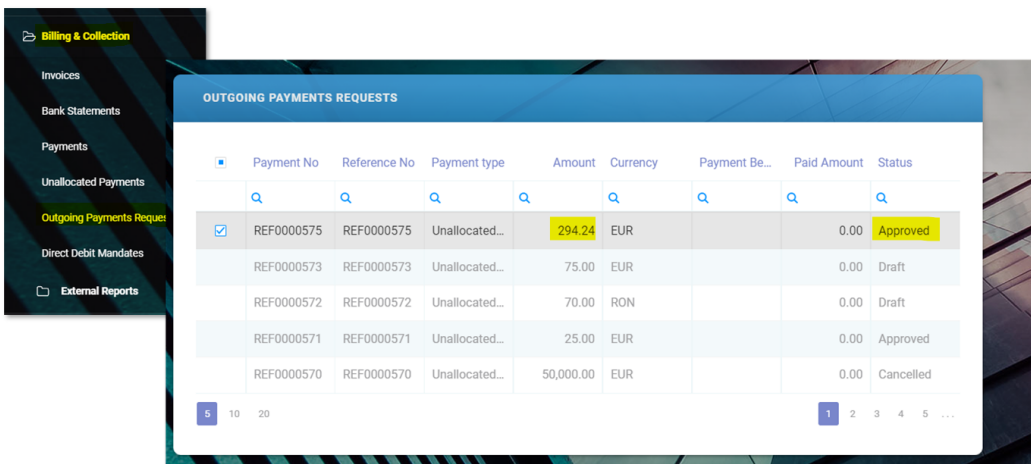
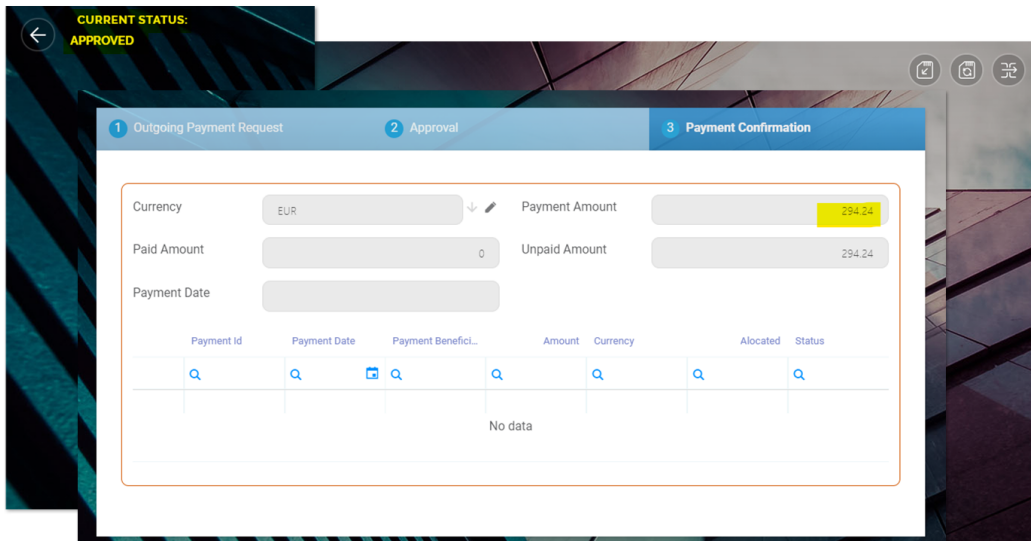
Below, you can see an example of an outgoing payment request form (the header section), with tabs.



## 4. Click Approve

In the approval tab, click **Approve** to validate the outgoing payment. When you click **Approve**, at the top left of your screen, you can see that the business status of the current record changed from **Proposed** to **Approved**.

Below there are two examples of the same record in **Approved** status.



After approval, the outgoing payment is submitted to the automatic allocation flow.

**NOTE**

After a while, and upon a bank statements upload, in which the **Billing and Collection** solution received the payment confirmation for that particular payment request, it registers the confirmation in the system, automatically inserts the necessary details into the payment confirmation fields and changes the status of the record to **Paid**.

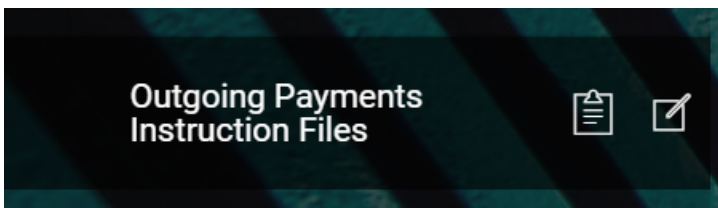
Click **Save and close** at the top right corner of your screen to close the form.



**HINT**

You can easily find your **Draft** payment request in the [Outgoing Payment Requests List](#) page, at a later date. If needed, use the **Search by Status** option to find your record.

## Outgoing Payments Instruction Files




The **Payments Instruction File** is an important element of the automatic payment management featured by the **Billing and Collection** solution. This **File** contains instructions that trigger the payments from the insurer's bank account - for a claim payment, for example. The [FTOS\\_PYMT\\_OutgoingPayment](#) - the **Outgoing Payments Requests** entity, is the source of the **Payments Instruction File**.

A [daily job](#) automatically examines the outgoing payments requests registered into the system - irrespective of the source, process, or system it came from. Next, it prepares the **File** containing the payment instructions for the eligible requests - for example, for the outgoing payments in **Scheduled** status. The **File** is then transmitted to the bank and the amounts are disbursed from the insurer's bank account, according to the instructions.

### OPI Files View and Form

In your portal, the **O(utgoing) P(ayments) I(nstruction) Files** section offers you an overview of all the payment instruction files generated by the system, with the newest record registered at the top.

OUTGOING PAYMENTS INSTRUCTION FILES LIST

<input type="checkbox"/>	File name	Date of generation
	<input type="text" value="Q"/>	<input type="text" value="Q"/> 
	OutgoingPaymentsInstructionFile_20220205.txt	05/02/2022 07:00
	OutgoingPaymentsInstructionFile_20220204.txt	04/02/2022 07:00
	OutgoingPaymentsInstructionFile_20220203.txt	03/02/2022 15:29
	OutgoingPaymentsInstructionFile_20220111.txt	11/01/2022 07:00
	OutgoingPaymentsInstructionFile_20211229.txt	29/12/2021 07:00

5 10 20 1 2 3 4 5 6

**IMPORTANT!**  
 This section contains only the **approved** payment requests, in **Scheduled** status.

This is an all-inclusive view; yet, you also can search and sort your records for easier processing. For example if you want to view the payment instruction file that was generated in a certain day, you can use the **Search by Date** option.

Follow the steps to view your outgoing payments instruction files:

1. In your **FintechOS Portal**, navigate down the main menu of the **Billing and Collection** solution.
2. From the dropdown list, click **Outgoing Payments Instruction Files** to open the **Outgoing Payments Instruction Files List** window.
3. On the **Outgoing Payments Instruction Files List** page, to inspect a record from the grid, double-click it.

Below is an example of a payments instruction file Form:

EDIT OUTGOING PAYMENTS INSTRUCTION FILES

File name  Date of generation

**HINT**

You can export one or more records by pressing **Export**, at the top right corner of your screen.

## Adding Payment Requests

Payment requests are automatically registered by your system - for example, when you use the **Bank Statements** functionality to upload your bank statement files or when you receive an API call, with the payment request details. Consequently, the [Outgoing Payment Requests list view](#) is continuously updated, helping you understand the requests for payment you received, in real time.

However, there are cases when you need to manually insert a new payment request. For example for a one-time payment that goes towards a consultancy company.

**NOTE**

The payer's details (e.g. your company) can be auto-filled by the system. For this you have to configure a default **Payer Name** and **IBAN**, set in the processor `FTOS_PYMT_OutgoingPayments_PayerDetails`.

Take the below steps for adding a new payment request:

### 1. Launch the Outgoing Payment Request Form

In order to insert a new payment request, you must launch the **Outgoing Payment Request Form**. In order to do so, take the following steps:

1. At the top left corner of your **FintechOS Portal**, click the main menu icon to open the main dropdown list.
2. From the main list, click **Billing and Collection**. A second dropdown opens.
3. Next, click **Outgoing Payment Requests** to go to the **Outgoing Payment Requests List**.

4. Inside the **Outgoing Payment Requests List** page, click **Insert**, at the top right of your screen. This launches the **Outgoing Payment Request** proposal form. For more details about how this form is structured, go to the **Outgoing Payment Request Form** section, from the [Outgoing Payment Requests](#) page.

Proceed to the next step.

## 2. Insert the Payment Request Details

When you open the **Outgoing Payment Request** form, you can see the **Payment Request**, **Payment Beneficiary** and **Payer Details** sections and, also, you can see that some fields are gray - those are non-editable fields; they are automatically filled in by the system.

Proceed to insert the necessary details into the **Payment Request** section, as follows:

Field Name	Description
Payment No	Give a registration number to the outgoing payment.
Created On	The day when the request is created. This is automatically filled in by the system.
Payment Amount	The amount to be paid.
Currency	The currency of the incoming payment.
Reference No	The reference number for the outgoing payment. This is automatically filled in by the system.
Payment Type	Use the option set to select a type for the outgoing payment. The types are as follows: <ul style="list-style-type: none"> <li>• paymentOrder</li> <li>• brokerPremiumPayment</li> <li>• paymentUnallocated</li> <li>• bankCharges</li> <li>• outgoingPayment</li> </ul>
Payment Due Date	Use the option set to select a due date for the payment.

Field Name	Description
Scheduled Date	Use the option set to schedule the payment date.
Proposed By	The user who registers the initial payment. This is automatically filled in by the system.

Below is an example of the **Payment Request** section:

Next, proceed to insert the necessary details into the **Payment Beneficiary** section, as follows:

Field Name	Description
Beneficiary Type	<p>Use the option set to select a beneficiary type for the payment. The types are as follows:</p> <ul style="list-style-type: none"> <li>insured</li> <li>contractor</li> <li>policyBeneficiary</li> <li>broker</li> <li>serviceProvider</li> <li>other</li> </ul>

Field Name	Description
Beneficiary Category	Use the option set to select a beneficiary category for the payment. The categories are as follows: <ul style="list-style-type: none"> <li>• Other</li> <li>• Individual Person</li> <li>• Legal Person</li> </ul>
Name	The name of the payment beneficiary.
IBAN	The IBAN of the beneficiary.
Bank	This is the bank of the outgoing payment beneficiary. It is automatically filled in, based on the IBAN number provided.

Below is an example of the **Payment Beneficiary** section:

The screenshot shows a form titled "PAYMENT BENEFICIARY" with the following fields and controls:

- Beneficiary Type:** A dropdown menu with "Select..." as the current selection.
- Beneficiary Category:** A dropdown menu with "Other" as the current selection.
- Name:** A text input field.
- IBAN:** A text input field.
- Bank:** A dropdown menu.

Each field has a red asterisk indicating it is required. There are also edit (pencil) and delete (trash) icons for the dropdown menus.

If not auto-filled, proceed to insert the necessary details into the **Payer Details** section, as follows:

Field Name	Description
Name	The name of the payer - that is your company. This is automatically filled in by the system.
IBAN	The IBAN of the payer. This is automatically filled in by the system.
Bank	The bank of the payer. This is automatically filled in by the system.
Comments	This area is for comments. Insert details relevant for the user that is going to approve your request for payment.

Below is an example of the **Payer Details** section:

**PAYER DETAILS**

Name

IBAN  Bank

Comments

If you press **Save and close** before completing the flow (submitting your request), the record is saved in **Draft** status and you can come back at it, at a later date.

Below, you can see an outgoing payment request in **Draft** status.

**Outgoing Payment Request**

**PAYMENT REQUEST**

Payment No: REF0000596 Created On: 23/10/2021

Outgoing Payment Type: Unallocated Payment Reference No: REF0000596

**PAYMENT BENEFICIARY**

Beneficiary Type: [none] Beneficiary Category: Other

Name:

IBAN:  Bank: BANK BUCURESTI CENTRALA

**PAYER DETAILS**

Name

IBAN  Bank

Comments

To complete the flow, proceed to the next step.

### 3. Submit Your Outgoing Payment Proposal

Once you finished inserting the necessary details, locate the **Propose Payment Return** button, at the bottom of the form, and click it to submit your request to the outgoing payments flow.

After pressing **Propose Payment Return**, you can see, at the top left of your screen, that your record is in **Proposed** status.

Below is an example of an outgoing payment request in **Proposed** status.

The screenshot shows a mobile application interface for a payment request. At the top left, it says 'CURRENT STATUS: PROPOSED'. Below this is a navigation bar with two tabs: '1 Outgoing Payment Request' (active) and '2 Approval'. The main form area is titled 'PAYMENT REQUEST' and contains the following fields:

- Payment No: REF0000575
- Created On: 21/10/2021
- Outgoing Payment Type: Unallocated Payment
- Reference No: REF0000575
- Payment Amount: 294.24
- Currency: EUR
- Payment Due Date: 21/10/2021
- Scheduled Date: 21/10/2021
- Proposed By: (empty field)

Click **Save & Close** at the top right of your screen.

**HINT**

You can easily find your **Draft** payment request in the [Outgoing Payment Requests List](#) page, at a later date. If needed, use the **Search by Status** option to find your record.

## Returning Payments

There are cases when you need to return the payment back to the payer. For example when you cannot allocate the amount because the payment details are missing - such as providing the correct unique identifier for the installment or the number of the



policy. Yet another example is when you cannot allocate the whole amount of the incoming payment and need to return the rest.

**IMPORTANT!** The system does not allow you to return payments that are in **Closed** status (in other words, payments that were fully allocated, already). You can make returns for payments in **Unallocated** or **Partially Allocated** status only.

**Unallocated Payments** can be either handled by the system, automatically or managed manually, by you. For the automatic return, the [FTOS\\_PYMT\\_Payment\\_ReturnUnallocatedAmount](#) job creates payment returns for all payments which are in **Unallocated** status for a number of days that you can configure. For more details, consult the [Flow Parameters and Scheduled Jobs](#) page. For the manual return, you use the **Return** button to initiate the return flow by yourself. More details to follow.

**Partially Allocated Payments**, however, need to be managed only manually. That is: for each partially allocated payment, you need to initiate the return flow by yourself. You can do this easily by pressing the **Return** button inside the selected payment record.

For returning payments, take the below steps:

## 1. Find Your Record and Launch the Return Flow

1. At the top left corner of your **FintechOS Portal**, click the main menu icon to open the main dropdown list.
2. From the main list, click **Billing and Collection**. A second dropdown opens.
3. Next, click **Unallocated Payments** to go to the **Unallocated Payments List**.
4. Inside the **Unallocated Payments List** page, use the search functionality to find the desired payment record.
5. Double click the record to open it. Or click the **Return** button next to it, right away. (If you choose to do so, jump to the [Fill in the Return Details](#) step, below.)

Below is an example of the **Unallocated Payments List** page, with the **Return** button next to the records.

UNALLOCATED PAYMENTS									
<input type="checkbox"/>	Payment ID	Payer Name	Payment Registrati...	Payment Type	Curre...	Amount	Business Status	Option	
	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
	3		25/04/2018	Payment Order	EUR	907.50	Partially Allocated	<input type="button" value="Return"/>	
	3		25/04/2018	Payment Order	EUR	907.50	Unallocated	<input type="button" value="Return"/>	
	1		14/07/2021	Payment Order	RON	0.00	Unallocated	<input type="button" value="Return"/>	
	1		15/04/2021	Payment Order	RON	1,000.00	Unallocated	<input type="button" value="Return"/>	
	1		23/07/2021	Payment Order	EUR	95.00	Unallocated	<input type="button" value="Return"/>	

- Once the record is opened, inside the header section, locate the **Return** button and click it.

Below is an example of a payment record in **Partially Allocated** status, with **Return** button.

**CURRENT STATUS: PARTIALLY ALLOCATED**

**Client Payment**

Payment ID: 2      Payer Name: Crimtech Srl

Payment Registration Date: 01/09/2020      Bank Reference: ING

Payment Date: 08/07/2021      Business Status: **Partially Allocated**

Amount: 1,000      Currency: EUR

Comments:

Proceed to the next step.

## 2. Fill in the Return Details

When you press **Return**, you create a request for payment, in **Draft** status. Inside the **Unallocated Payment** record, scroll down to the **Outgoing Payments** section and double-click on the **Draft** request to open it.

Below, you can see the bottom section of a payment record with one outgoing payment request in **Draft** status.

OUTGOING PAYMENTS						
<input type="checkbox"/>	Payment No	Outgoing Payment T...	Payment Amount	Currency	Created On	Business Status
	<input type="text" value="REF0000571"/>	<input type="text" value="Unallocated Payment"/>	<input type="text" value="25.00"/>	<input type="text" value="EUR"/>	<input type="text" value="20/10/2021"/>	<input type="text" value="Draft"/>

When you open the **Outgoing Payment Request** form, you can see all the details of the **Draft** request that the system automatically fills in - into the **Payment Request**, **Payment Beneficiary**, and **Payer Details** sections. These details are not editable.

Field Name	Description
Payment No	The number of the incoming payment.
Created On	The day when the Outgoing Payment <b>Draft</b> request is created.
Payment Amount	The amount to be returned. It is automatically calculated.
Currency	It matches the currency of the incoming payment.
Reference No	The reference number for the outgoing payment. It is automatically generated to match the number of the incoming payment.
Payment Type	It matches the payment type of the incoming payment. The outgoing payment types are as follows: <ul style="list-style-type: none"> <li>• paymentOrder</li> <li>• brokerPremiumPayment</li> <li>• <b>paymentUnallocated</b> - this is your current case</li> <li>• bankCharges</li> <li>• outgoingPayment</li> </ul>
Proposed By	The user who registers the initial payment.
Bank	In the <b>Payment Beneficiary</b> section, it is the bank that sent the incoming payment.
Name	In the <b>Payer Details</b> section, it is the name of the payer - that is your company.
IBAN	In the <b>Payer Details</b> section, it is the IBAN of the payer.
Bank	In the <b>Payer Details</b> section, it is the bank of the payer.

Below, you can see an outgoing payment request in **Draft** status.

Proceed to fill in the necessary details into the following editable fields:

Field Name	Description
Payment Due Date	The payment due date.
Scheduled Date	The payment scheduled date.
Beneficiary Type	<p>The payment beneficiary type. The types are as follows:</p> <ul style="list-style-type: none"> <li>insured</li> <li>contractor</li> <li>policyBeneficiary</li> <li>broker</li> <li>serviceProvider</li> <li>other</li> </ul>

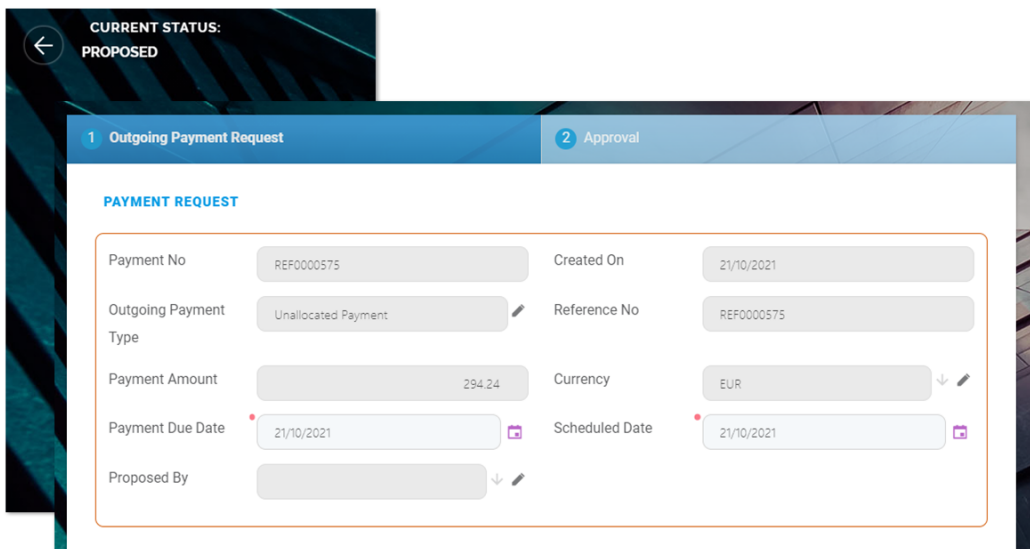
Field Name	Description
Beneficiary Category	The beneficiary category. The category types are as follows: <ul style="list-style-type: none"> <li>• Other</li> <li>• Individual Person</li> <li>• Legal Person</li> </ul>
Name	The name of the payment beneficiary.
IBAN	The IBAN of the beneficiary.
Comments	Insert details relevant for the user that is going to approve your request for payment.

Proceed to the next step.

### 3. Submit your Return Request to Approval

Once finished, press the **Propose Payment Return** button, at the bottom of the form, to submit the request to the outgoing payments flow.

After pressing **Propose Payment Return**, you can see, at the top left of your screen, that your record is in **Proposed** status. However, if you press **Save and close** before this step (submitting your request), the record is saved in **Draft** status and you can come back at it, at a later date. Below is an example of an outgoing payment request in **Proposed** status.



The next step of this flow is the **Approval** of the request. If you have the necessary security role to do the approval, you can continue with the **Outgoing Payment Request** flow. If so, consult the [Outgoing Payment Requests](#) page for more details about approval. If not so, you can leave the record as such, in the **Proposed** status.

Click **Save & Close** at the top right of your screen.

#### HINT

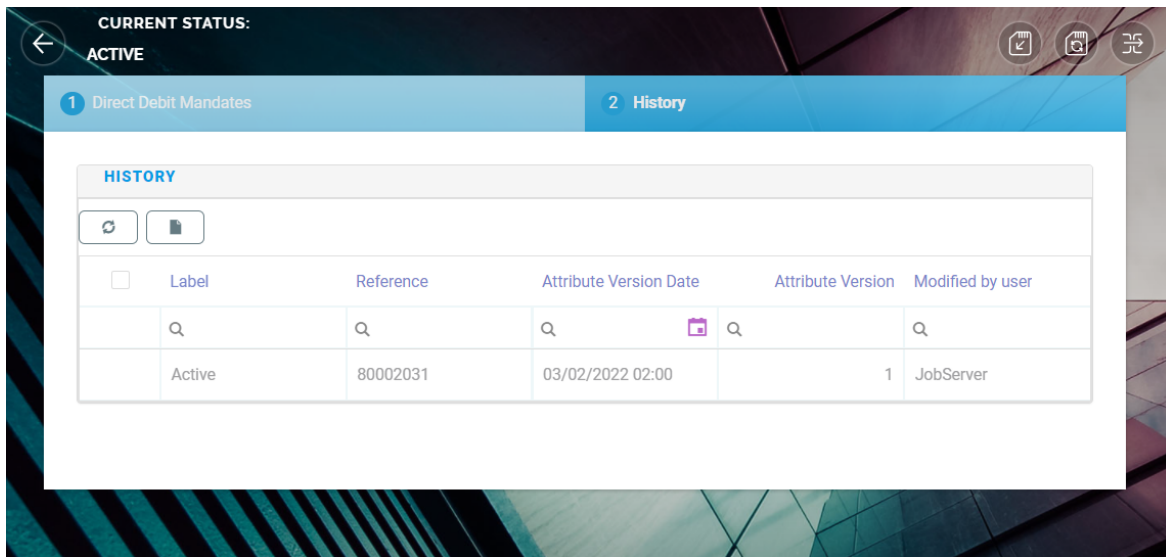
You can easily find your **Draft** payment request in the [Outgoing Payment Requests List](#) page, at a later date. If needed, use the **Search by Status** option to find your record.

## Direct Debit

Once the policyholder agreed to pay the premiums by direct debit, the insurer initiates a direct debit activation procedure in order to notify the bank about the payment arrangement. After the mandate is activated by the bank, the insurer must regularly send the necessary payment instructions files, in order for the bank to respond by transferring the agreed premium amounts into insurer's accounts. When the system receives a notification about the cancellation of a mandate, it changes the status of the specified mandate from **Active** to **Cancelled**, and logs this change into the mandate history.

The **Billing and Collection** solution helps [SEPA\\*](#) and [UK](#) insurers to handle direct debit payment operations - starting with the direct debit mandate activation procedure (in order to notify the bank about the payment arrangement between insurer and insured) up to the moment the payment is collected. Find out more on the [Direct Debit SEPA\\*](#) and [Direct Debit UK](#) pages. **Both direct debit flows are fully automated.** For example, for direct debit payments that respect some rules (such as providing the correct payment details, having an active mandate, and a bank account available for charging), the process of billing and collecting is completely automated, and the insurer does not need to intervene at all. Second to that, some functionalities allow manual interaction, and you can read about those features on the [Debit SEPA\\*](#) and [Direct Debit UK](#) pages.

Below, an example of a mandate activated automatically, by the solution:



**IMPORTANT!**

The solution does not allow for any direct debit mandate to be manually deleted.

To find out more details about how the solution works, check the following pages:

- [Direct Debit SEPA](#) - for business details about the **SEPA\*** flow.
- [Direct Debit UK](#) - for business details about the **UK** flow.
- [SEPA Direct Debit](#) - for technical details about the **SEPA\*** flow.
- [UK Direct Debit](#) - for technical details about the **UK** flow.
- [Flow Parameters And Scheduled Jobs](#) - for details about the automated jobs handling the direct debit processing.

---

\***Single Euro Payments Area (SEPA)** allows customers to make cashless euro payments – via credit transfer and direct debit – to anywhere in the European Union.

## Setting The Solution For DIDE Processing

### 1. Choose A DIDE Type

The **Billing and Collection** solution can be applied for two types of direct debit processing:

- DIDE Sepa - functionality used for the European area,
- DIDE UK - functionality used for the UK area.

For activating your direct debit functionality, you must first choose a direct debit type - either **Sepa** or **Uk**. In order to do so, please take one of the following paths:

In **Innovation Studio** - Digital Experience > Digital Journeys > Processor Settings > Digital Flow Settings List > **FTOS\_PYMT\_DIDEConfiguration** > Edit Digital Flow Settings > **FTOS\_PYMT\_DIDEProcessor** > Edit Processor Settings: Insert the value for the flow you need to implement - either **Sepa** or **Uk**.

In your **Portal** - Main menu > Settings > Flow Settings > Digital Flow Settings List > **FTOS\_PYMT\_DIDEConfiguration** > Edit Digital Flow Settings > **FTOS\_PYMT\_DIDEProcessor** > Edit Processor Settings: Insert the value for the flow you need to implement - either **Sepa** or **Uk**.

If the option configured in the processor is **Sepa**, the **Billing and Collection** solution takes in consideration the functionality implemented for DIDE Sepa. Otherwise, if the inserted value is **Uk**, the solution takes into consideration the flow implemented for DIDE UK.

## 2. Prerequisites Parameters Check

The following parameters must be filled in, according to your needs:

- **numberOfDaysInAdvance**: numeric value expected,
- **runningOnBankHolidays**: boolean value expected,
- **excludeWeekDays**: [text value expected, ... ].

More about these parameters, on the [Flow Parameters And Scheduled Jobs](#) page.

## 3. Insert UK Specific Default Values



This step is only applicable when you are configuring the direct debit functionality for UK.

The following are the available processors:

## DIDEPaymentsFileDefaultValues

This processor sets the value for the **DIDEUK\_Instructions** job that collects all the **Draft** mandates and sends instructions for their activation to BACS.

The following parameters must be filled in, according to your needs:

- **destinationSort**: numeric value expected,
- **destinationAcct**: numeric value expected,
- **destinationType**: 0,
- **usersName**: text value expected

## DIDEPaymentsFileDefaultValuesUk

This processor sets the value for the **FTOS\_PYMT\_ DIDEInstructionFile** job that, for the active mandates registered in the system, assembles the file containing the direct debit payment instructions that are delivered to BACS, and then to the payer's bank.

The following parameters must be filled in, according to your needs:

- **destinationSort**: numeric value expected,
- **destinationAcct**: numeric value expected,
- **destinationType**: 0,
- **transaction**: string value expected,
- **freeFormat**: numeric value expected,

- **amount**: numeric value expected,
- **userName**: text value expected.

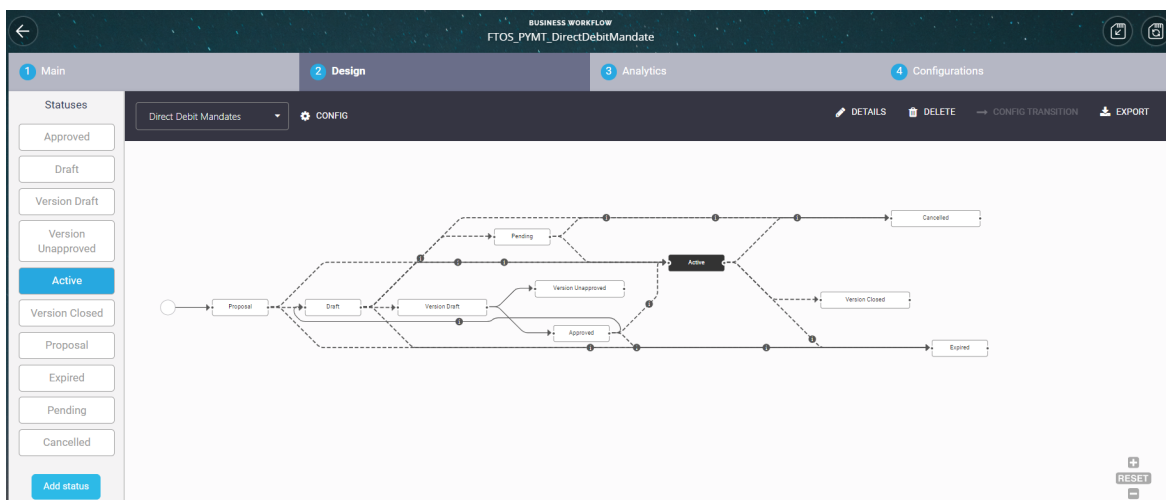
For more details about the jobs and parameters, check the [Flow Parameters And Scheduled Jobs](#) page.

The direct debit functionalities can be applied to policies that have the payment type set to **Direct Debit**, and their business status is in **InForce** or **Suspended**. The solution is going to process all the invoices that are in **Generated** business status - with their **due date** less or equal to the current date, added to the value set in the **numberOfDaysInAdvance** parameter.

## Direct Debit Business Workflow

**FTOS\_PYMT\_DirectDebitMandate** is the master business workflow that handles the different types of changes affecting a direct debit mandate during its lifetime - like transitioning the mandate through different versions and business states (from **Draft** to **Expired**). All the updates are logged in the mandate history tab, for further analytical use.

Below, an example of the **FTOS\_PYMT\_DirectDebitMandate** business workflow - as it is displayed on **Innovation Studio**:



## Mandate Behavior

The following are the behaviors - characteristic to any mandate, managed by this business workflow:

- Every mandate (or mandate version) starts in **Draft** status and must go through an approval process before reaching the **Active** status. Approval is automatic, based on bank response.
- Once a mandate is in **Active** status, its settings can no longer be modified.
- If you want to manually update an **Active** mandate, you must create a new mandate version. After editing, you must manually approve the new mandate version; use the **status picker** to pick the **Approved** status.
- When you create a new mandate version, the current version is retired.
- Only one version of a mandate can be live at one time.
- Only one draft version can be active at one time. If the case, you must close the current draft version and then, open another one.
- Mandates in **Cancelled** status cannot be edited (versioned) - by you or the system.
- No mandate can be deleted manually from the system.
- The system also can automatically adjust a mandate, based on mandate data received. All adjustments are automatically integrated into a new version of the mandate, and logged into the mandate history.

## Mandate Stages

A direct debit mandate stage can take one of the following values:

- **N (new)**,
- **D (deleted, cancelled)**, or
- **M (modified)**.

These stages are registered by the solution, following the processing of payment files/ payment data. For example, if a request for payment through an active mandate is denied (a number of times, which you can configure), this is picked out and sorted by the system, and the mandate is transitioned to the **Deleted** stage. The system also automatically logs this update on the relevant mandate record (this is done by creating an actualized version of the mandate).

## Mandate States

Status name	Description
Proposal	Initial state for any mandate registered in the system.
Draft	Initial state for any mandate registered in the system.
Pending	Ongoing state - the mandate is pending approval from the bank.
Active	Ongoing state.
Version Draft	When the selected mandate becomes editable.
Version Unapproved	When a draft version of a mandate is canceled.
Approved	Ongoing state.
Version Closed	When a draft version of a mandate is closed.
Expired	Final state. The mandate is expired. The mandate cannot be moved from this state to any other states.
Cancelled	Final state. The mandate is cancelled. The mandate cannot be edited or moved from this state to any other states.

## Mandate State Transitions

Transition	Description
_Proposal	Initial state.
Active_Cancelled	When the system registers a notification about the mandate cancellation.
Active_Expired	When the mandate reaches its end day. The expiration triggers the automatic change of the payment type on the policy, from Direct Debit to <b>Bank Transfer (OP)</b> .
Active_VersionClosed	When a version of the mandate is closed. Used for mandate versioning.

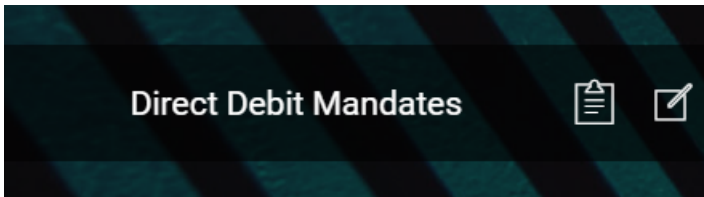
Transition	Description
Approved_Active	When the mandate reaches its start day.
Approved_Draft	When a change of the <b>policy payment type</b> from Bank Transfer (OP) into <b>Direct Debit</b> is performed on a policy.
Approved_Expired	When the number of days for the activation of a mandate were exhausted.
Cancelled_Active	When a mandate is reactivated. This status is only visible/ available for DIDE UK processing.
Draft_Active	When the mandate reaches its start day. This transition is triggered automatically by a specific job that verifies if the current date is the mandate begin date and changes the status to <b>Active</b> for all eligible mandates.
Draft_Cancelled	When the system registers a notification about the mandate cancellation.
Draft_Expired	When the number of days for the activation of a mandate were exhausted. The expiration triggers the automatic change of the payment type on the policy, from Direct Debit to <b>Bank Transfer (OP)</b> .
Draft_Pending	After the instruction file for the mandate's activation is generated, and the mandate is pending approval from the bank.
Draft_VersionDraft	When the mandate versioning process starts.
Pending_Active	When the mandate reaches its start day.
Pending_Cancelled	When the system registers a notification (e.g. AUDDIS file) about canceling the mandate, or a user manually cancels the mandate.

Transition	Description
Proposal_Active	When the mandate reaches its start day.
Proposal_Draft	When the mandate is registered in the system but it is not activated yet, or its begin date is yet to come.
Proposal_Expired	When the number of days for the activation of a draft mandate were exhausted.
VersionDraft_Approved	When a version of the mandate is approved. Used for mandate versioning.
VersionDraft_VersionUnapproved	When the opened version is not approved. Used for mandate versioning.

**HINT**

For more details, consult also the [Flow Parameters And Scheduled Jobs](#) page.

## Direct Debit SEPA



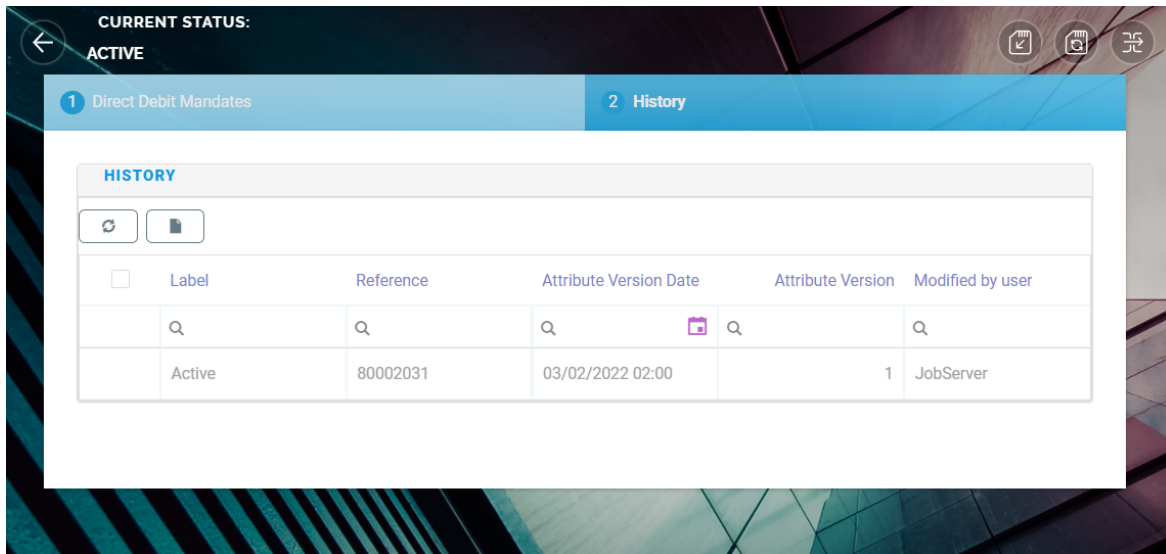
The **Billing and Collection** solution helps **SEPA\*** insurers to handle direct debit payment operations - starting with the direct debit activation procedure (in order to notify the bank about the payment arrangement between insurer and insured) up to the moment the payment is collected.

**IMPORTANT!**

Follow the steps detailed in the [Direct Debit](#) page, for setting the solution to perform direct debit processing for SEPA\* area.

For direct debit payments that respect some rules (such as providing the correct payment details), the process of billing and collecting is completely automated, and you can check the technical details about it on the [SEPA Direct Debit](#) page. Details about the automated jobs handling the direct debit payments can be found on the [Flow Parameters And Scheduled Jobs](#) page.

Below is an example of a mandate activated automatically, by the **Billing and Collection** solution:



**IMPORTANT!**

The direct debit mandates cannot be deleted manually.

\***Single Euro Payments Area (SEPA)** allows customers to make cashless euro payments – via credit transfer and direct debit – to anywhere in the European Union.

## Direct Debit Mandates View

In your portal, the **Direct Debit Mandates** section offers you an overview of the SEPA direct debit mandates registered in your system - with the newest activated mandate at the top.

This view is permanently updating since many of the files are generated by the system, based on available connections with other third-party systems that feed different kinds of direct debit data into the **Billing and Collection**

solution, and also based on different scheduled jobs that automate direct debit processing. Second to that, some functionalities allow manual interaction. Scroll down to the [External Reports](#) section, for more details.

DIRECT DEBIT MANDATES								
<input type="checkbox"/>	Current No.	Payer PIN/UTR	Reference	Amount	Currency	Begin date	End date	Status
	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

This list also gives you the possibility to search and sort the mandates for easier processing. For example if you want to view all the mandates in **Active** status - that is mandates from which premiums are paid, you can use the **Search by Status** option and sort all your mandates accordingly.

Follow the steps below to view the **SEPA** direct debit mandates registered into your system:

1. In your **FintechOS Portal**, navigate down the main menu of the **Billing and Collection** solution.
2. From the dropdown list, click **Direct Debit Mandates** to open the **Direct Debit Mandates** list.

On the **Direct Debit Mandates** page:

- To inspect a record from the grid, double-click it. The form allows you to see the direct debit mandate related details and its history, in a second tab. See details in the next section, below.
- To add a new mandate, manually, click **Insert**, at the top right corner of the page, to open the **Insert Mandate Form**. See details in the [Insert Mandate](#) section, below.



- To edit a mandate, press **Edit Mandate** and use the form to make your adjustments. Editing is available for mandates in **Draft, Pending** or **Active** status. See details in the [Edit Mandate](#) section, below.

**HINT**

You can export one or more records by pressing **Export**, at the top right corner of your screen.

## SEPA Direct Debit Mandate Form

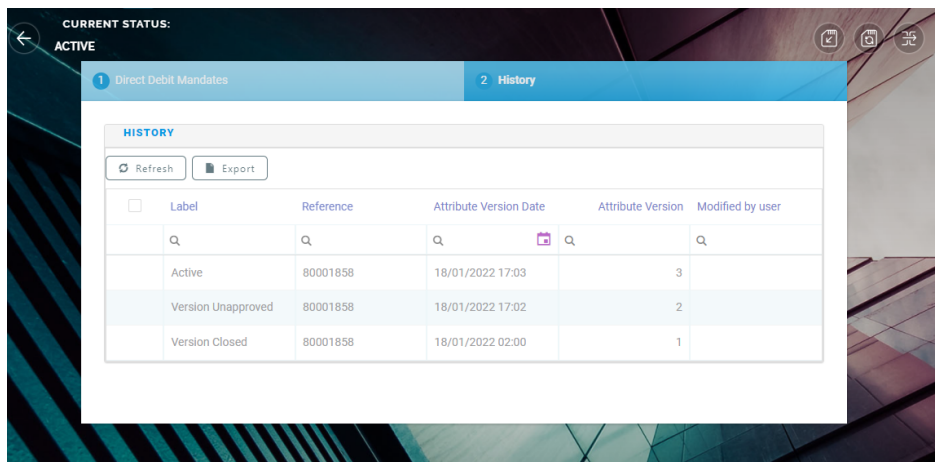
The **SEPA Direct Debit Mandate Form** allows you to inspect, edit or cancel a mandate. The form is organized as follows:

### Direct Debit Mandates Tab

Inside this first section, the following fields are automatically populated with details extracted from the direct debit activation file, coming from the bank and the related policy. These fields are not editable.

Field	Description
Current No.	The number of the current mandate.
Payer first name	The first name of the payer.
Payer last name	The last name of the payer.
Payer PIN/ UTR	The Personal Identification Number or unique ID of the payer.
Reference	The reference for the payments - policy number.
IBAN	The IBAN code for the payments.
Currency	The currency for the payments.
Bank branch	The bank branch.
Amount type	The amount type.
Amount	The amount of the payment.
Mandate stage	The stage of the mandate. The options set values are: <b>New, Deleted, Modified.</b>
Begin date	The beginning date of the mandate. It is automatically completed with the current date - the date when the direct debit <b>Instructions File</b> (containing the instructions for the current mandate, also) is generated in the system.
End date	The end date of the mandate.

## History Tab



Inside this section, you can see the history of the direct debit record. The system makes updates on a direct debit record through the standard versioning mechanism and logs them in this section, helping you to keep track of every change. These fields are not editable.

## Edit a Direct Debit Mandate

1 Direct Debit Mandates
2 History

**DIRECT DEBIT MANDATES**

Current No.	<input type="text" value="1"/>		<input type="button" value="Edit Mandate"/>
Payer first name	<input type="text"/>	Payer last name	<input type="text"/>
Payer PIN/UTR	<input type="text" value="1870927409112"/>	Reference	<input type="text" value="80001994"/>
IBAN	<input type="text"/>	Currency	<input type="text" value="EUR"/> <small>↓ ✎</small>
Bank branch	<input type="text" value="Colentina"/>	Amount type	<input type="text" value="F"/>
Amount	<input type="text" value="25"/>	Mandate stage	<input type="text" value="Modified"/> <small>✎</small>
Begin date	<input type="text" value="27/01/2022"/>	End date	<input type="text" value="29/01/2022"/>

The versioning functionality allows you to edit a mandate, and all your updates are logged into the system. After editing, you must manually approve the new mandate version. See the instructions below.

**IMPORTANT!**  
Mandates in **Cancelled** status cannot be edited.

1. Click **Edit Mandate** to allow the read-only fields to become editable.

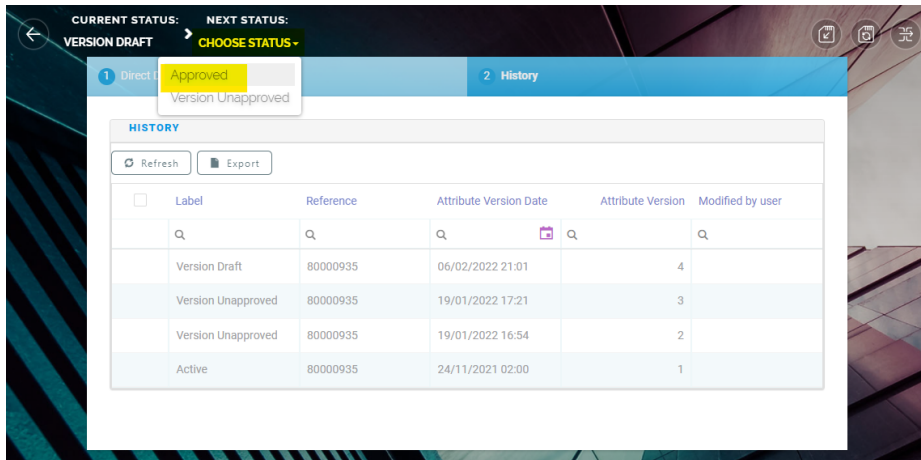
Below , an example of a SEPA mandate ready to be adjusted:

However, some fields are still not editable (see the table below). Proceed to make your changes into the form. Inside this first tab, the fields are automatically populated with the direct debit record details. The following fields are available:

Field	Description
Current No.	The number of the current mandate. <b>Not editable!</b>
Payer first name	The first name of the payer.
Payer last name	The last name of the payer.
Payer PIN/ UTR	The Personal Identification Number or unique ID of the payer.
Reference	The reference for the payments - policy number. <b>Not editable!</b>
IBAN	The IBAN code for the payments.
Currency	The currency for the payments. <b>Not editable!</b>
Bank branch	The bank branch.
Amount type	The amount type.
Amount	The amount of the payment.
Mandate stage	The stage of the mandate. The options set values are: <b>New, Deleted, and Modified. Not editable!</b>
Begin date	The beginning date of the mandate.
End date	The end date of the mandate.

2. Use the **Status picker**, at the top left of the form, to approve your changes.

Below is an example of a SEPA mandate History tab and the **Status picker** (actually available on both tabs of the form):



3. Once approved, the **History** tab opens and you can check the logging of the version you recently created on the selected direct debit mandate.

4. Click **Save and close**.

## Cancel a Direct Debit Mandate

**IMPORTANT!**

Canceling a mandate triggers the cancellation of **all its correlated invoices** - that are in **Generated** status.

Follow the steps below, to cancel a mandate:

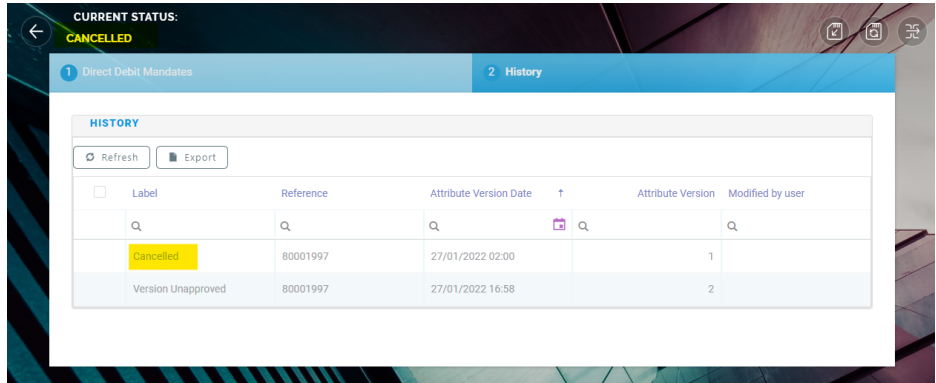
1. Click **Cancel mandate** to move the mandate into this final state.

**NOTE**

Cancellation is irreversible and the mandate cannot be edited after this step.

2. Once cancelled, the **History** tab opens and you can check the logging of this final adjustment you made on the mandate.

Below is an example of the History tab of a cancelled SEPA mandate:



3. Click **Save and close**.

## Insert SEPA Mandate Form

Even if rare, there are cases when you need to manually insert a mandate, on behalf of a customer. For this, you use the **Insert Mandate Form**. All mandates inserted manually are registered in **Draft** status, with **mandateStage = N** (new) and **Begin Date = null**. Next, for all the mandates - SEPA and UK, registered in the system, the **FTOS\_PYMT\_DirectDebitMandateStatus** scheduled job moves the mandate in **Active** status, based on some rules. More on the [SEPA Direct Debit](#) page.

### IMPORTANT!

Manually, you can only add mandates with the mandate stage set to **New**. Additionally, a system validation is in place that prevents you from adding a mandate with the **same reference** (policy number) as any other mandate already existing in the system.

Below is an example of an **Insert Mandate Form**:

Insert Mandate

**DIRECT DEBIT MANDATES**

Current No.

Payer first name  Payer last name

Payer PIN/UTR  Reference

IBAN  Currency

Bank branch  Amount type

Amount  Mandate stage

Begin date  End date

Follow the steps below to add a mandate:

1. In your **FintechOS Portal**, navigate down the main menu of the **Billing and Collection** solution.
2. From the dropdown list, click **Direct Debit Mandates** to open the **Direct Debit Mandates** list.
3. On the **Direct Debit Mandates** page, click **Insert**, at the top right corner of the page, to open the **Insert Mandate Form**.
4. Proceed to insert the necessary details into the form, as follows:

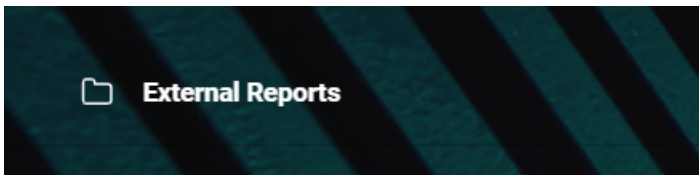
Field	Description
Current No.	The number of the mandate. <b>Not editable!</b> Automatically generated by the system (unique number).
Payer first name	The first name of the payer.
Payer last name	The last name of the payer.
Payer PIN/ UTR	The Personal Identification Number or unique ID of the payer.
Reference	The reference for the payments - policy number. <b>Not editable!</b>
IBAN	The IBAN code for the payments.
Currency	Choose from the dropdown the currency for the payments.
Bank branch	The bank branch.



Field	Description
Amount type	The amount type. String values: <b>M</b> (maximum amount) or <b>F</b> (fixed amount).
Amount	The amount of the payment (with 2 decimals).
Mandate stage	The stage of the mandate. By default completed with <b>N</b> (new). <b>Not editable!</b>
Begin date	When the direct debit process should start.
End date	When the direct debit process should end.

5. Click **Save and close**.

## External Reports



The **External Reports** hold details referring to the management of direct debit mandates - the notifications sent by the bank regarding the activation of the mandate and also about its status (such as if the mandate is still active), the instruction files containing the payment requests for the selected installments, the details regarding the success or failure of the direct debit transactions, and more.

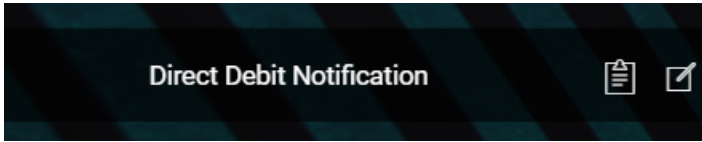
In the **External Reports** section, you can search inside the repositories by **File** or **Import Date**. You can inspect a record by double-clicking it.

### **IMPORTANT!**

The bulk of handling of these operations is done through automatic flows. Where file insert is possible, you will find an **Insert** button. Uploading direct debit data into the system, triggers automatic changes and tracking of those changes. Updates are made and logged for every record through **FintechOS** standard versioning mechanism.

Inside **External Reports**, the following sections are available:

## Direct Debit Notification Section



The **Direct Debit Notifications** sections displays all the files that contain **notifications about changes** made on the direct debit mandates registered on the system. A notified mandate is a mandate that received an update.

This section is permanently updating since many of the updates are generated by the system, also, based on different scheduled jobs that automate direct debit processing. For example if a request for payment through an active mandate is denied (a number of times), the system changes the status of the mandate to **Deleted**.

DIRECT DEBIT NOTIFICATIONS LIST	
File	Import Date
<input type="text" value=""/>	<input type="text" value=""/>
Mandat ex2 - update D to M.txt	17/09/2021 17:41
Mandat ex2 - update D to M.txt	17/09/2021 16:31
Mandat ex2 - draft update 2.txt	17/09/2021 11:49
Mandat ex2 - draft update.txt	16/09/2021 21:16
Mandat ex2 - draft.txt	16/09/2021 21:14

**NOTE**  
 In this section, file import is possible. You can use this functionality to bulk upload notifications (updates) about the existing mandates. A **Direct Debit Notification** file must have **.csv** or **.txt** format in order to be processed and stored by the system.

There are also cases when you need to manually modify some details existing on mandate in **Active** status - for example, the policyholder's account number has changed and there is a business request to update the mandate

urgently, instead of waiting for the automated jobs to handle the change. You can do this by importing a direct debit notification file with **M** (modified) type.

Follow the steps below in order to manually import a **Direct Debit Notification** file:

## Import File Instructions

Below is an example of an **Import Form**:

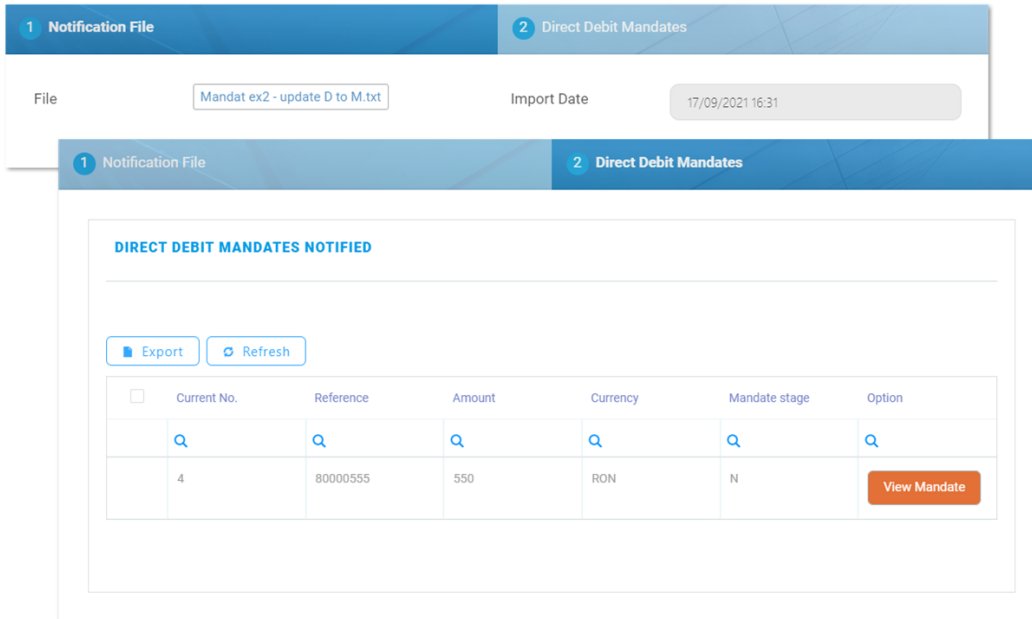
1. While in the same section, click **Insert**, at the top right corner of your screen.
2. When the **Import Form** opens, click on the **Select File** button and insert your **Direct Debit Notification File** and then click **Import Data**.

### **IMPORTANT!**

You must import an **M** type direct debit **Notification File** in order to modify an **Active** mandate.

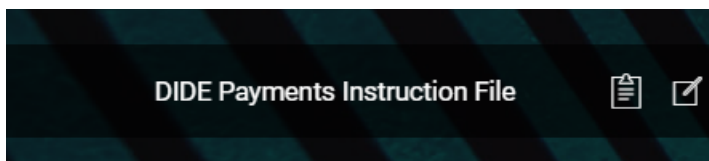
3. (Optional) Check the tabs of the record, for further details. See also the next paragraph that explains how the record is organized.
4. Click **Save and close**.

Inside any **Direct Debit Notification** record, the details about the notified mandate are organized as follows:



- The **Notification File** tab - This first tab includes the actual **File** with the notification for payment and the auto-populated **Import Date** field.
- The **Direct Debit Mandate** tab - This second tab includes a **View Mandate** button which allows you to visualize the actual **mandate record**.

## DIDE Payments Instruction Files Section

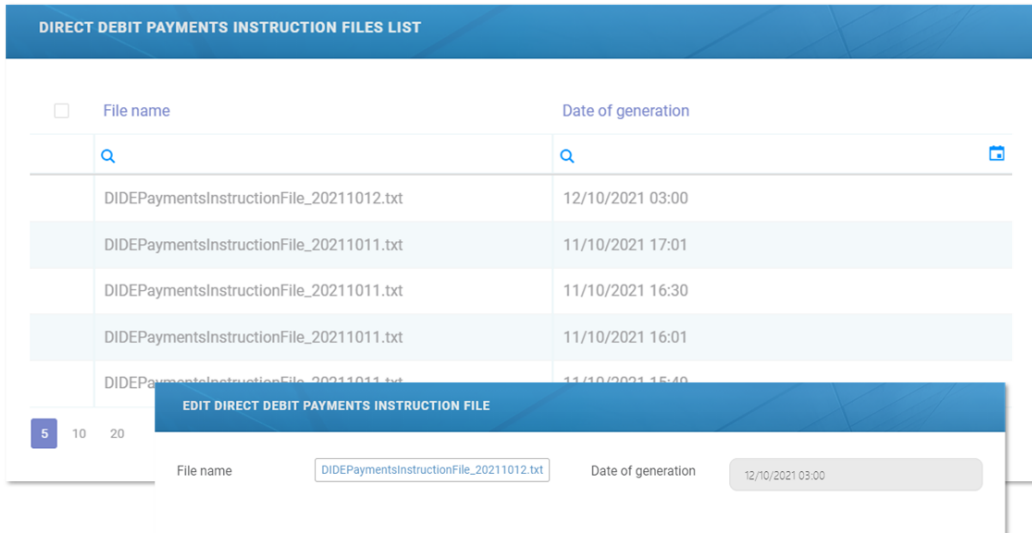


### NOTE

All the details from this section are automatically filled in by the system and they are not editable.

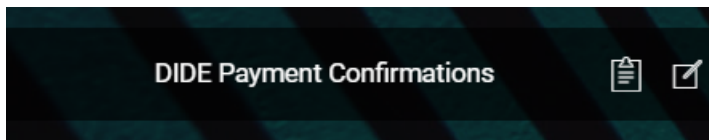
A **direct debit job** runs daily in the system, verifying all the policies with direct debit payment type and their payment schedules, for all the insurance products that are in **Active** status. Where the case, for all the qualifying

installments, this job generates invoices and automatically sends to the bank a request for payment in form of a direct debit instruction file containing the invoice details.



The **Direct Debit Payment Instruction Files** section hosts all your files containing the instructions for direct debit payments generated by the system. You can search inside this repository by **File** or **Date of Generation**. You can inspect a record by double-clicking it. If you want to inspect the details inside any **Direct Debit Payment Instruction File**, you must download the file.

## DIDE Payment Confirmations Section



### NOTE

In this section, file import is possible. A **DIDE Payment Confirmations File** must have **.txt** format in order to be processed and stored by the system.

The **DIDE Payment Confirmations** section contains details about all the direct debit payments confirmed in the system. This section is permanently updating since many of the files are generated by the system, based on different scheduled jobs that automate direct debit processing.

File	Import Date
Payments confirmation file.txt	01/10/2021 14:13
Payments confirmation file.txt	01/10/2021 14:11
Payments confirmation file.txt	01/10/2021 14:11
Payments confirmation file.txt	01/10/2021 14:11
Payments confirmation file.txt	01/10/2021 14:07

You can search inside this repository by **File** or **Import Date**. You can inspect a record by double-clicking it. If necessary, you can manually import a **DIDE Payment Confirmations** file. See right below:

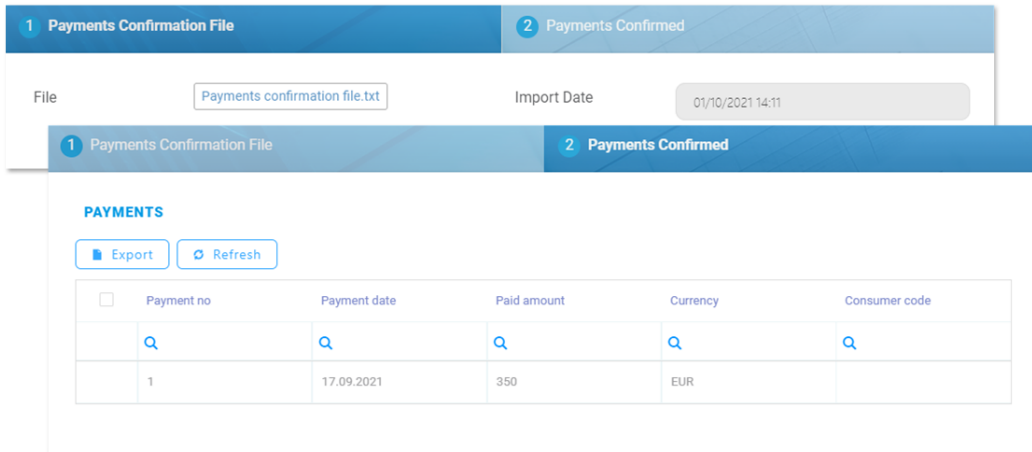
## Import File Instructions

Below is an example of an **Import Form**:

1. While in the same section, click **Insert**, at the top right corner of your screen.
2. When the **Import Form** opens, select your file and then click **Import Data**.

3. (Optional) Check the tabs of the record, for further details. See also the next paragraph that explains how the record is organized.
4. Click **Save and close**.

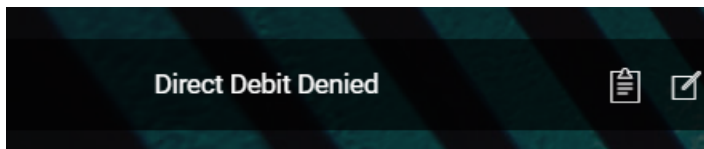
Inside any **Direct Debit Confirmation** record, the details about the confirmed payment are organized as follows:



The first tab - includes the actual **File** with the confirmation of a payment for a specific mandate and the auto-populated **Import Date** field.

The second tab - includes a grid containing every denied payment extracted from the imported file. For the case when there are many records listed, you also can search inside this grid by different keywords - such as **Payment no**, **Payment date**, **Paid amount**, **Currency**, or **Consumer code**.

## Direct Debit Denied Section



**NOTE**

In this section, file import is possible. A **Direct Debit Denied** must have **.csv** or **.txt** format in order to be processed and stored by the system.

The **Direct Debit Denied** section contains details about all the requests for direct debit payment that were denied by the bank and are registered into your system. This section is permanently updating since many of the files are generated by the system, based on different scheduled jobs that automate direct debit processing.

DIRECT DEBIT DENIED LIST	
File	Import Date
Payments denied file - Copy (2).txt	11/10/2021 17:03
Payments denied file - Copy (2).txt	11/10/2021 16:47
Payments denied file - Copy (2).txt	11/10/2021 16:03
Payments denied file - Copy (2).txt	11/10/2021 16:00
Payments denied file - Copy (2).txt	08/10/2021 17:51
Payments denied file - Copy (2).txt	06/10/2021 18:23
Payments denied file - Copy (2).txt	06/10/2021 18:10

You can search inside this repository by **File** or **Import Date**. You can inspect a record by double-clicking it. If necessary, you can manually import a **Direct Debit Denied** file. See right below:

## Import File Instructions

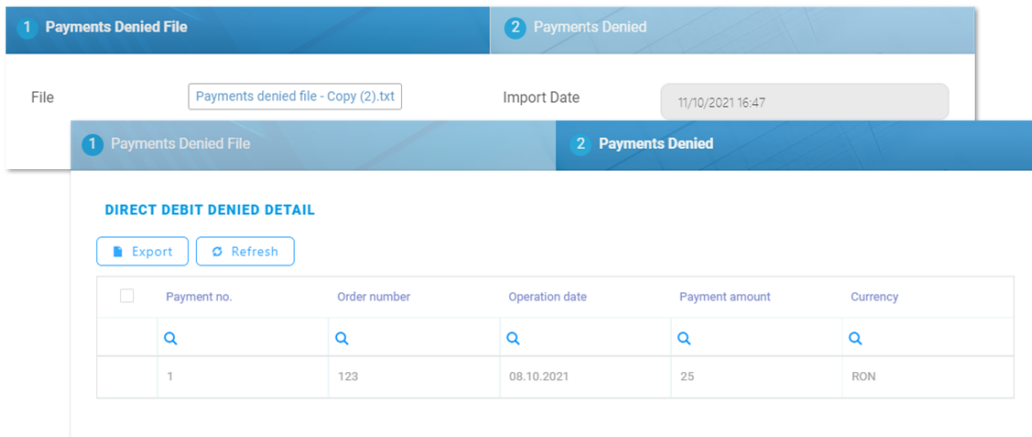
Below is an example of an **Import Form**:

The screenshot shows a mobile-style 'Import' form. It has a blue header with the word 'Import'. Below the header, there are two input fields: 'Import Date' followed by a grey text box, and 'File' followed by a button labeled 'Select file' and the text 'or Drop file here'. At the bottom right of the form is an orange button labeled 'Import Data'.



1. While in the same section, click **Insert**, at the top right corner of your screen.
2. When the **Import Form** opens, select your file and then click **Import Data**.
3. (Optional) Check the tabs of the record, for further details. See also the next paragraph that explains how the record is organized.
4. Click **Save and close**.

Inside any **Direct Debit Denied** record, the details about the denied payment are organized as follows:



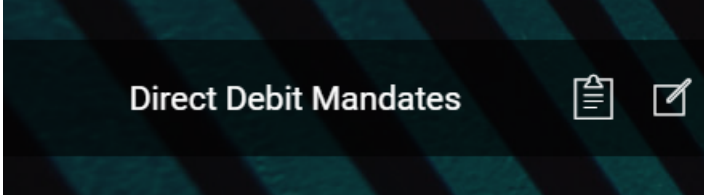
The first tab - includes the actual **File** with the details of the denied request for direct debit payment for a specific mandate, from the bank and the auto-populated **Import Date** field.

The second tab - includes a grid containing every denied payment extracted from the imported file. For the case when there are many records listed, you also can search inside this grid by different keywords - such as **Payment no.**, **Order number**, **Operation date**, or **Payment amount**.

**HINT**

For more information about the import rules and configurations, consult also the [Direct Debit](#) and the [Flow Parameters And Scheduled Jobs](#) pages.

# Direct Debit UK

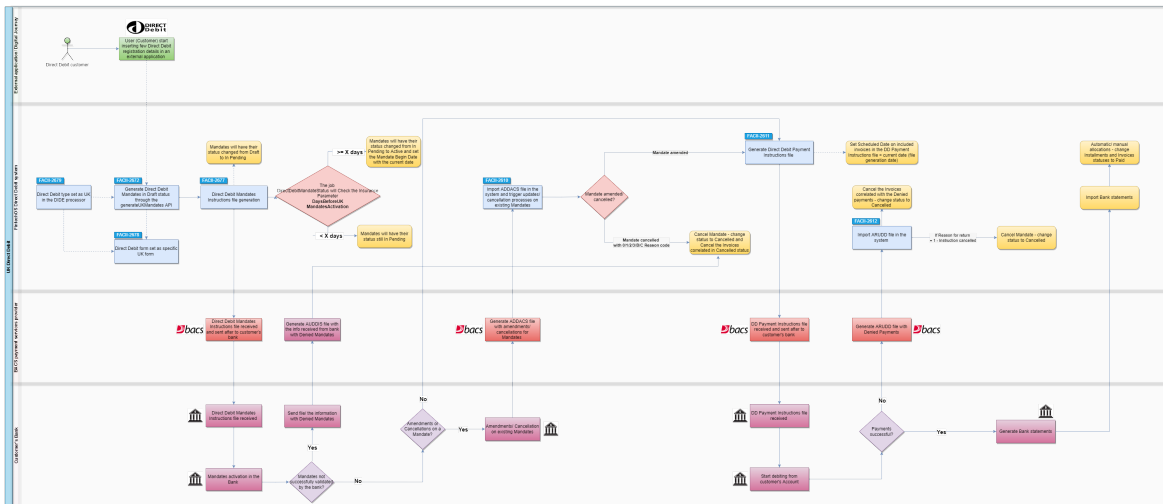


The **Billing and Collection** solution helps **UK** insurers to handle direct debit payment operations - starting with the direct debit mandate activation procedure (in order to notify the bank about the payment arrangement between insurer and insured) up to the moment the payment is collected.

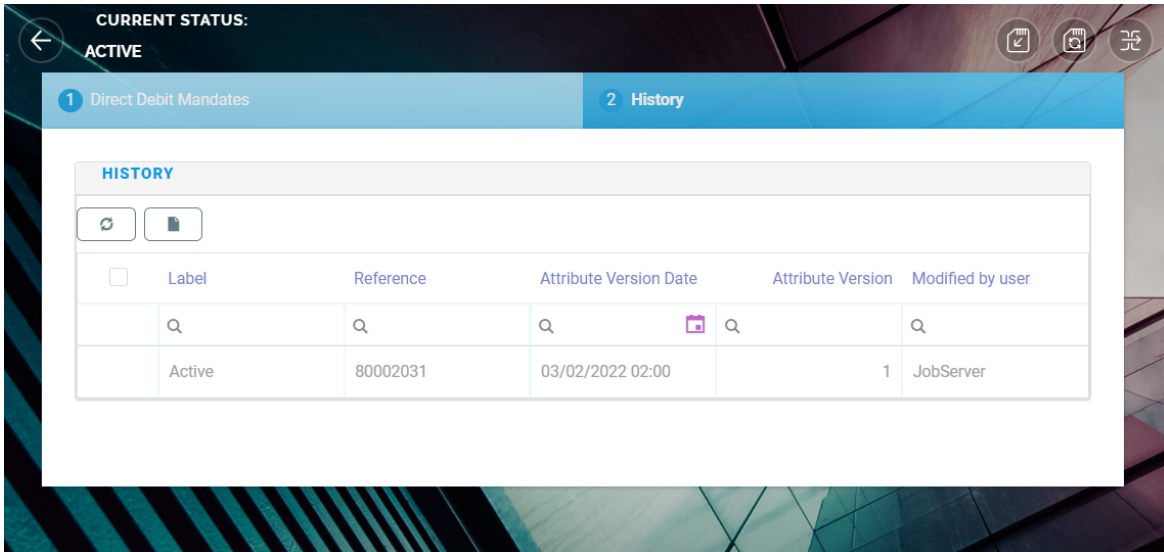
**IMPORTANT!**  
 Follow the steps detailed in the [Direct Debit](#) page, for setting the solution to perform direct debit processing for UK area.

For direct debit payments that respect some rules (such as providing the correct payment details), the process of billing and collecting is completely automated, and you can check the technical details about it on the [UK Direct Debit](#) page. Details about the automated jobs handling the direct debit processing can also be found on the [Flow Parameters And Scheduled Jobs](#) page.

The following is a diagram of how the **Billing and Collection** solution handles the processing of direct debit payments for the UK area.



Below is an example of a mandate activated automatically, by the **Billing and Collection** solution:



**NOTE**

The direct debit mandates cannot be deleted manually.

## Direct Debit Mandates View

In your portal, the **Direct Debit Mandates** section offers you an overview of the UK direct debit mandates registered in your system - with the newest activated mandate at the top.

This view is permanently updating since many of the files are generated by the system, based on available connections with other third-party systems that feed different kinds of direct debit data into the **Billing and Collection** solution, and also based on different scheduled jobs that automate direct debit processing. Second to that, some functionalities allow manual interaction. Scroll down to the [Direct Debit UK Functionalities](#) section, for more details.

DIRECT DEBIT MANDATES								
<input type="checkbox"/>	Current No.	Payer PIN/UTR	Reference	Amount	Currency	Begin date	End date	Status
<input type="checkbox"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
	1	1870927409112	80000699	25.00	EUR	13/10/2021	13/10/2022	Cancelled
	1	1870927409112	80000646	25.00	EUR	13/10/2021	13/10/2022	Active
	1	1870927409112	80000658	25.00	EUR	11/10/2021	11/10/2021	Expired
	2	1870927409112	80000659	25.00	EUR	11/10/2021	11/10/2021	Cancelled
	1	1870927409112	80000653	25.00	EUR	11/10/2021	11/10/2021	Cancelled

This list also gives you the possibility to search and sort the mandates for easier processing. For example if you want to view all the mandates in **Active** status - that is mandates from which premiums are paid, you can use the **Search by Status** option and sort all your mandates accordingly.

Follow the steps below to view the **UK** direct debit mandates registered into your system:

1. In your **FintechOS Portal**, navigate down the main menu of the **Billing and Collection** solution.
2. From the dropdown list, click **Direct Debit Mandates** to open the **Direct Debit Mandates** list.

On the **Direct Debit Mandates** page:

- To inspect a record from the grid, double-click it. The form allows you to see the direct debit mandate related details and its history, in a second tab. See details in the next section, below.
- To add a new mandate, manually, click **Insert**, at the top right corner of the page, to open the **Insert Mandate Form**. See details in the [Insert Mandate](#) section, below.
- To edit a mandate, press **Edit Mandate** and use the form to make your adjustments. Editing is available for mandates in **Draft**, **Pending** or **Active** status. See details in the [Edit Mandate](#) section, below.

**HINT**

You can export one or more records by pressing **Export**, at the top right corner of your screen.

## UK Direct Debit Mandate Form

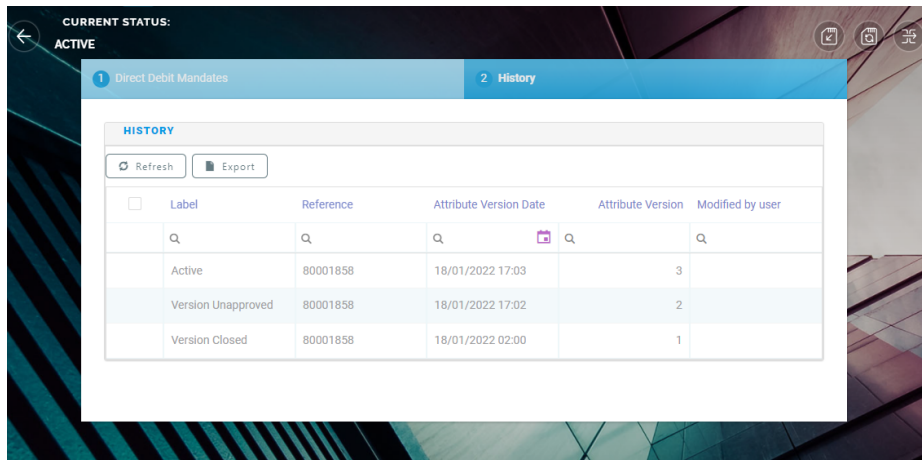
The **UK Direct Debit Mandate Form** allows you to inspect, edit or cancel a mandate. The form is organized as follows:

### Direct Debit Mandates Tab

Inside this first section, the following fields are automatically populated with details extracted from the direct debit activation file, coming from the bank and the related policy. These fields are not editable.

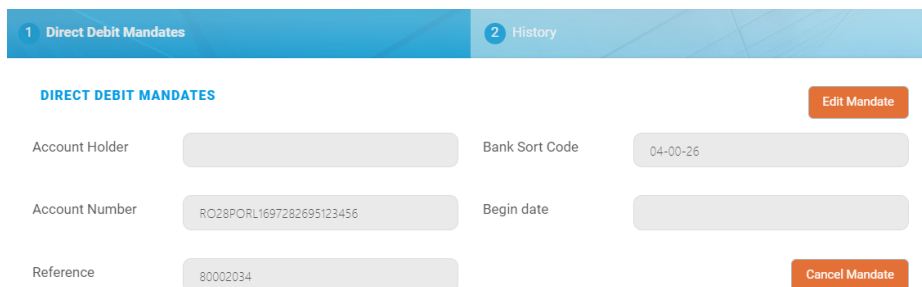
Field	Description
Account Holder	The name of the payer.
Bank Sort Code	The bank sort code.
Account Number	The account number used for direct debit payments.
Begin date	The beginning date of the mandate. It is automatically completed with the current date - the date when the direct debit <b>Instructions File</b> (containing the instructions for the current mandate, also) is generated in the system.
Reference	The reference for the payments - policy number.

### History Tab



Inside this section, you can see the history of the direct debit record. The system makes updates on a direct debit record through the standard versioning mechanism and logs them in this section, helping you to keep track of every change. These fields are not editable.

## Edit a UK Direct Debit Mandate



The versioning functionality allows you to edit a mandate, and all your updates are logged into the system. After editing, you must manually approve the new mandate version. See the instructions below.

**IMPORTANT!**  
Mandates in **Cancelled** status cannot be edited.

1. Click **Edit Mandate** to allow the read-only fields to become editable.

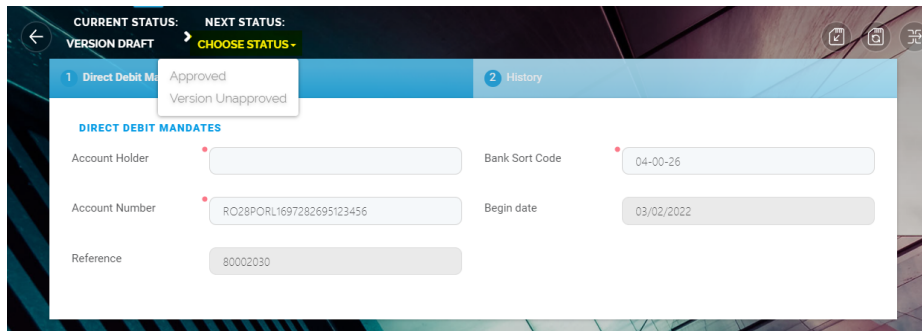
Below, an example of a UK mandate ready to be adjusted:

However, some fields are still not editable (see the table below). Proceed to make your changes into the form. Inside this first tab, the fields are automatically populated with the direct debit record details. The following fields are available:

Field	Description
Account Holder	The name of the payer, account holder.
Bank Sort Code	The bank sort code.
Account Number	The account number of the payer.
Begin date	The beginning date of the mandate. <b>Not editable!</b>
Reference	The reference for the payments - policy number. <b>Not editable!</b>

2. Use the **Status picker**, at the top left of the form, to approve your changes.

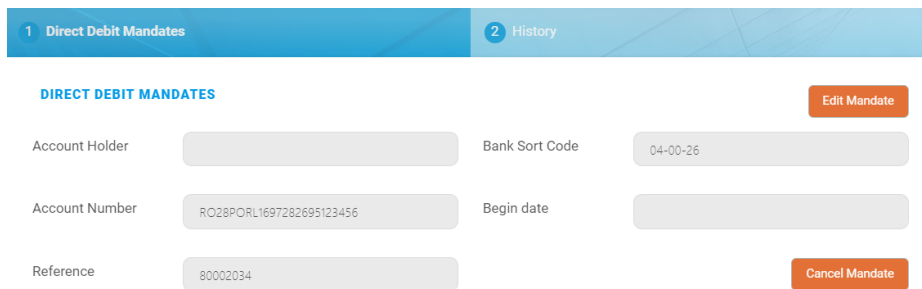
Below is an example of a UK mandate History tab and the **Status picker** (actually available on both tabs of the form):



3. Once approved, the **History** tab opens and you can check the logging of the version you recently created on the selected direct debit mandate.

4. Click **Save and close**.

## Cancel a Direct Debit Mandate



### IMPORTANT!

Canceling a mandate triggers the cancellation of **all its correlated invoices** - that are either in **Generated** or in **OnGrace** status.

Follow the steps below, to cancel a mandate:

1. Click **Cancel mandate** to move the mandate into this final state.

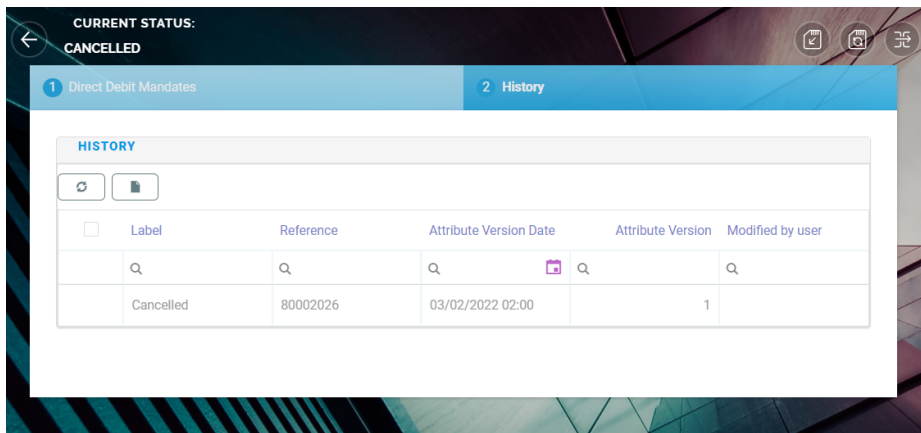


**NOTE**

Cancellation is irreversible and the mandate cannot be edited after this step.

2. Once cancelled, the **History** tab opens and you can check the logging of this final adjustment you made on the mandate.

Below is an example of the History tab of a cancelled UK mandate:



3. Click **Save and close**.

## Insert UK Mandate Form

Even if rare, there are cases when you need to manually insert a mandate, on behalf of a customer. For this, you use the **Insert Mandate Form**. All mandates inserted manually are registered in **Draft** status, with **mandateStage = N** (new) and **Begin Date = null**. Next, for all the mandates - SEPA and UK, registered in the system, the **FTOS\_PYMT\_DirectDebitMandateStatus** scheduled job moves the mandate in **Active** status, based on some rules. More on the [UK Direct Debit](#) page.

**IMPORTANT!**

You can only add mandates with the mandate stage set to **New**. Additionally, a system validation is in place that prevents you from adding

a mandate with the **same reference** (policy number) as any other mandate already existing in the system.

Below is an example of an **Insert Mandate Form**:

Insert Mandate

**DIRECT DEBIT MANDATES**

Account Holder  Bank Sort Code

Account Number  Begin date

Reference

Follow the steps below to add a mandate:

1. In your **FintechOS Portal**, navigate down the main menu of the **Billing and Collection** solution.
2. From the dropdown list, click **Direct Debit Mandates** to open the **Direct Debit Mandates** list.
3. On the **Direct Debit Mandates** page, click **Insert**, at the top right corner of the page, to open the **Insert Mandate Form**.
4. Proceed to insert the necessary details into the form, as follows:

Field	Description
Account Holder	The name of the payer, account holder.
Bank Sort Code	The bank sort code.
Account Number	The account number of the payer.
Begin date	The beginning date of the mandate. <b>Not editable!</b>
Reference	The reference for the payments - policy number.

5. Click **Save and close**.

## Direct Debit UK Functionalities

In order to accommodate the differences regarding the direct debit payments, the **Billing and Collection** solution has dedicated workflows for the **Single Euro Payments Area (SEPA)** and for the **UK** financial area. Each flow has different menu items that display their respective functionalities and help you to handle direct debit payments processing according to SEPA or UK regulations.

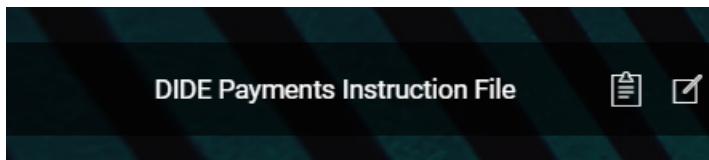
Read below about the different functionalities that are part of the UK direct debit flow.

### IMPORTANT!

The bulk of handling of these operations is done through automatic flows. Where file insert is possible, you will find an **Insert** button. Uploading direct debit data into the system, triggers automatic changes and tracking of those changes. Updates are made and logged for every record through **FintechOS** standard versioning mechanism.

The following sections are available:

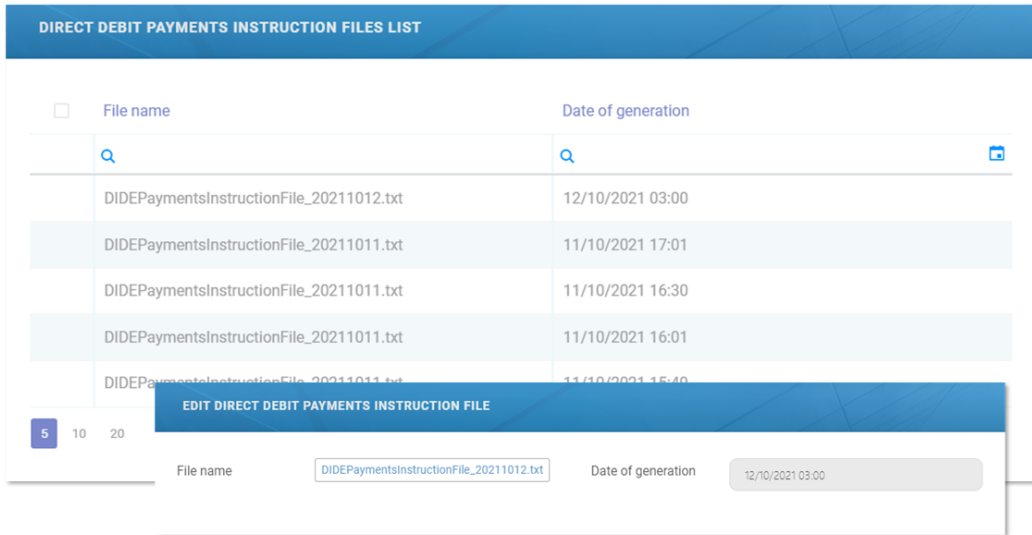
## DIDE Payments Instruction Files Section



### NOTE

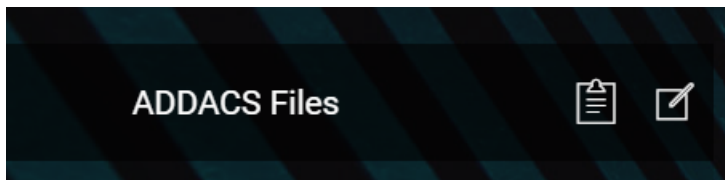
All the details from this section are automatically filled in by the system and they are not editable.

A **direct debit job** runs daily in the system, verifying all the policies with direct debit payment type and their payment schedules, for all the insurance products that are in **Active** status. Where the case, for all the qualifying installments, this job generates invoices and automatically sends to the bank a request for payment in form of a direct debit instruction file containing the invoice details.



The **Direct Debit Payment Instruction Files** section hosts all your files containing the instructions for direct debit payments generated by the system. You can search inside this repository by **File** or **Date of Generation**. You can inspect a record by double-clicking it. If you want to inspect the details inside any **Direct Debit Payment Instruction File**, you must download the file.

## ADDACS Files Section



### HINT

The **ADDACS report (file)** contains data that impact the status of the **UK Direct Debit Mandates**.

## ADDACS Report Description

Once a **Direct Debit Instruction (Mandate)** has been set up, it might change.

When it does, the **Paying Bank** reports the change to **BACS** (Bankers Automated Clearing Service). BACS generates an ADDACS (Automated Direct Debit Amendment and Cancellation Service) report about the changes that were made and sends it to the insurer.

The following are the **Reasons** which can be received in the **ADDACS** file, and the corresponding behavior of the **Billing and Collection** solution:

Code	Mention
0	<b>Instruction cancelled refer to Payer.</b> If this reason is received, then the mandate status is changed to Cancelled.
1	<b>Instruction cancelled by Payer.</b> If this reason is received, then the mandate status is changed to Cancelled.
2	<b>Payer deceased.</b> If this reason is received, then the mandate status is changed to Cancelled.
3	<b>Account transferred to a new Bank or Building Society</b> If this reason is received, then the mandate status is changed to Cancelled.
B	<b>Account closed.</b> If this reason is received, then the mandate status is changed to Cancelled.
C	<b>Account/ Instruction transferred to a different branch of Bank/ Building Society.</b> If this reason is received, then the mandate status is changed to Cancelled.
D	<b>Advance notice disputed.</b> If this reason is received, then no action is taken.
E	<b>Instruction amended.</b> If this reason is received, then update the mandate.
R	<b>Instruction re-instated.</b> If this reason is received, then reactivate the mandate.

**NOTE**

An **ADDACS report** (file) is produced **whenever** a change is

made to the details of an existing **Direct Debit Instruction** (Mandate).

It is critically important that the correct action is taken in response to an ADDACS report as this will enable the insurer to continue to receive payments. These updates should be acted upon within 3 days of receipt. It is in the best interest of insurers to act quickly on receipt of an ADDACS report, otherwise, it can be risky making incorrect collections and facing indemnity claims, loss of revenue, and damage to organization’s reputation as offering poor customer service.

**ADDACS files are emitted by the BACS** (Bankers Automated Clearing Service) payment network, the most popular method for sending and receiving business payments in the UK.

The **ADDACS Files** section contains all the ADDACS files ever imported in the system. The mandates can be accessed from the ADDACS Mandates Grid even if their effective date is in the future. You can search inside this repository by **File** or **Import Date**. You can inspect a record by double-clicking it.

DIRECT DEBIT NOTIFICATIONS LIST	
File	Import Date
<input type="text" value="Q"/>	<input type="text" value="Q"/>
Mandat ex2 - update D to M.txt	17/09/2021 17:41
Mandat ex2 - update D to M.txt	17/09/2021 16:31
Mandat ex2 - draft update 2.txt	17/09/2021 11:49
Mandat ex2 - draft update.txt	16/09/2021 21:16
Mandat ex2 - draft.txt	16/09/2021 21:14

5 10 20 1 ... 4 5 6 7 ...

**NOTE**

In this section, file import is possible and you use this functionality to upload **ADDACS reports** (files) about the existing mandates, whenever you receive them. A file must have **.csv** or **.txt** format in order to be processed and stored by the system.

Follow the steps below in order to manually import an **ADDACS report**. See right below:

## Import File Instructions

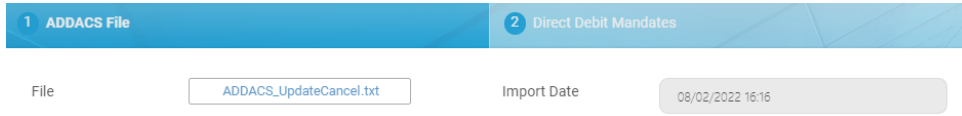
Below is an example of an **Import Form**:

1. While in the same section, click **Insert**, at the top right corner of your screen.
2. When the **Import Form** opens, select your file and then click **Import Data**.
3. (Optional) Check the second tab of the record, for further details. When you upload an ADDACS file, the system automatically parses the data and displays the findings in the second tab of the ADDACS record. See also the next paragraph that explains how the record is organized.
4. Click **Save and close**.

Inside any **ADDACS report** record, the details about the notified mandates are organized as follows:

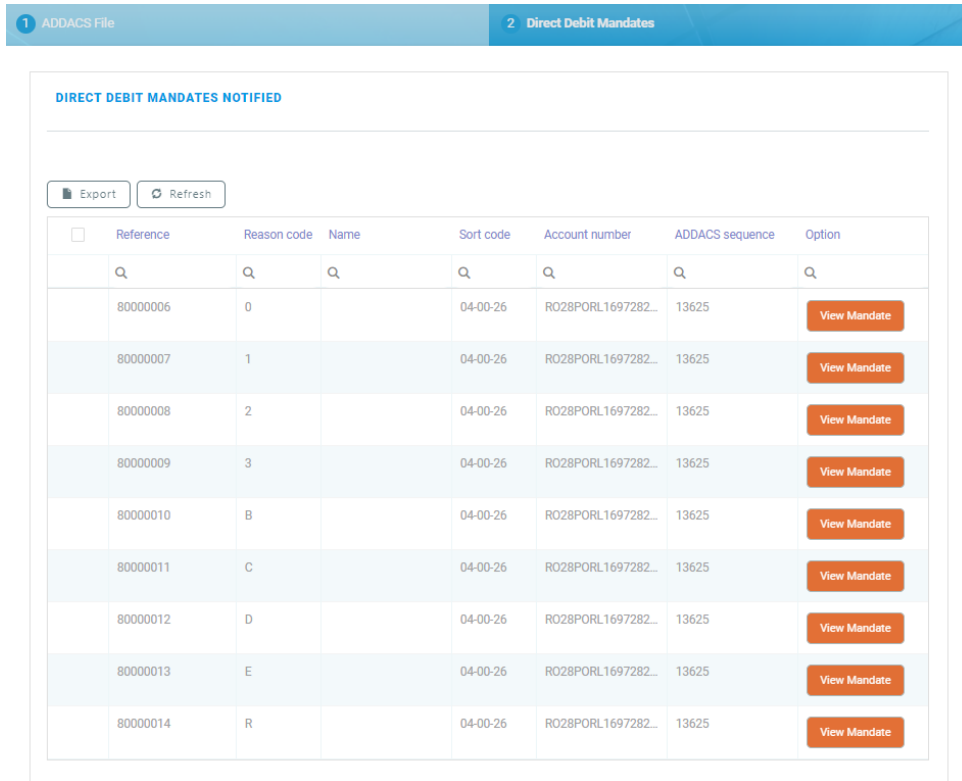
- The **ADDACS File** tab - This first tab includes the actual **File** with the notification (or notifications) for the mandates existing in your system. The **Import Date** field is completed automatically.

Below is an example of the first tab for an ADDACS record:



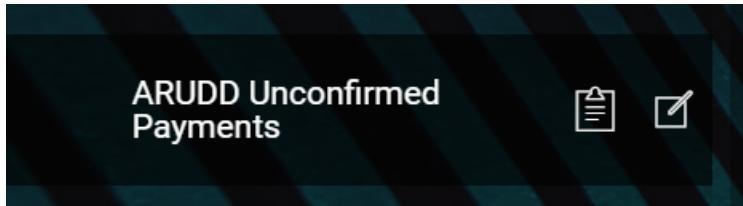
- The **Direct Debit Mandates** tab - This second tab includes a **View Mandate** button which allows you to visualize the actual **mandate record** that was updated. When an ADDACS record cancels more than one mandate, the record points to all those mandates that have the reference specified in the ADDACS file. Accordingly, this form displays all the mandates, in the grid, with view buttons next to every one.

Below is an example of the second tab for an ADDACS record, which informs about different status changes for multiple mandates:





## ARUDD Unconfirmed Payments Section



### HINT

The **ARUDD report (file)** contains data that impact the status of the **payments** for an UK Direct Debit Mandate.

## ARUDD Report Description

Once a Direct Debit Instruction (Mandate) has been set up, it might fail to deliver payments.

When it does, the Paying Bank reports the change to BACS (Bankers Automated Clearing Service). BACS generates an ARUDD (Automated Return of Unpaid Direct Debits) report about the unpaid premiums and sends it to the insurer.

An **ARUDD report (file)** is triggered after you have submitted a payments collection file in an attempt to collect from your Payers. It becomes available to insurers **one day after Direct Debits are due** to be collected. This report contains information about any of the payments that could not be collected.

### NOTE

An **ARUDD report (file)** is produced **whenever** a payment failed, for any of your existing **Direct Debit Instructions (Mandates)**.

**ARUDD files are emitted by the BACS** (Bankers Automated Clearing Service) payment network, the most popular method for sending and receiving business payments in the UK.

**NOTE**

In this section, file import is possible. An **ARUDD File** must have **.txt** format in order to be processed and stored by the system.

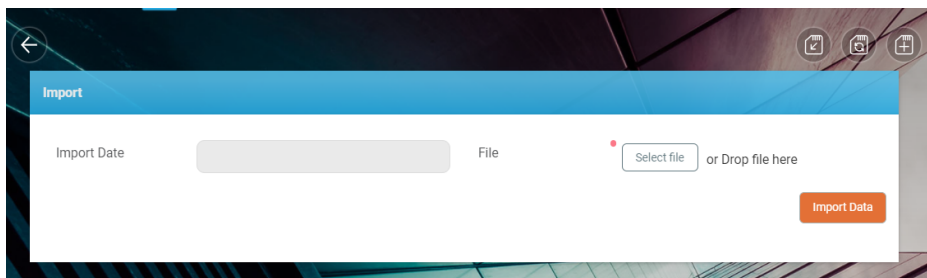
The **ARUDD Unconfirmed Payments** section contains details about all the unconfirmed direct debit payments, registered in the system.

DIRECT DEBIT CONFIRMATIONS LIST	
File	Import Date
<input type="checkbox"/>	
<input type="text" value=""/>	<input type="text" value=""/>
Payments confirmation file.txt	01/10/2021 14:13
Payments confirmation file.txt	01/10/2021 14:11
Payments confirmation file.txt	01/10/2021 14:11
Payments confirmation file.txt	01/10/2021 14:11
Payments confirmation file.txt	01/10/2021 14:07

You can search inside this repository by **File** or **Import Date**. You can inspect a record by double-clicking it. When the case, you can also manually import a **ARUDD** file. See right below:

## Import File Instructions

Below is an example of an **Import Form**:

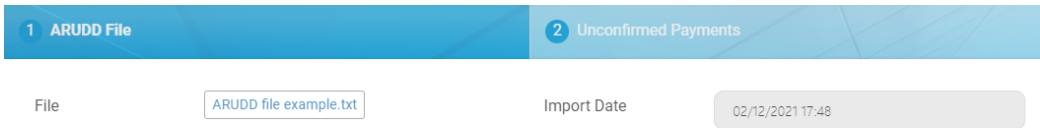


1. While in the same section, click **Insert**, at the top right corner of your screen.

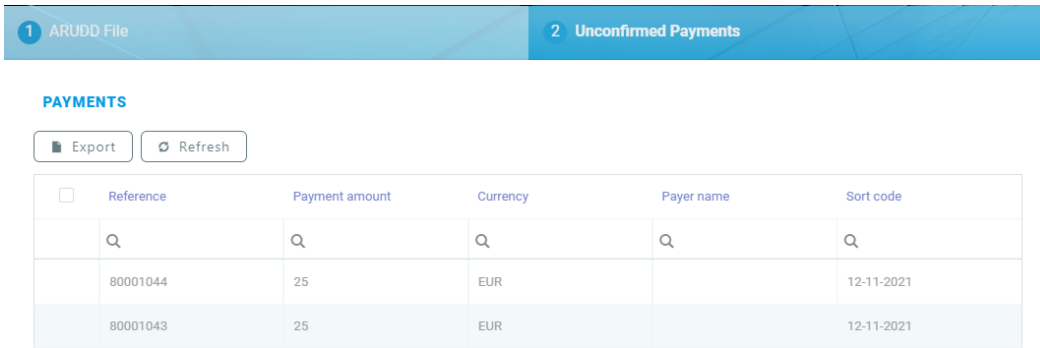
2. When the **Import Form** opens, select your file and then click **Import Data**.
3. (Optional) Check the tabs of the record, for further details. See also the next paragraph that explains how the record is organized.
4. Click **Save and close**.

Inside any **ARUDD File** record, the details about the unconfirmed payments are organized as follows:

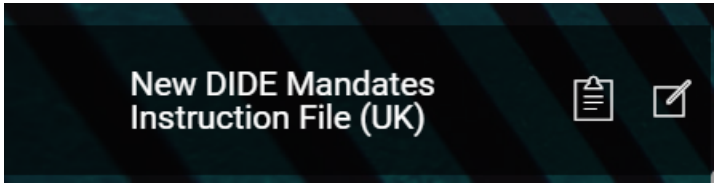
**The first tab** - includes the actual **File** with the unconfirmed payment for a specific mandate and the auto-populated **Import Date** field.



**The second tab** - includes a grid containing every denied payment extracted from the imported file. For the case when there are many records listed, you also can search inside this grid by different keywords - such as **Reference**, **Payment amount**, **Currency**, **Payer name**, and **Sort code**.



## New DIDE Mandates Instruction File Section



**NOTE**  
 All the details from this section are automatically filled in by the system and they are not editable.

For the new direct debit mandate calls, received through the [Generate UK Mandate API](#), there is a daily job - DIDEUK\_Instructions, that is scheduled to generate mandate activation instructions for the BACS system, compliant with the BACS standardized .txt file format. This section holds all the files containing the instructions for the activation of direct debit mandates generated by **Billing and Collection** solution. This is where you go to check how many new mandates were activated.

**DIRECT DEBIT INSTRUCTION FILES LIST**

File name	Date of generation
NEW_DIDE_Mandates_BACSInstructions_20220203.txt	03/02/2022 15:32
NEW_DIDE_Mandates_BACSInstructions_20220203.txt	03/02/2022 12:49
NEW_DIDE_Mandates_BACSInstructions_20220203.txt	03/02/2022 12:46
NEW_DIDE_Mandates_BACSInstructions_20220203.txt	03/02/2022 12:25
NEW_DIDE_Mandates_BACSInstructions_20220201.txt	01/02/2022 17:26

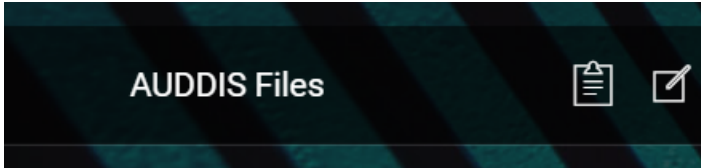
You can search inside this repository by **File name** or **Date of Generation**. You can inspect a record by double-clicking it.

**DIRECT DEBIT INSTRUCTION FILE**

File name:  Date of generation:

If you want to inspect the details inside a particular mandate activation request, you must download the file.

## AUDDIS Files Section



### HINT

The **AUDDIS report (file)** contains data that impact the status of the **payment collection** for an UK Direct Debit Mandate.

## AUDDIS Report Description

Once a policyholder agreed to pay the premiums by direct debit, the insurer must regularly send direct debit payment instructions to BACS. These instructions might not be accepted (either by BACS or the Payer’s bank). When this happens, **BACS** (Bankers Automated Clearing Service) generates an AUDDIS (Automated Direct Debit Instruction Service) report about the failed direct debit instruction for payment and sends it to the insurer.

The following are the **Reasons** which can be received in the **AUDDIS** file:

Reason code	Comments
F	Invalid account type
1	Instruction cancelled by payer
2	Payer deceased
H	Instruction has expired
C	Account transferred
I	Payer Reference is not unique
B	Account closed
L	Incorrect payer’s Account Details
5	No account

Reason code	Comments
K	Instruction cancelled by paying bank
6	No Instruction
G	Bank will not accept Direct Debits on account

Upon importing an **AUDDIS report** in the system, the following is the corresponding behavior of the **Billing and Collection** solution:

Field	Description
Record type	No validations, receive the exact value from the file.
Reference	Policy number.
Reason code	See related table from above.
Payer's name	No validations, receive the exact value from the file.
Sort code	Sort code for the initiated mandate. No validations, receive the exact value from the file.
Account number	Account number correlated with the initiated mandate No validations, receive the exact value from the file.
A/C type	No validations, receive the exact value from the file.
Original proc date	No validations, receive the exact value from the file.
Effective date	Date when the mandate validation decision has been made. No validations, receive the exact value from the file.
Tran code	No validations, receive the exact value from the file.
Originator sort code name	No validations, receive the exact value from the file.
Details account no	No validations, receive the exact value from the file.
Note	No validations, receive the exact value from the file.
ADDACS sequence	No validations, receive the exact value from the file.

**NOTE**

An **AUDDIS report** (file) is produced **whenever** a payment transfer problem appears to an existing **Direct Debit Instruction** (Mandate).

**AUDDIS files are emitted by the BACS** (Bankers Automated Clearing Service) payment network, the most popular method for sending and receiving business payments in the UK.

**NOTE**

In this section, manual file import is possible. An **AUDDIS file** must have **.csv.txt** format in order to be processed and stored by the system.

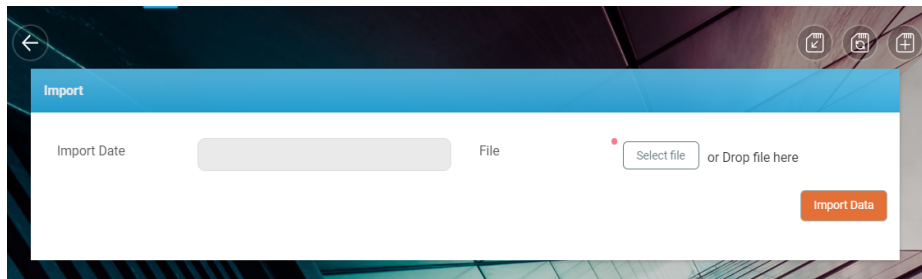
The **AUDDIS Files** section contains details about all the requests for direct debit payment that were denied and are registered into your system. If necessary, you can also manually import **AUDDIS reports**.

AUDDIS FILES LIST	
File name	Import date
AUDDIS_ .txt	04/02/2022 11:06
AUDDIS_ .txt	03/02/2022 18:12
AUDDIS_ .txt	03/02/2022 18:11
AUDDIS_ .txt	03/02/2022 13:08
AUDDIS_ .txt	03/02/2022 12:51

You can search inside this repository by **File name** or **Import Date**. You can inspect a record by double-clicking it. If necessary, you can manually import an **AUDDIS report** file. See right below:

## Import File Instructions

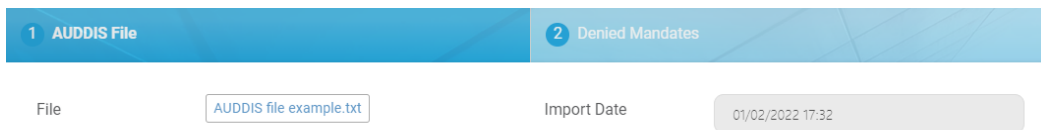
Below is an example of an **Import Form**:



1. While in the same section, click **Insert**, at the top right corner of your screen.
2. When the **Import Form** opens, select your file and then click **Import Data**.
3. (Optional) Check the tabs of the record, for further details. See also the next paragraph that explains how the record is organized.
4. Click **Save and close**.

Inside any **AUDDIS** record, the details about the denied payment are organized as follows:

**The first tab** - includes the actual **File** with the details of the denied request for a specific mandate and the auto-populated **Import Date** field.



**The second tab** - includes a grid containing every denied mandate extracted from the imported file. For the case when there are many records listed, you also can search inside this grid by different keywords - such as **Record type**, **Reference**, **Reason code**, **Payer name**, **Account number**, **Effective date**, and **Option**.



1 AUDDIS File

2 Denied Mandates

**MANDATE DENIALS**

---

Export
 Refresh

<input type="checkbox"/>	Record type	Reference	Reason code	Payer name	Account number	Effectiv date	Option
	<input type="text" value="Q"/>	<input type="text" value="Q"/>	<input type="text" value="Q"/>	<input type="text" value="Q"/>	<input type="text" value="Q"/>	<input type="text" value="Q"/>	<input type="text" value="Q"/>
	N	80002026	xyz	Mihai Gurau	2222222	01.02.2022	<a href="#" style="background-color: #e67e22; color: white; padding: 2px 5px; text-decoration: none;">View Mandate</a>

**HINT**

For more information about the import rules and configurations, consult also the [UK Direct Debit](#) and the [Flow Parameters And Scheduled Jobs](#) pages.

# Configurations

**Billing and Collection** enables you to simplify your billing and collection operations. **Invoicing is automatic only**. **Invoices** are constantly generated into your system for all your insurance products that are in **Active** status.

You can also configure **Billing and Collection** to deal with payments received from different sources or systems - such as online payment processors, bank payment orders, etc. The solution lets you process different types of payments, manually or automatically. For payments that respect some rules, the **allocation flow is also completely automated**.

Check the following pages to find out more about how this solution works:

- [Flow Parameters and Scheduled Jobs](#) - for details about the parameters and jobs that control the behaviors of the **Billing and Collection** solution.
- [Import Bank Statements](#) - for details about how bank statements are imported in the system.
- [Incoming Payments](#) - for general details about how incoming payments are handled by the system, and also:
  - [Invoice Generation](#)
  - [Automatic Allocation](#)
  - [Manual Allocation](#)
  - [SEPA Direct Debit](#)
  - [UK Direct Debit](#)
- [Outgoing Payments](#) - for general details about how outgoing payments are handled by the system, and also:
  - [Outgoing Payments Admin](#)
  - [Outgoing Payment Allocation](#)

- [Manual Outgoing Payment Requests](#)
- [Billing and Collection Endpoints](#) - for details about dedicated APIs.
- [Security Roles](#) - for details about the security roles that come predefined with the solution.
- [Digital Assets](#) - for descriptions of the **Billing and Collection** digital assets.

## Flow Parameters and Scheduled Jobs

The following flow parameters and scheduled jobs are used with the **Billing and Collection** solution.

### 1 Flow Parameters

Parameter Name	Days After Unpaid Due Date
Details	Type: Integer; Code: <b>DAUDD</b>
Parameter name	<b>PayOnTime Retrial Days</b>
Parameter name	<b>FTOS_PYMT_DIDEConfiguration</b>
Parameter name	<b>PAID Payment Config</b>
Parameter name	<b>Days Before Expiration Grace Period</b>
Parameter name	<b>No. of Days Before Return</b>
Parameter name	<b>Generating Statement Days in Advance/ No. of Days in Advance</b>
Parameter name	<b>Write Off Limits</b>
Parameter name	<b>Days before UK Mandates activation</b>

<b>Parameter Name</b>	<b>Days After Unpaid Due Date</b>
Component	<b>Billing and Collection</b>
Correlated with	<b>FTOS_PA_Policy Lapsed</b> scheduled job
Description	This parameter sets the number of days after an unpaid installment’s due date. When the parameter is fulfilled, the policy is automatically moved from <b>InForce</b> to <b>Lapsed</b> status.
<b>Parameter name</b>	<b>PayOnTime Retrial Days</b>
Details	Type: Collection; Code: <b>POTRD</b>
Component	<b>Billing and Collection</b>
Correlated with	<b>GeneratePaymentForInstallments</b> scheduled job
Description	This parameter is used to set the number of days, after the initial due date set for an installment, after which the <b>GeneratePaymentForInstallments</b> scheduled job triggers the <b>PayOnTime</b> retrial process. This being a collection type parameter, you can add or set multiple values for the number of days in which a new payment retrial must be made. Use the parameter form to define a new item in the collection, as needed.
<b>Parameter name</b>	<b>FTOS_PYMT_DIDEConfiguration</b>
Details	Type: JSON; Code: <b>DIDE</b>
Component	<b>Billing and Collection</b>
Correlated with	<b>FTOS_PYMT_DIDEInstructionFile</b> scheduled job
<b>Parameter name</b>	<b>PAID Payment Config</b>
<b>Parameter name</b>	<b>Days Before Expiration Grace Period</b>
<b>Parameter name</b>	<b>No. of Days Before Return</b>
<b>Parameter name</b>	<b>Generating Statement Days in Advance/ No. of Days in Advance</b>
<b>Parameter name</b>	<b>Write Off Limits</b>
<b>Parameter name</b>	<b>Days before UK Mandates activation</b>

Parameter Name	<b>Days After Unpaid Due Date</b>
Parameter name	<b>PayOnTime Retrial Days</b>
Parameter name	<b>FTOS_PYMT_DIDEConfiguration</b>
Description	This parameter sets the <b>DIDE type</b> configuration. In order to set which DIDE type will be applied, a <b>specific key</b> must be set in the Direct Debit processor allowing the user to choose the desired option. If the option configured in the processor is <b>“Sepa”</b> - the DIDE process will take in consideration the functionality implemented for DIDE Sepa. Otherwise, if the value is <b>“UK”</b> , the DIDE process will take into consideration the flow implemented for DIDE UK.
Parameter name	<b>PAID Payment Config</b>
Details	Type: Collection; Code: <b>PAIDPYMT</b>
Component	<b>Billing and Collection</b>
Correlated with	N/A
Description	This parameter sets the details regarding PAD Payments. The parameter contains details about: <ul style="list-style-type: none"> <li>- The commission set for payments,</li> <li>- The beneficiary for payments,</li> <li>- The IBAN where the payments are made,</li> <li>- The bank correlated with the IBAN.</li> </ul>
Parameter name	<b>Days Before Expiration Grace Period</b>
Details	Type: Integer; Code: <b>DBEGP</b>
Component	<b>Billing and Collection, Notifications</b>
Correlated with	<b>FTOS_BC_PaymentUnconfirmedGracePeriod</b> scheduled job
Description	This parameter sets the number of days before the expiration of the grace period for an installment. Also, a <b>Pre-announce lapsing</b> notification is sent to a chosen address, within the number of days set through this parameter.
Parameter name	<b>No. of Days Before Return</b>
Parameter name	<b>Generating Statement Days in Advance/ No. of Days in Advance</b>
Parameter name	<b>Write Off Limits</b>
Parameter name	<b>Days before UK Mandates activation</b>

Parameter Name	Days After Unpaid Due Date
Parameter name	PayOnTime Retrial Days
Parameter name	FTOS_PYMT_DIDEConfiguration
Parameter name	PAID Payment Config
Parameter name	Days Before Expiration Grace Period
Parameter name	No. of Days Before Return
Details	Type: Integer; Code: PRDAY
Component	Billing and Collection
Correlated with	FTOS_PYMT_Payment_ReturnUnallocatedAmount scheduled job
Description	This parameter sets the number of days before automatically generating a <b>Payment Return</b> for a payment which is still in <b>Unallocated</b> status in the system. The value set in this parameter is used by the <b>FTOS_PYMT_Payment_ReturnUnallocatedAmount</b> scheduled job in order to trigger the payment return generation process.
Parameter name	Generating Statement Days in Advance/ No. of Days in Advance
Details	Type: Integer; Code: SGDAY
Component	Billing and Collection
Correlated with	FTOS_PYMT_InsertStatementDueDate scheduled job
Parameter name	Write Off Limits
Parameter name	Days before UK Mandates activation

Parameter Name	Days After Unpaid Due Date
Parameter name	PayOnTime Retrial Days
Parameter name	FTOS_PYMT_DIDEConfiguration
Parameter name	PAID Payment Config
Parameter name	Days Before Expiration Grace Period
Parameter name	No. of Days Before Return
Parameter name	Generating Statement Days in Advance/ No. of Days in Advance
Description	<p>This parameter sets the day for generating the statement (invoice) in advance with a number of days before the payment's due date. This parameter is set for all the invoices to be generated for the installments on a policy. The <b>FTOS_PYMT_InsertStatementDueDate</b> scheduled job that triggers the generation of the invoice is using this parameter.</p> <div style="background-color: #e6f2ff; padding: 10px; border: 1px solid #add8e6;"> <p><b>NOTE</b> The configuration of this parameter is made at the product level, where the user can select that the current product keeps the default parameter value or can be configured with a specific value, as desired.</p> </div>
Parameter name	Write Off Limits
Details	Type: Collection; Code: <b>WO</b>
Component	<b>Billing and Collection</b>
Correlated with	N/A
Parameter name	Days before UK Mandates activation

Parameter Name	Days After Unpaid Due Date
Parameter name	PayOnTime Retrial Days
Parameter name	FTOS_PYMT_DIDEConfiguration
Parameter name	PAID Payment Config
Parameter name	Days Before Expiration Grace Period
Parameter name	No. of Days Before Return
Parameter name	Generating Statement Days in Advance/ No. of Days in Advance
Parameter name	Write Off Limits
Description	<p>This parameter sets the limit values for a <b>Write-Off</b> payment. This kind of payment is generated automatically by the system during the allocation of payments. This parameter can be set for RON and EUR currencies.</p> <div style="background-color: #e6f2ff; padding: 10px; border: 1px solid #add8e6;"> <p><b>NOTE</b> The configuration of this parameter is made at the product level, where the user can select that the current product keeps the default parameter value or can be configured with a specific value, as desired.</p> </div>
Parameter name	Days before UK Mandates activation
Details	Type: int; Code: <b>DBUKMA</b>
Component	<b>Billing and Collection</b>
Correlated with	N/A
Description	<p>This parameter sets the number of days allowed for receiving notifications (e.g. AUDDIS) about the cancellation of a mandate in <b>Pending</b> status. E.g. If this parameter is set to 5 days, the <b>Draft</b> mandate becomes <b>Active</b> in 5 days time, provided the mandate is not included in any imported AUDDIS file (containing a cancellation notification for it) by the 5th day (respectively by the last day of the period configured in the parameter).</p>



## 2 Scheduled Jobs

<b>Job name</b>	<b>GeneratePaymentForInstallments</b>
Scheduled	At 06:32 AM, daily run
Description	<p>This job notifies the system to withdraw a payment amount - through the PayUOnTime process.</p> <p>After the initial due date set for an installment, this job also uses the PayUOnTime retrial days parameter to withdraw the payment amount.</p> <p>For example:</p> <ul style="list-style-type: none"> <li>- First payment retrial – 0 day after installment’s due date</li> <li>- Second payment retrial – 3 days after installment’s due date</li> <li>- Third payment retrial – 11 days after installment’s due date.</li> </ul>
<b>Job name</b>	<b>FTOS_PYMT_Payment_ReturnUnallocatedAmount</b>
Scheduled	At 02:00 AM, daily run
Description	<p>This job creates payment returns for payments which are in <b>Unallocated</b> status for more than the number of days set by the <b>No. of Days Before Return</b> parameter.</p> <p>For example: If a payment is in <b>Unallocated</b> status for more than 5 days, a payment return is automatically generated for this unallocated payment, in the sixth day.</p>
<b>Job name</b>	<b>FTOS_PYMT_InsertStatementDueDate</b>
Scheduled	At 03:00 AM, daily run
<b>Job name</b>	<b>FTOS_PYMT_PaymentReminder</b>
<b>Job name</b>	<b>FTOS_GetExchangeRate_BNR</b>
<b>Job name</b>	<b>FTOS_PYMT_DirectDebitMandateStatus</b>
<b>Job name</b>	<b>FTOS_PYMT_OutgoingPaymentInstructionFile</b>
<b>Job name</b>	<b>FTOS_PYMT_DIDE_ADDACS</b>
<b>Job name</b>	<b>DIDEUK_Instructions</b>
<b>Job name</b>	<b>FTOS_PYMT_DIDEInstructionFile</b>

<b>Job name</b>	<b>GeneratePaymentForInstallments</b>
<b>Job name</b>	<b>FTOS_PYMT_Payment_ReturnUnallocatedAmount</b>
<b>Job name</b>	<b>FTOS_PYMT_InsertStatementDueDate</b>
Description	<p>This job generates an insurer statement (invoice) for policies in <b>Proposal</b>, <b>Issued</b> or <b>InForce</b> status, at each installment’s due date.</p> <div style="background-color: #e6f2ff; padding: 10px; border: 1px solid #add8e6;"> <p><b>NOTE</b></p> <p>Policies in <b>Cancelled</b> status are excluded for new invoice generation.</p> <p>For products on which the <b>Specific SGDAY</b> option is checked, the invoices are generated taking into account the <b>Specific SGDAY</b> chosen value.</p> <p>For products on which the <b>Specific Day of the Month</b> option is checked, and the product becomes active after that specific day of the month, the invoices are generated starting with the next month, on the specified day.</p> </div>
<b>Job name</b>	<b>FTOS_PYMT_PaymentReminder</b>
Scheduled	At 09:00 AM in the working day set by the <b>Payment Reminder</b> parameter
Description	<p>This job sends a payment reminder notification about any installment in <b>Unpaid</b> status to the policyholders.</p> <p>This job runs in the nth working day set in <b>Payment Reminder</b> parameter (number of days after DIDE generation).</p>
<b>Job name</b>	<b>FTOS_GetExchangeRate_BNR</b>
Scheduled	At 03:20 AM, daily run
Description	This job gets the official BNR (National Bank of Romania) Exchange Rate daily values for currencies, in order to be used inside the different <b>Billing and Collection</b> operations.
<b>Job name</b>	<b>FTOS_PYMT_DirectDebitMandateStatus</b>
Scheduled	At 05:00 AM, daily run
Description	For all the mandates registered in the system, the job checks the <b>Begin</b> and <b>End</b> Dates against the current date and it updates the mandate statuses accordingly. For example if the current date is passed the <b>End Date</b> set for a mandate, the mandate status is changed to <b>Expired</b> .
<b>Job name</b>	<b>FTOS_PYMT_OutgoingPaymentInstructionFile</b>
Scheduled	At 05:00 AM, daily run
<b>Job name</b>	<b>FTOS_PYMT_DIDE_ADDACS</b>
<b>Job name</b>	<b>DIDEUK_Instructions</b>
<b>Job name</b>	<b>FTOS_PYMT_DIDEInstructionFile</b>

<b>Job name</b>	<b>GeneratePaymentForInstallments</b>
<b>Job name</b>	<b>FTOS_PYMT_Payment_ReturnUnallocatedAmount</b>
<b>Job name</b>	<b>FTOS_PYMT_InsertStatementDueDate</b>
<b>Job name</b>	<b>FTOS_PYMT_PaymentReminder</b>
<b>Job name</b>	<b>FTOS_GetExchangeRate_BNR</b>
<b>Job name</b>	<b>FTOS_PYMT_DirectDebitMandateStatus</b>
<b>Job name</b>	<b>FTOS_PYMT_OutgoingPaymentInstructionFile</b>
Description	<p>Job purpose: to generate a payment <b>instruction file</b> with all outgoing payments complying with the following conditions:</p> <ul style="list-style-type: none"> <li>• the outgoing payment is in <b>Scheduled</b> status;</li> <li>• the payment date is scheduled for the current date, or before the current date.</li> </ul> <p>File format: .csv with .txt extension; time stamped.                      File structure: Reference No; Beneficiary's Name; Beneficiary's IBAN; Beneficiary's Bank; Payment Amount; Currency; Scheduled Date; Payer's Name; Payer's IBAN; Payer's Bank.</p>
<b>Job name</b>	<b>FTOS_PYMT_DIDE_ADDACS</b>
Scheduled	At 5:15 AM, daily run
Description	This job is scheduled to perform status changes (canceling, modifying or reactivating) on the existing mandates, based on the unprocessed records from the buffer - the <b>FTOS_PYMT_DIDE_ADDACS</b> entity.
<b>Job name</b>	<b>DIDEUK_Instructions</b>
Scheduled	At 04:00 AM, daily run
Description	The job is scheduled to generate mandate activation instructions for the BACS* system, in a standardized <b>.txt</b> file ( <b>DIDEUK_Instructions</b> ) - based on which the system starts the mandate activation procedure. This job is dedicated to the UK direct debit processing, only. It is the job that collects all the <b>Draft</b> mandates and sends instructions for their activation to BACS - in a file that complies with the format requested by BACS.
<b>Job name</b>	<b>FTOS_PYMT_DIDEInstructionFile</b>
Scheduled	At 03:00 AM, daily run
Description	This job uses the <b>FTOS_PYMT_DIDEConfiguration</b> parameter in order to generate the direct debit instruction file in the system. The file is generated in a standardized <b>.txt</b> format, in order to be suitable to the bank that is going to process it. The file contains the instructions for direct debit billings for an active mandate - based on which the bank transfers the installment amount from the insured into insurer's account. Depending on the configuration, this job generates the <b>DIDEPaymentsInstructionFile</b> (for SEPA) and <b>DD_BACSPayments</b> (for UK), also.

\*BACS Payment Schemes Limited, previously known as Bankers' Automated Clearing System, is responsible for the clearing and settlement of UK automated direct debit and BACS Direct Credit and the provision of third-party services.

## Import Bank Statements

The **Import Bank Statements** functionality lets you import bank statement files for payment processing. In **FintechOS Portal**, select **Billing and Collection** and then select **Bank Statements**. On the **Bank Statements List** page, click the **Insert** button to load a new bank account statement. The **Insert** triggers the import of the bank file in the system and changes the business status of the record to **Imported**. This functionality also allows for import of payment files from online payment processors.

The journeys, entities, libraries, and endpoints related to the **Import Bank Statements** functionality are as follows:

### Data model

Entity **FTOS\_PYMT\_PaymentGroup** with all attributes.

### Payment Group Insert Journey

General description:

This is a user journey aimed at implementing the **Insert Bank Statement** functionality. The journey is marked as **Default** for insert on **FTOS\_PYMT\_PaymentGroup** entity.

After selecting the file to import, when pressing **Import Data**, the record is saved - using the `ebs.saveEditForm` method, from the **FintechOS** Client Side SDK. Next, the flow is redirected to the default form driven flow of **FTOS\_PYMT\_PaymentGroup** entity - using the `ebs.goToUrl` method from **FintechOS** Client Side SDK.

### Payment Group Journey

General description:

This is a user journey aimed at implementing the payment group functionality. The journey is marked as **Default** for insert on **FTOS\_PYMT\_PaymentGroup** entity. This journey has three steps:

1. **Bank Statement** - Processing the information from the bank statement.
2. **Payments** - Making the payment.
3. **GL** - Creating the general ledger record for the payment.

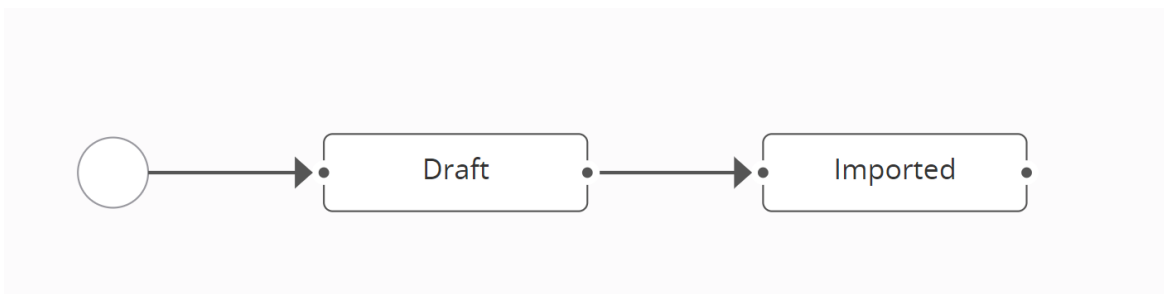
The journey uses the `initiateTotals` function, which collects all the unallocated payments - `paymentOrder`, `paymentExternal`, `brokerPremiumPayment` and `outgoing` - and shows them to the **Payments** and **Outgoing Payments** grids.

**Input** parameters: `paymentGroupId` - The Id of the payment group.

**Output** Parameters: N/A.

## Business Workflow Configurations Actions

Below is the diagram with the statuses and transitions managed through the **Import Bank Statements** functionality.



Business Workflow Transitions:

Status	Description
_Draft	Initial status
Draft_Imported	From <b>Draft</b> to <b>Imported</b> - final status. This transition is manual and it happens when the user presses the <b>Import</b> button.

## Server Side Script Libraries

### FTOS\_PYMT\_PaymentGroup\_Library

From this library, the **General** function is used. This function includes the following functions:

#### getAllocatedPayments

This function gets all the allocated payments from a specific bank statement, based on Id.

**Input** parameters: `paymentGroupId` - The Id of the payment group.

**Output** parameters: Returns the results of the fetch.

#### paymentGroupResult

Based on the result of the **getAllocatedPayments** function, this function throws an error if a bank statement has an assigned payment.

**Input** parameters: `paymentGroupId` - The Id of the payment group.

**Output** parameters: **throwException** - If conditions are met.

### FTOS\_ImportPaymentFiles

From the **FTOS\_ImportPaymentFiles** library, the following functions are used:

#### BRD

Function used to process standardized bank statement files (complying with the **MT940** format) for payment accounts with **BRD code**.

Inside the `BRD` function, the following functions are used:

## getPaymentGroupDetails

This function gets details about the payment group - `paymentAccountId`, `file`, `FTOS_PYMT_PaymentGroupid`, `paymentGroupId`, `paymentGroupId`.

**Input** parameters: `paymentGroupId` - The Id of the payment group.

**Output** parameters: `pg` - The result of the query, which returns the details about the payment group.

## getImportTemplateByName

This function gets the import template.

**Input** parameters: `templateName` - The name of the template.

**Output** parameters: `t` - The result of the query made from the data import entity.

## updateOldImportedValues

This function updates the `lastImport` attribute of previous payments in `FTOS_PYMT_Payment_BRD` entity to false.

**Input** parameters: N/A.

**Output** parameters: N/A.

## getPaymentAccountDetails

This function gets the details about the payment account.

**Input** parameters: `paymentAccountId` - The Id of the payment account.

**Output** parameters: `pa` - The details about the payment account, returned by the query.

## getUnprocessedBRDPayments

This function gets the unprocessed payments.

**Input** parameters: `paymentGroupId` - The Id of the payment group.

**Output** parameters: `initialPayments` - The result of the query containing all unprocessed BRD payments.

## generateDetailsJSON

This function generates an object containing payment details.

**Input** parameters: `brdPaymentDetails`

**Output** parameters: `returnObj` - The object containing payment details.

## processBRDPayments

This function executes the following:

- selects the payments by group id, with the details about the unprocessed payments;
- calls the `FTOS_PYMT_PaymentAllocation` library to allocate payment;
- updates the `isProcessed` flag attribute to true, to mark the payment as processed;
- changes the business status to **Imported**.

**Input** parameters:



- `paymentGroupId` - The Id of the payment group.
- `paymentAccountId` - The Id of the payment account.

**Output** parameters: N/A.

## PAYU

Function used to process standardized statement files from PayU.

Inside the `PAYU` function, the following functions are used:

### getImportTemplateByName

This function gets the import template.

**Input** parameters: `templateName` - The name of the template.

**Output** parameters: `t` - The result of the query made from the data import entity.

### getPaymentDetails

This function gets payment details - `file`, `paymentAmount`, `currencyId`, `businessStatusId`.

**Input** parameters: `paymentId` - The unique identifier of the payment.

**Output** parameters: `pa` - The details about the payment, returned by the query.

### deleteRevenuesByPaymentId

This function deletes revenue records, by id, from the `FTOS_PYMT_ExternalPaymentRevenue` entity.

**Input** parameters: `paymentId` - The unique identifier of the payment.

**Output** parameters: N/A.

## importFile

This function imports a **PayU** file.

**Input** parameters:

- `fileAttr` - The file.
- `additionalValues` - The additional values.
- `templateId` - The Id of the template.

**Output** parameters: N/A.

## ExternalRevenue

Inside the `ExternalRevenue` function, the following functions are used:

### getStatementByReference

This function gets the statement information.

**Input** parameters: `reference` - The reference for the invoice.

**Output** parameters:

- `rez` - The result of the query.
- `null` - In case the query result is empty.

### getConfirmedPaymentsByStatementId

The function gets the payments in **Confirmed** status.

**Input** parameters: `statementId` - The statement identifier.

**Output** parameters: `fetchResult` - The result of the query.

## getCurrencyDetailsById

The function gets the currency code.

**Input** parameters: `currencyId` - The currency identifier.

**Output** parameters:

- `rez` - The result of the query.
- `null` - In case the currency does not exist.

## getTotalsBdx

The function gets the total amount on external payment revenues.

**Input** parameters: `paymentId` - The payment unique identifier.

**Output** parameters:

- `sum` - The total amount on external payment revenues.
- `0` - If there are no external payment revenues.

## getCurrencyByCode

The function gets the currency Id by currency code.

**Input** parameters: `code` - The currency code.

**Output** parameters:

- `rez` - The result of the query.
- `null` - If there is no currency.

## getExternalRevenueByPaymentId

The function gets details about the external payment revenue.

**Input** parameters: `paymentId` - The payment identifier.

**Output** parameters: `rez` - The details returned by query.

## validateExternalRevenueData

The function checks external revenue data.

**Input** parameters: `paymentId` - The payment identifier.

**Output** parameters: `bdxOk` - `true/ false`.

## checkReconciliationProcess

The function checks the reconciliation process by subtracting the `paymentAmount` from the `totalAmount`.

**Input** parameters:

- `paymentId` - The payment identifier.
- `paymentDetails` - The payment details.

**Output** parameters: `true/ false`.

## validatePayUPayment

The function performs the PayU payment validation. Next, it changes the business status to **Closed**.

**Input** parameters: `paymentId` - The payment identifier.

**Output** parameters: N/A.

## isReadyToValidate

The function checks whether the payment is already closed. Next, it verifies if there are unmatched records or reconciliation errors.

**Input** parameters: `paymentId` - The payment identifier.

**Output** parameters: N/A.

## MT

Inside the `MT` function we have the following functions:

### getFileStr

Inside this function, the `getTextFileReader` function is used to store the content of a file.

**Input** parameters: `fileRealName` - The Id of the file.

**Output** parameters: `content` - Returns the file content in string format.

### normalizeStr

The function normalizes strings.

**Input** parameters:

- `str` - The string.
- `splitter` - A splitter.
- `code` - The code.

**Output** parameters: `items` - The normalized items.

## setItemObj

The function reads the imported file.

**Input** parameters:

- `itemStr`
- `code`

**Output** parameters: `item` - The idem found.

## processMTItems

The function processes the **MT940** document.

**Input** parameters:

- `items` - The items form the file.
- `code`

**Output** parameters: `true/ false`

## generateDetailsMTJSON

The function gets the payment details for the incoming payments.

**Input** parameters:

- `paymentDetails` - The details of the payment.
- `broker` - The broker details.

**Output** parameters: `returnObj` - Returns an object with the payment details.

## generateDetailsMTJSONOutput

The function gets the payment details for the outgoing payments.

**Input** parameters: `paymentDetails` - The details of the payment.

**Output** parameters: `returnObj` - Returns an object with the payment details.

## getReferenceFromStr

The function gets reference from a string.

**Input** parameters: `str` - The string containing the invoice reference.

**Output** parameters: `reference` - The reference for the invoice.

## getPolicyNoFromStr

The function gets the policy number from a string.

**Input** parameters: `str` - The string containing the number of the policy.

**Output** parameters: `policyNo` - The policy number.

## processMTPayments

This function calls the `FTOS_PYMT_PaymentAllocation` library in order to automatically allocate payments.

**Input** parameters:

- `paymentGroupId` - The Id of the payment group.
- `paymentAccountId` - The Id of the payment account.
- `code` - The code of the payment.

**Output** parameters: N/A.

## getBankId

This function gets the bankId from the IBAN.

**Input** parameters: `iban` - The IBAN of the payment.

**Output** parameters: `bankId` - The Id of the bank.

## isPayerNameInAccount

This function gets the payer.

**Input** parameters: `name` - The name of the payer.

**Output** parameters: `accountId` - Returns the id of the account associated with the input name, if it exists. If not, the output is **null**.

## getBrokerId

This function gets the identifier for the broker.

**Input** parameters: `iban` - The IBAN of the broker.



**Output** parameters: `brokerId` - Returns the Id of the broker account associated with the input value, if it exists. If not, the output is null.

## getUnprocessedMTPayments

This function gets the unprocessed payments.

**Input** parameters:

- `paymentGroupId` - The Id of the payment group.
- `code` - The code of the payment.

**Output** parameters: `initialPayments` - The previous payments.

## getPaymentGroupDetails

This function gets the payment group details- `paymentAccountId`, `file`, `FTOS_PYMT_PaymentGroupid`, `paymentGroupId`, `paymentGroupId`.

**Input** parameters: `paymentGroupId` - The Id of the payment group.

**Output** parameters: `pg` - The result of the query, which returns the details about the payment group.

## getPaymentAccountDetails

This function gets the payment account details.

**Input** parameters: `paymentAccountId` - The Id of the payment account.

**Output** parameters: `pa` - The details about the payment account, returned by the query.

# ING

Function used to process standardized bank statement files (complying with the **MT940** format) for payment accounts with **ING code**.

Inside the **ING** function, the following functions are used:

## getTransactions

This function gets the transactions details.

**Input** parameters: **itemStr** - The string containing the payment details.

**Output** parameters: **transactions** - The transactions from the bank statement.

## setTransactionLineItems

This function gets the transactions lines.

**Input** parameters:

- **insertINGItemTransaction** - The transaction details.
- **transactionLineStr** - The string containing the line of the transaction.
- **currency** - The currency of the payment.

**Output** parameters: N/A.

## setTransactionDescription

This function defines the transaction details.

**Input** parameters:

- `insertINGItemTransaction` - The transaction details.
- `transactionDescription` - The description of the transaction.

**Output** parameters: N/A.

## BRDMT

Function used to process standardized bank statement files (complying with the **MT940** format) for payment accounts with **BRDMT code**.

Inside the `BRDMT` function, the following functions are used:

### function `getTransactions(itemStr)`

This function gets the transactions from a string.

**Input** parameters: `itemStr` - The string containing the payment details.

**Output** parameters: `transactions` - The transactions from the bank statement.

## `setTransactionLineItems`

This function inserts the transaction lines.

**Input** parameters:

- `insertBRDMTItemTransaction` - The transaction details.
- `transactionLineStr` - The string containing the line of the transaction.
- `currency` - The currency of the payment.

**Output** parameters: N/A.

## setTransactionDescription

This function defines the transaction details.

**Input** parameters:

- `insertBRDMTItemTransaction` - The transaction details.
- `transactionDescription` - The description of the transaction.

**Output** parameters: N/A.

## ABNMT

Function used to process standardized bank statement files (complying with the **MT940** format) for payment accounts with **ABNMT code**.

Inside the `ABNMT` function, the following functions are used:

### function getTransactions(itemStr)

This function gets the transactions from a string.

**Input** parameters: `itemStr` - The string containing the payment details.

**Output** parameters: `transactions` - The transactions from the bank statement.

## setTransactionLineItems

This function inserts the transaction lines.

**Input** parameters:

- `insertBRDMTItemTransaction` - The transaction details.
- `transactionLineStr` - The string containing the line of the transaction.
- `currency` - The currency of the payment.

**Output** parameters: N/A.

## setTransactionDescription

This function defines the transaction details.

**Input** parameters:

- `insertBRDMTItemTransaction` - The transaction details.
- `transactionDescription` - The description of the transaction.

**Output** parameters: N/A.

# Incoming Payments

Incoming payments are received by the insurer from the policyholder as payment for policy coverage. With **Billing and Collection**, the data about the incoming payments is processed and matched (mainly automatically) to a policyholder's account, in order to label an installment as being paid. When an invoice opened for an installment is closed, the details about the payment and the clearing date are updated and stored in the system.

The **Billing and Collection** solution can be used in conjunction with different types of incoming payments such as bank payment orders, direct debit payments, credit card payments, payments made through online processors.

See more details about handling **Incoming Payments** in the configuration pages:

- [Invoice Generation](#) - for details about how invoices are generated in the system.
- [Direct Debit SEPA](#) - for details about how SEPA direct debit mandates are handled by the system.
- [Direct Debit UK](#) - for details about how UK direct debit mandates are handled by the system.
- [Manual Allocation](#) - for details about how the manual allocation for incoming payments works.
- [Automatic Allocation](#) - for details about how the automated allocation for incoming payments works.

## Invoice Generation

### Scheduled Job

The **FTOS\_PYMT\_InsertStatement** invoice statement generation job is scheduled to run every day at 3:00 AM. This job finds the installments that are eligible for the generation of new statements. This job calls the **FTOS\_PYMT\_InsertStatementDueDate** endpoint.

### Server Automation Scripts

The **FTOS\_PYMT\_InsertStatementDueDate** server side script uses the following functions:

## insertStatement

Example of calling the function:

```
1 | insertStatement(maxDate, null)
```

Based on the imported library, `insertStatement(maxDate, null)` is the function that handles the generation of statements.

**Input** parameters:

- `maxDate`: It is obtained by adjoining the `invariantDate` of the current date to the `daysBeforePolicyScheduleDueDate` parameter. This variable represents the maximum date until the policies are filtered based on installments.
- `null`: `accountId`.

**Output** parameters: N/A.

**Variables:**

- `libFlowParameter`: import from the `FTOS_PA_FlowParameter` library.
- `daysBeforePolicyScheduleDueDate`: based on the imported library, through the `getNoOfDaysInAdvance` function, this variable receives an integer value that is set as the SGDAY flow parameter .
- `statementGenerationLib`: import from the `FTOS_PYMT_Statements` library.

### Server Automation Script Libraries

From the `FTOS_PYMT_Statements` server automation script library, the following functions are used:

## getStatementGenerationRules

This function gets a JSON object from the `FTOS_DFP_ProcessorSettings` entity. The object contains the settings for statement processing and has the following structure:

```
1 | {
```

```

2   "StatementsRules": {
3     "rulesArr": [
4       "multiplePolicies",
5       "singlePolicies",
6       "multiplePoliciesByQuoteMultipleNext",
7       "multipleFirstPoliciesByQuoteSingleNext"
8     ],
9     "OP": 0,
10    "PayU": 0,
11    "PayU-on Time": 1,
12    "brokerCollection": 0
13  }
14 }

```

## Object.prototype.getKey(value)

Example of calling the function:

```
1 | statementRules.getKey(k)
```

This function returns an array of payment types based on each **Rule Name Key**, from the **FTOS\_PYMT\_StatementProcessor** parameter.

This function returns an array of payment types based on each **Rule Name Key** for processing statements, from the **FTOS\_DFP\_ProcessorSettings** entity.

The function is used for filtering installments by payment types based on rule names (**rulesArr**).

**Input** parameters: **value** - The rule name key. For example: 0, 1, 2, ...

**Output** parameters: An **array** of strings with the **Payment Types**. For example:

- For **multiplePolicies**, an output example is ["OP", "PayU", "brokerCollection"].
- For **singlePolicies**, an output example is ["PayU-on Time"].
- For **multiplePoliciesByQuoteMultipleNext** and **multipleFirstPoliciesByQuoteSingleNext**, an output example is an empty array [].

## getLastMonth



Example of calling the function:

```
1 | getLastMonth
```

This function returns an object with invariant dates: the `firstDay` of last month and the `lastDay` of last month, in order to help filtering the installments with `brokerCollection` Payment Type .

**Input** parameters: N/A,

**Output** parameters: An object that contains the invariant values as following:

```
1 | lastMonthObj = {
2 |     "firstDay": firstDay,
3 |     "lastDay": lastDay
4 | }
```

## getMonthId

Example of calling the function:

```
1 | getMonthId(month)
```

This function gets the option set item Id from the months option set, based on the month number and, next, returns an object with the month Id and month name.

**Input** parameters: `monthNo` - The month number; it can take values from 1 to 12.

**Output** parameters: An object that contains `monthId` and `monthName`.

## getStatementMonthId

Example of calling the function:

```
1 | getStatementMonthId(invariantDate)
```

This function returns the `FTOS_PYMT_StatementMonthid` primary key attribute, from the `FTOS_PYMT_StatementMonth` entity, based on a query

that searches for the `monthId` and `year`. If the fetch is empty, this function inserts the current search values into the `FTOS_PYMT_StatementMonth` entity.

**Input** parameters: `maxDate` - The invariant date for the statement.

**Output** parameters: `statementMonthId` - The Id of the statement month.

## compareObjects

Example of calling the function:  
 1 | `compareObjects(object1, object2)`

This function compares objects.

**Input** parameters:

`object1` - An object containing the specified details.

`object2` - An object containing the specified details.

**Output** parameters: boolean.

## compareArrays

Example of calling the function:  
 1 | `compareArrays(a, b)`

This function compares arrays.

**Input** parameters:

- `a` - An array containing the specified data.
- `b` - An array containing the specified data.

**Output** parameters: boolean.

## getPosition

Example of calling the function:  
 1 | `getPosition(elem, arrEleme)`

This function gets the position of a specified array element from inside the array.

**Input** parameters:

- `elem` - The element to compare.
- `arrEleme` - The array.

**Output** parameters: The **element position**, if found - else, `null`.

## getGenericProdPosition

Example of calling the function:

```
1 | getGenericProdPosition(arr, elem)
```

This function gets the position of an array element from an array object.

**Input** parameters:

- `elem` - The element to compare.
- `arr` - The array.

**Output** parameters: The **element position**, if found - else, `null`.

## getMaxDate

Example of calling the function:

```
1 | getMaxDate(days)
```

This function gets the maximum date for the payment - such as the SGDAY or the last day of the month. The maximum date variable represents the maximum date until the policies are filtered based on installments.

**Input** parameters: `days` - The maximum date for a payment.

**Output** parameters: `InvariantDate` - The invariant date.

## compareProdDateInArr

Example of calling the function:

```
1 | compareProdDateInArr(arr, day)
```

This function compares dates in an array of dates.

**Input** parameters:

- `arr` - The array of the specified dates.
- `day` - The invariant date.

**Output** parameters: The product `Id`, if the days match - else, it returns `false`.

## compareProdDate

Example of calling the function:

```
1 | compareProdDate(day1, day2)
```

This function compares dates.

**Input** parameters:

- `day1` - The invariant date.
- `day2` - The invariant date.

**Output** parameters:

The product `Id`, if the days match - else, it returns `false`.

## getProducts

Example of calling the function:

```
1 | getProducts(sgDay)
```

This function gets all the products that are in **VWDraft**, **VWVersion Draft** and **VWVersion Unapproved** status and also with the `writeOffId` attribute set to **NoWriteOff**, **GenericWriteOff**, or **SpecificWriteOff**, at the product level.

Next, it filters all found products by start date and end date.

**Input** parameters: `sgDay` - The SGDAY parameter (integer) from the processor.

**Output** parameters: `obj` - An object containing the requested data.

## groupedSpecificProducts

Example of calling the function:

```
1 | groupedSpecificProducts(productsArr)
```

This function filters for products with **Specific WriteOff** settings, based on their `writeOffId` attribute.

**Input** parameters: `productsArr` - An array of products.

**Output** parameters: `productsResultArr` - An array of products that satisfy the request.

## generateGroupedProducts

Example of calling the function:

```
1 | generateGroupedProducts(resultGroupedProducts)
```

This function gets all products filtered by date and **WriteOff** type.

**Input** parameters: `resultGroupedProducts` - An array containing the products.

**Output** parameters: `resultGroupedProducts` - An array containing the products found by the fetch.

## insertStatement

Example of calling the function:

```
1 | insertStatement(invariantDate, accountId)
```

This function prepares the rules for generating the invoices, by using a loop of conditions.

Based on the chosen parameter, from the **FTOS\_PYMT\_StatementProcessor** entity, the function can trigger four types of statement generation flows:

- **multiplePolicies**: where multiple policies are included, based on the Contractor, Currency, Payment Type, Product and Due Date;
- **singlePolicies**: where a single policy is included;
- **multiplePoliciesByQuoteMultipleNext**: for **first installments** - where multiple policies are generated by the same quote or for **next installments** - where multiple policies are included, based on the Contractor, Currency, Payment Type and Due Date;
- **multipleFirstPoliciesByQuoteSingleNext**: for **first installments** - where multiple policies are generated by the same quote or for **next installments** - where a single policy is included.

**Input** parameters:

- **maxDate**: The invariant date values for the statement.
- **accountId**: Null.

**Output** parameters: N/A.

**Variables**:

- **rulesArr** - An array of rule names. For example:  
[multiplePolicies, singlePolicies, multiplePoliciesByQuoteMultipleNext, multipleFirstPoliciesByQuoteSingleNext]
- **ruleName** - A string, for each rule name. For example:  
multiplePolicies.
- **groupByPolicy** - A boolean (true, false) value, used to group by policy.

- `groupByQuote` - A boolean (true, false) value, used to group by quote.
- `paymentTypeArr` - An array of payment types. For example: [OP, PayU, brokerCollection].
- `groupedProducts` - An array of products.

## insertStatementByPaymentType

Example of calling the function:

```
1 insertStatementByPaymentType(invariantDate, accountId,
  ruleName, groupByPolicy, groupByQuote, paymentTypeArr, 1)
```

**Input** parameters:

- `invariantDate` - The invariant date of the statement.
- `accountId` - The Id of the account.
- `ruleName` - The name of the rule.
- `groupByPolicy` - The boolean for grouping by policy.
- `groupByQuote` - The boolean for grouping by quote.
- `paymentTypeArr` - The array containing the payment types.
- `installmentNo` - The installment number variable. It can take the following values:
  - `null` - for a filter without installment number,
  - `1` - for the first installment,
  - `2` - for all other installments except the first.

**Output** parameters: N/A.

**Variables:**

- `currentDate` - The current day, with the following hour format 00:00:00.000. This variable is used in statement generation for filling in the `installmentsStartDate` and `statementDate` attributes.
- `lastMonth` - An object containing the first day of last month and the last day of last month, returned by the `getLastMonth` function.
- `statementMonthId` - The `statementMonthid` attribute from **FTOS\_PYMT\_StatementMonth** entity, returned by the `getStatementMonthId(invariantDate)` function. This variable is used in statement generation for filling in the `statementMonthId` attribute.
- `libParameter` - For importing the **FTOS\_PA\_FlowParameter** library.
- `brokerScope` - based on the **FTOS\_PA\_FlowParameter** imported library, this variable receives a string value from the option set item that is set as the `BKSCOPE` flow parameter.
- `businessStatusPolicyProposal` - The Id of **Proposal** status from the **FTOS\_INSPA\_Policy** entity.
- `businessStatusPolicyIssued` - The Id of **Issued** status from the **FTOS\_INSPA\_Policy** entity.
- `businessStatusPolicyEnforced` - The Id of **Enforced** status from the **FTOS\_INSPA\_Policy** entity.
- `businessStatusPolicySuspended` - The Id of **Suspended** status from the **FTOS\_INSPA\_Policy** entity.
- `policyPaymentTypeOPOptionSetValue` - The Id of the **OP** option set item from the **FTOS\_INSPA\_PolicyPaymentType** option set.
- `policyPaymentTypePayUOptionSetValue` - The Id of the **PayU** option set item from the **FTOS\_INSPA\_PolicyPaymentType** option set.
- `policyPaymentTypePayUOnTimeOptionSetValue` - The Id of the **PayU-on time** option set item from the **FTOS\_INSPA\_PolicyPaymentType** option set.
- `policyPaymentTypeBrokerCollection` - The Id of the **brokerCollection** option set item from the **FTOS\_INSPA\_PolicyPaymentType** option set.



- fetchInstallmentGroup** - This fetch executes the following actions:

Gets all the payment schedules (installments) from **FTOS\_INSQB\_PaymentSchedule** entity, for policies that have currency - **currencyId** is not null.

Adds attributes needed for filtering from the **FTOS\_INSQB\_PaymentScheduleDetail** - the installments entity and also from the **FTOS\_INSPA\_Policy** entity.

Calculates the sum out of the following installments attributes: **installmentAmount**, **paidAmount**, **commissionAmount**, **taxAmount**, **netPremium**, from the **FTOS\_INSQB\_PaymentScheduleDetail** entity, based on the rules conditions.
- fetchInstallments** - This fetch executes the following:

Gets each filtered payment schedule found in the **fetchInstallmentGroup** fetch.

Generates the statement detail according to rules and inserts the required values in the **FTOS\_PYMT\_StatementDetail** entity.

Calculates and updates the statement **DueDate** attribute from the **FTOS\_INSQB\_PaymentScheduleDetail** entity.

Changes the installment status in **StatementIssued**.

### Filtering Configurations

The following are examples about how to define the filtering for the fetch needed to generate statements:

## Filtering with fetchInstallmentGroup

For the **fetchInstallmentGroup** function, you can define the filtering for the fetch based on the following attributes:

Attributes	Conditions	Entity
accountId	Is not null.	FTOS_INSPA_Policy
businessStatusId	Is in <b>Proposal</b> , <b>Enforced</b> , <b>Issued</b> or <b>Suspended</b> status.	FTOS_INSPA_Policy
currencyId	Is not null.	FTOS_INSQB_PaymentSchedule

Attributes	Conditions	Entity
dueDate	Is not null.	FTOS_INSQB_ PaymentScheduleDetail
hasStatementDetail	Is null or is equal with false.	FTOS_INSQB_ PaymentScheduleDetail
installmentAmount	Is grater than or equal with 0.	FTOS_INSQB_ PaymentScheduleDetail
installmentId	Is in <b>On time</b> status.	FTOS_PYMT_ StatementDetail
insuranceProductId	Is not null.	FTOS_INSPA_Policy
policyPaymentTypeId	Is not an empty array. Can take one of the following values: <b>Bank Transfer, PayU, PayU-On Time or Broker Collection.</b>	FTOS_INSPA_Policy

The following are the types of push rules for fetching the needed data for generating statements:

Fetch Type	Specific Conditions	Description
multiplePolicies	insuranceBroker is null	The statement is generated based on the <b>accountId, currency, payment type</b> and <b>due date</b> . Do not apply additional conditions.
singlePolicies	group by policy	The statement is generated based on the <b>FTOS_INSPA_Policyid</b> attribute.
multiplePoliciesByQuoteMultipleNext - first installment	group by quote number	The statement is generated for the multiple policies logic based on the <b>quote number</b> .

Fetch Type	Specific Conditions	Description
multiplePoliciesByQuoteMultipleNext - next installments	insuranceBroker is null	The statement is generated based on the <b>accountId, currency, payment type</b> and <b>due date</b> .
multipleFirstPoliciesByQuoteSingleNext - first installment	group by quote number	The statement is generated for the single policies logic based on the <b>quote number</b> .
multipleFirstPoliciesByQuoteSingleNext - next installments	group by policy	The statement is generated based on the <b>FTOS_INSPA_Policyid</b> attribute.
brokerConditionRules.notBrokerCollectionRules	InsuranceBrokerId is not null. The Policy Payment Type is: <b>Bank Transfer, PayU or PayU-On Time</b>	The statement is generated for a Broker policy based on the <b>InsuranceBrokerId</b> attribute.
brokerConditionRules.BrokerCollectionRules	InsuranceBrokerId is not null. Due Date is in the previous month. Policy Payment Type takes the <b>Broker Collection</b> value.	The statement is generated for a Broker policy based on the <b>InsuranceBrokerId</b> attribute, on the date set by the Broker_Billing_Day_Statement_Generation parameter.

## Filtering for Broker Policies - Example

The following is an example of an object aimed at filtering for **Broker** policies that do not have the Payment Type set to **Broker Collection**:

```

1  if (brokerConditionRulesObj.notBrokerCollectionRules ==
    true) {
2      fetchInstallmentGroup.entity.join
    [0].entity.attributelist.push({
3          "name": "dueDate"
4      });
5      fetchInstallmentGroup.entity.join[0].entity.join
    [1].entity.attributelist.push({
6          "name": "AccountId"
7      });
8
9
10     if
    (brokerConditionRulesObj.brokerScopeDueInstallments =
    true) {
11         fetchInstallmentGroup.where.expressionlist.push
    ({
12             "type": "and",
13             "conditionlist": [{
14                 "first": "e.policyPaymentTypeId",
15                 "type": "notequals",
16                 "second": "val(" +
    policyPaymentTypeBrokerCollection + ")"
17             }]
18         });
19     }
20
21     fetchInstallmentGroup.where.expressionlist.push({
22         "type": "and",
23         "conditionlist": [{
24             "first": "b.dueDate",
25             "type": "lte",
26             "second": "val(" + invariantDate + ")"
27         }]
28     });
29 }
30 if (brokerConditionRulesObj.pushInsuranceBroker == true)
    {
31     fetchInstallmentGroup.entity.join[0].entity.join
    [1].entity.attributelist.push({
32         "name": "insuranceBrokerId"
33     });
34 }

```

After defining the structure of the fetch object, use it as a parameter for the function responsible for getting the results. Next, store the result in a variable:

```
1 | var installmentGroupResult = getByQuery
   | (fetchInstallmentGroup)
```

For each installment of `installmentGroupResult`, take the values needed and store them into `statementInsertObj` object in order to generate the statement:

```
1 | var statementId = insert('FTOS_PYMT_Statement',
   | statementInsertObj)
```

Based on the `statementId`, this changes the business status into **Generated**.

## Automatic Allocation

Bellow are the journeys, entities, libraries and endpoints related to the incoming payments **Automatic Allocation** functionality. For incoming [Payments Deallocation](#) scroll down or click the link.

## Data Model

- Entity **FTOS\_PYMT\_Payment** with all attributes.
- Entity **FTOS\_PYMT\_PaymentScheduleItemXPayment** with all attributes.

## Payment Allocation

From the **FTOS\_PYMT\_PaymentAllocation** server side library, the following functions are used for automatic payment allocation:

### getPayment

This function gets the details of the payment to be allocated.

**Input** parameters: `paymentId` - The Id of the payment.

**Output** parameters: `payment[0]` - Query result containing details of the requested payment .

## getReferenceValues

This function gets the reference values to be applied.

**Input** parameters:

- `payment` - The selected payment.
- `processor` - The selected processor.

**Output** parameters: `result` - Variable containing all references to be applied.

## getStatementsByConditions

This function gets the invoices that match the specified conditions, passed in the input parameters.

**Input** parameters:

- `type` - The type of payment.
- `whereValuesObj` - An object with the values of the payment.
- `referenceCurrencyId` - The Id for the payment currency, extracted from the invoice reference.

**Output** parameters: `result` - Query result containing the invoices matching the conditions.

## getPoliciesByPolicyNo

This function gets a list of policies grouped by policy number.

**Input** parameters: `policyNo` - The policy number.

**Output** parameters: `result` - Query result containing the specified policies.

## checkEnforcedPolicy

This function verifies whether the status of a policy is contained into the input parameter `policyStatusCheck` values.

**Input** parameters:

- `policy` - The selected policy.
- `policyStatusCheck` - The status of the selected policy.

**Output** parameters: `true/ false` .

## checkUnpaidInstallments

This function gets the list of all unpaid installments for the specified policy.

**Input** parameters: `policyId` - The Id of the policy.

**Output** parameters: `result` - Query result containing the unpaid installments for the policy.

## applyPolicyRules

This function verifies whether the policy is eligible for automatic allocation.

**Input** parameters:

- `policyNo` - The policy number.
- `processor` - The selected processor.

**Output** parameters: `true/ false`.

## findStatements

This function finds the invoices that comply with the references contained in the `processor` input parameter.

**Input** parameters:

- `payment` - The selected payment.
- `processor` - The selected processor.

**Output** parameters: `statementsList` - Query result containing a list of invoices.

## getStatement

This function gets details about the requested invoice.

**Input** parameters: `statementId` - The Id of the invoice.

**Output** parameters: `statementList[0]` - Query result.

## getStatementDetail

This function verifies whether the installment unpaid amount is greater than 0.

**Input** parameters: `statementDetailId` - The Id of the invoice detail (only the amount of the payment).

**Output** parameters: `statementDetailList` - Query result.

## getStatementDetailList

This function gets the invoice details about unpaid installments.

**Input** parameters:

- `statementId` - The Id of the invoice.
- `exceptStatementDetailId` - The Id of the invoice detail for an unpaid installment.

**Output** parameters: `statementDetailList` - The result of the query.

## getStatementDetailInstallmentList

This function gets the list of installments.



**Input** parameters: `statementId` - The Id of the invoice.

**Output** parameters: `statementDetailList` - The result of the query.

## getDefaultExchangeRateType

This function gets the type of the default exchange rate.

**Input** parameters: N/A.

**Output** parameters: `defaultExchangeRateType` - The result of the query.

## getExchangeRateId

This function gets the Id for the exchange rate.

**Input** parameters:

- `exchangeType` - The exchange type for the payment.
- `fromCurrency` -
- `toCurrency` -
- `exchangeDate` - The exchange date for the payment.

**Output** parameters: `result` - The result of the query.

## insertAllocation

This function carries out the following actions:

- Generates a `statementPayment` - that is a merger between the invoice and the payment made for that invoice. The `statementPayment` parameter contains the `statementPaymentDetail` and the `statementPaymentXPayment` values.
- Updates the payment and the invoice amounts.
- Inserts the allocation on the specified installment.

- Updates the remaining amounts, if the case.
- Inserts statement details for the specified statement (invoice).

**Input** parameters:

- `payment` - A payment.
- `statement` - An invoice.
- `amount` - The amount of the payment.
- `alocatedAmount` - The allocated amount.
- `exchangeRateId` - The Id of the exchange rate.
- `statementDetailId` - The Id of the invoice detail.

**Output** parameters: `statementPaymentId` - The Id of the statementPayment (invoice & payment merger).

## getInstallmentItemList

This function gets the installment details for the specified installment.

**Input** parameters: `installmentId` - The Id of the installment.

**Output** parameters: `fetchInstallmentItemList` - Query result.

## insertInstallmentItemAllocation

This function inserts a record into the **FTOS\_PYMT\_PaymentScheduleItemXPayment** entity with the status **Generated** for the references specified in the input parameter.

**Input** parameters: `param`

**Output** parameters: N/A.

## insertAllocationStatementDetail

This function inserts the invoice details and the **FTOS\_PYMT\_StatementPaymentDetailXPayment** record, according to the allocated amount and the installment unpaid amount.

**Input** parameters:

- `statementDetailList` - The list containing the statement details.
- `amount` - The amount of the payment.
- `statementPaymentId` - The Id of the statementPayment (invoice & payment merger).
- `payment` - The selected payment.

**Output** parameters: N/A.

## changeStatusStatementPayment

This function changes the status for the specified statementPayment (invoice & payment merger).

**Input** parameters:

- `statementPaymentId` - The Id of the statementPayment (invoice & payment merger).
- `newStatus` - The new business status of the installment.

**Output** parameters: N/A.

## getInstallment

This function gets the data about the specified installment.

**Input** parameters: `installmentId` - The Id of the installment.

**Output** parameters: `fetchResult[0]` / `null` - The result of the query.

## getCurrencyCode

This function gets the code of the specified currency.

**Input** parameters: `currencyId` - The Id of the currency.

**Output** parameters: `fetchResult[0].code` / `null` - The result of the query.

## getToleranceValue

This function gets the specified tolerance value.

**Input** parameters:

- `processor` - The selected processor.
- `currencyId` - The Id of the currency.

**Output** parameters: `result` - Tolerance value for the requested currency.

## getExchangeCalculationType

This function gets the type of calculation for the exchange rate.

**Input** parameters: `processor` - The selected processor.

**Output** parameters: `exchangeCalculationType` - The result of the query.

## insertWriteOff

This function inserts the payment record containing the write-off value.

**Input** parameters:

- `writeOffAmount` - The write-off amount.
- `statement` - The selected invoice.
- `exchangeRateId` - The Id of the exchange rate for the selected currency.

**Output** parameters: `paymentId` - The Id of the payment.

## checkStatement

This function verifies if the invoice is eligible for automatic allocation.

**Input** parameters: `statementId` - The Id of the invoice.

**Output** parameters: `true/ false`.

## automaticPaymentAllocation

This function verifies all the conditions and calls the `insertAllocation` function.

**Input** parameters: `paymentId` - The Id of the payment.

**Output** parameters: N/A.

## Payment Deallocation

Serve side functions used for automatic payment deallocation:

## deallocateStatement

This function deallocates the allocated amount from the invoice and returns it to the payment.

**Input** parameters:

- `statementId` - The Id of the invoice.
- `paymentId` - The Id of the payment.

**Output** parameters: N/A.

## deallocateInstallments

This function updates the installment paid amount to 0.

**Input** parameters:

- `installmentId` - The Id of the installment.
- `statementId` - The Id of the invoice.
- `paymentId` - The Id of the payment.

**Output** parameters: N/A.

## getExchangeRateValue

This function gets the exchange rate value for the specified payment.

**Input** parameters: `exchangeRateId` - The Id of the exchange rate for the selected currency.

**Output** parameters: `fetchResult[0].a_exchangeRate` - The result of the query.

## getPayments

This function gets details about the specified payment.

**Input** parameters:

- `statementId` - The Id of the invoice.
- `paymentId` - The Id of the payment.

**Output** parameters: `fetchResult` - The query result containing details about the requested payment.

## getWOPayments

This function gets details about the specified write-off payment.

**Input** parameters:

- `statementId` - The Id of the invoice.
- `statementNo` - The invoice number.

**Output** parameters: `fetchResult` - The query result containing details about the requested write-off payment.

## deleteInstallmentItemAllocation

This function deletes the afferent installment item allocation record from the **FTOS\_PYMT\_PaymentScheduleItemXPayment** entity.

**Input** parameters: **allocationId** - The Id of the allocation record.

**Output** parameters: N/A.

## deleteStatementPaymentDetailXPaymentAndStatementPaymentDetail

**Input** parameters:

- **statementId** - The Id of the invoice.
- **paymentId** - The Id of the payment.
- **tip** -

**Output** parameters: N/A.

## deleteStatementPaymentXPaymentAndStatementPayment

This function

**Input** parameters:

- **statementId** - The Id of the invoice.
- **paymentId** - The Id of the payment.
- **tip**

**Output** parameters: N/A.

## deallocateStatementDetail

This function updates the invoice detail - the line containing the amount of the payment.

**Input** parameters:

- `statementDetailId` - The Id of the invoice detail.
- `updateObj` - An object containing the details for the update.

**Output** parameters: N/A.

## deallocateStatementAmount

This function updates invoice to initial values.

**Input** parameters:

- `statementId` - The Id of the invoice.
- `amountDeallocateFromStatement` - The amount to be deallocated from the invoice.

**Output** parameters: N/A.

## deallocatePayments

This function updates payment to initial values.

**Input** parameters:

`paymentId` - The Id of the payment.

`newRemainingPaymentAmount` - The remaining amount of the payment.

`newAllocatedAmount` - The new allocated amount.

**Output** parameters: N/A.

## getAllStatementDetailList

This function gets a list with invoice details - that is payment amounts.

**Input** parameters: `statementId` - The Id of the invoice.

**Output** parameters: `statementDetailList` - The result of the query.

## getStatementDetailForPaymentList



This function gets a list with all the payments (from invoice details) to be compared with the data from the payment list.

**Input** parameters:

`statementId` - The Id of the statement.

`paymentId` - The Id of the payment.

**Output** parameters: `statementDetailList` - The result of the query.

## Manual Allocation

Here are the journeys, entities, libraries and endpoints related to the **Manual Allocation** functionality:

## Data Model

Entity **FTOS\_PYMT\_Payment** with all attributes. This entity stores payment data from all **FintechOS** insurance solutions.

### FTOS\_PYMT\_Payment\_EditForm Journey

This is a user journey aimed at implementing the **Manual Allocation** functionality. This journey is marked as **Default** for insert on **FTOS\_PYMT\_Payment** entity.

**Search Statement:** After clicking the **Allocate** button on the **FTOS\_PYMT\_Payment\_EditForm**, the **Statement Search** grid is displayed. Clicking the **Search Statement** button returns data about an **Installment** or a **Statement**, based on the following search parameters:

- `startDueDateFrom`
- `startDueDateTo`
- `firstName`
- `statementAmount`
- `lastName`

- `PaymentTypeSearchId`
- `policyNo`
- `currencyId`
- `currencyName`

**NOTE**

In order to perform a search the user must specify at least one parameter.

In order to show the results, we are using the following functions:

## getInstallmentSearch

This function uses `FTOS_AllocationInstallmentSearch` automation script to get installments based on search. Next, it uses the `generateCustomGrid` function to return the results in a grid.

**Input** parameters: N/A.

**Output** parameters: Returns the fetch - installments to be allocated.

## getStatementSearch

This function uses the `FTOS_INSP_ManualAllocationSearch` automation script to get statements based on search. Next, it uses the `generateCustomGrid` function to return the results in a grid.

**Input** parameters: N/A.

**Output** parameters: N/A.

## generateCustomGrid

This function generates the grid that displays the results of the search.

**Input** parameters:

- `viewId` - (String) – The name of the CSS Id.
- `viewDataSource` – (Object/JSON) – The data resulted from calling the `getInstallmentSearch` or `getStatementSearch()` function.
- `viewColumns` – (Variable) – The type of columns that are displayed in the grid.

**Output** parameters: N/A.

## generateCustomGridInstallments

This function generates the grid that displays the found installments.

**Input** parameters:

- `viewId` - (String) – The name of the CSS Id.
- `viewDataSource` – (Object/JSON) – The data result from the `getInstallmentSearch` or `getStatementSearch` function.
- `viewColumns` – (Variable) – The type of columns that are displayed in the grid.

**Output** parameters: N/A.

## allocateClickFunction

This function uses the `FTOS_PYMT_ManualPaymentAllocation` server automation script in order to allocate the selected payment.

**Input** parameters: `details` – (Object) – The needed data from the `generateCustomGridInstallments` function in order to manually allocate a payment.

**Output** parameters: N/A.

In order to manually allocate a payment, the user clicks on the **Add** button next to the desired **Statement** or **Installment** from the list.

## Server Automation Scripts

### FTOS\_AllocationInstallmentSearch

This script fetches the installments for the search made on the **FTOS\_PYMT\_Payment\_EditForm** form, after clicking on the **Search Statement** button.

The script contains the flowing functions:

#### getInstallmentSearchFetch

This function creates the fetch for the results.

**Input** parameters: N/A.

**Output** parameters: Returns the fetch.

#### addSearchConditionsInstallmentSearch

This function adds the search conditions from the input parameter to the original fetch.

**Input** parameters: `searchObj` – (Object) – This object contains the search values selected by the user in the front end.

**Output** parameters: Returns the fetch with the newly added conditions.

### FTOS\_INSP\_ManualAllocationSearch

This script fetches the statements for the search made on the **FTOS\_PYMT\_Payment\_EditForm** form, after clicking on the **Search Statement** button.

The script contains the flowing functions:

#### getStatementsSearchFetch

This function creates the fetch for the results.

**Input** parameters: N/A.

**Output** parameters: Returns the fetch.

## addSearchConditionsStatementSearch

This function adds, to the original fetch, the search conditions from the input parameter.

**Input** parameters: `searchObj` – (Object) – This object contains the search values selected by the user in the front end.

**Output** parameters: Returns the fetch with the newly added conditions.

### Server Automation Script Library

The `FTOS_PYMT_PaymentAllocation_Manual` library contains an object with the following functions:

## getPaymentAmount

This function fetches the payment.

**Input** parameters: `paymentId` – (variable) -The Id of the payment.

**Output** parameters: Returns the fetch.

## getPaymentScheduleDetail

This function fetches the payment schedule.

**Input** parameters: `installmentId` – (variable) -The Id of the installment.

**Output** parameters: Returns the fetch.

## insertStatementDetailForStatement

If there is an installment and the installment unpaid amount is greater than 0, then this function inserts the below object into the `FTOS_PYMT_StatementDetail` entity. The object properties are:

- `statementId`,
- `installmentId`,
- `insuranceProductId`,
- `installmentUnpaidAmount`,
- `dueDate`,
- `policyId`,
- `insuredId`,
- `grossInstallmentAmount`,
- `totalInstallmentAmount`,
- `customerId`.

**Input** parameters:

- `statementId` – (variable) – The Id of the statement.
- `installments` – (array) – An array with the installments.

**Output** parameters: N/A.

## insertStatementForInstallment

If there is a statement and the installment unpaid amount is greater than 0, then this function inserts the below object into the **FTOS\_PYMT\_Statement** entity. The functions also calls the `getPaymentScheduleDetail` and `insertStatementDetailForStatement` functions. Finally, it changes the business status to **Generated** for the selected payment. The object properties are:

- `statementType`,
- `customerId`,

- `installmentsStartDate`,
- `dueDate`,
- `statementMonthId`,
- `installmentsEndDate`,
- `statementAmount`,
- `statementDate`,
- `unpaidAmount`,
- `unconfirmedPayAmount`,
- `confirmedPayAmount`,
- `currencyId`,
- `brokerId`,
- `productId`,
- `isGenerated`,
- `expireDate`,
- `paymentTypeId`.

**Input** parameters: `installmentId` – (variable) – The Id of the installment.

**Output** parameters: If currency is missing , the output is an **error message**. If currency is present and the `sumInstallmentAmount` is greater than 0, the output is an `object` with the following properties:

- `statementId`,
- `statementDetailId`.

## getRemainingInstallmentInPaymentCurrency

This function returns the remaining installment amount in the payment currency.

This function also uses the following libraries:

- **FTOS\_PYMT\_PaymentAllocation** – To call the `getDefaultExchangeRate` function from the Payment Allocation object.
- **FTOS\_INS\_Exchange** - To call the `returnExchangeRate` function from the Exchange object.

**Input** parameters:

- `amount` – (variable) – The amount of the installment.
- `installmentCurrency` – (variable) – The installment currency.
- `paymentCurrency` – (variable) – The payment currency.
- `date` – (variable) – Not used.

**Output** parameters: Remaining installment amount, in payment currency.

## getAmountInPaymentCurrency

This function returns the installment amount in the payment currency.

This function also uses the following libraries:

- **FTOS\_PYMT\_PaymentAllocation** – To call the `getDefaultExchangeRate` and the `getExchangeCalculationType` functions from the Payment Allocation object.
- **FTOS\_DFP\_FlowProcessorSettings** – To call the `getFlowProcessorSettingsByType` function from the Flow Setting object.
- **FTOS\_INS\_Exchange** - To call the `returnExchangeRate` function from the Exchange object.

**Input** parameters:



- `amount` – (variable) – The amount of the installment.
- `installmentCurrency` – (variable) – The installment currency.
- `paymentCurrency` – (variable) – The payment currency.
- `paymentDate` – (variable) – The payment date.
- `statementDate` – (variable) – The statement date.

**Output** parameters: Returns the installment amount in payment currency.

## allocatePaymentOnInstallment

This function allocates a payment for an installment. First the statement is generated and then the payment is allocated. If the payment is greater than the installment, the remaining amount can be manually allocated to other installments.

This function also uses the following libraries:

- **FTOS\_PYMT\_PaymentAllocation** – To call the `getPayment`, `getStatement`, `getDefaultExchangeRateType` and `insertAllocation` functions from the Payment Allocation object.
- **FTOS\_INS\_Exchange** - To call the `returnExchangeRate` function from the Exchange object.

**Input** parameters:

- `installmentId` – (variable) – The Id of the installment.
- `paymentId` – (variable) – The Id of the payment.
- `statementId` – (variable) – The Id of the statement.
- `statementDetailId` – (variable) – The statement detail id.

**Output** parameters: N/A.

## getExchangeRateDetails

This function returns the exchange rate details.

This function also uses the following libraries:

- **FTOS\_PYMT\_PaymentAllocation** – To call the `getDefaultExchangeRateType` and `getExchangeCalculationType` functions from the Payment Allocation object.
- **FTOS\_DFP\_FlowProcessorSettings** – To call the `getFlowProcessorSettingsByType` function from the Flow Setting object.
- **FTOS\_INS\_Exchange** - To call the `returnExchangeRate` function from the Exchange object.

**Input** parameters:

- `paymentDate` – (variable) – The payment date.
- `paymentCurrencyId` – (variable) – The payment currency id.
- `statementDate` – (variable) – The statement date.
- `statementCurrencyId` – (variable) – The statement currency id.

**Output** parameters: The exchange rate for the selected payment.

## allocatePaymentOnStatement

This function allocates a payment on a statement.

This function also uses the **FTOS\_PYMT\_PaymentAllocation** library to call the `getPayment`, `getStatement`, `getDefaultExchangeRateType` and `insertAllocation` functions from the Payment Allocation object.

**Input** parameters:

- `paymentId` – (variable) – The Id of the payment.
- `statementId` – (variable) – The Id of the statement.

**Output** parameters: N/A.

## SEPA Direct Debit

This functionality handles the direct debit payment operations for SEPA area.

SEPA refers to the **Single Euro Payments Area** - a payment scheme which facilitates cashless payments (bank transfers via credit transfer and direct debit) anywhere inside the European Union.

Below are the journeys, entities, libraries and scripts related to this functionality:

### Digital Journeys

## Entity FTOS\_PYMT\_DirectDebitNotification

### Journey DirectDebitNotification

After uploading and importing the data from the **Notification** file, the user is redirected to the direct debit mandate **Registration** form. This form has two steps (tabs). In the first tab, the user finds information about the notification file - the name of the file and the import date. In the second tab, the user can see the direct debit mandates buffer activated by that specific notification file.

### Journey DirectDebitNotificationInsert

This is a user journey aimed at implementing the **Notification Manual Insert** functionality. The journey has a form that allows the user to upload a **Notification** file by pressing the **Import Data** button.

## Entity FTOS\_PYMT\_DirectDebitMandate

### Journey FTOS\_PYMT\_DirectDebitMandate

This form is used to see all the details about the direct debit mandates registered, on step 1 and the history of changes logged on the selected mandate, on step 2. Also, the form allows the user to manually select and cancel an **Active** mandate, by using the **Cancel Mandate** button.

## Entity FTOS\_PYMT\_DirectDebitMandateBuffer

This form is used to see all the details related to the direct debit mandates from the imported file.

## Entity FTOS\_PYMT\_DirectDebitConfirmation

### View Default

This view allows the user to see the list of all the direct debit payments files ever uploaded in the system. From this view, the user double-clicks the **FTOS\_PYMT\_DirectDebitConfirmation** in order to see the details of the selected record.

### Journey FTOS\_PYMT\_DirectDebitConfirmationInsert form

This insert form allows the user to manually upload a file containing direct debit payments (not confirmed yet) by pressing the **Import Data** upload button.

### Journey FTOS\_PYMT\_DirectDebitConfirmation form

After uploading and importing the data from the file, the user is redirected to this form which represents the confirmation of registering new direct debit mandate payments in the system. This form has two steps (tabs). In the first tab, the user finds the file confirmation details: the name of the file and the import date. In the second tab, the user has a view of the payments contained by that specific file.

## Entity FTOS\_PYMT\_DirectDebitConfirmationDetail

### Journey DirectDebitConfirmationDetail\_ReadOnly form

This form is used to see all the details about the payments confirmation records from the payments confirmation file.

### View Default

This view displays the list of payments confirmation files. From this view, the user can select any file, by double-clicking the **DirectDebitConfirmationDetail\_ReadOnly** form, in order to see the file details.

## Entity FTOS\_PYMT\_DIDE

**Journey FTOS\_PYMT\_DIDEReadOnly form**

This form displays details (file name and date of generation) about the DIDE instruction files generated by the [FTOS\\_PYMT\\_DIDEInstructionFile](#) scheduled job. Also, from this form the user can download the generated file.

**View FTOS\_PYMT\_DIDEReadOnly**

This view displays the list of all the DIDE instruction files generated by the [FTOS\\_PYMT\\_DIDEInstructionFile](#) scheduled job. From this view, the user can select any file, by double-clicking the **FTOS\_PYMT\_DIDEReadOnly** form, in order to see the file details.

## On Demand Server Automation Scripts

**FTOS\_PYMT\_DirectDebitNotification\_File\_Validation**

On demand script that is triggered when a file is uploaded during the **DirectDebitNotificationInsert** journey. After the user presses the **Import Data** button, the script validates that the file is in **.txt** format. The script also restricts the upload to only one file.

**Input** parameters: `file` - The DIDE payments file that needs to be uploaded.

**Output** parameters: `fileValidation` (boolean) - The result of the validation (true/ false).

**FTOS\_PYMT\_DirectDebitMandateStatusChange**

This on demand script calls the `changeMandateBusinessStatus (mandateId)` function from **FTOS\_PYMT\_Mandate** server side library. The `mandateId` parameter is passed as **null** value in order to run the status changes for all the mandates.

**Input** parameters: N/A.

**Output** parameters: N/A.

**FTOS\_PYMT\_DIDEInstructionFile**

This on demand script calls the `runDIDEInstructionFile` function from the `FTOS_PYMT_DIDE` server side script library, in order to generate the payment instructions file for the direct debit mandates.

**Input** parameters: N/A.

**Output** parameters: N/A.

## FTOS\_PYMT\_DirectDebitMandate\_cancelMandate

This on demand script calls the `changeBusinessStatus` function, in order to apply the **Cancelled** business status to the selected record.

**Input** parameters:

`recordID` - Where the `recordID` is the `mandateId`. This parameter is passed from the context of the client-side process that called this script.

`statusID` = `cancelled` - The value to be set for the mandate business status.

**Output** parameters: N/A.

## Business Workflow Configuration Actions

`FTOS_PYMT_DirectDebitMandate` is the master business workflow that handles the different types of changes affecting a direct debit mandate during its lifetime. For more details about the mandate behavior, its states and the business workflow diagram consult the [Direct Debit](#) page.

Transition	Description
_Proposal	Initial state.
Active_Cancelled	When the system registers a notification about the mandate cancellation.
Active_Expired	When the mandate reaches its end day. The expiration triggers the automatic change of the payment type on the policy, from Direct Debit to <b>Bank Transfer (OP)</b> .
Active_VersionClosed	When a version of the mandate is closed. Used for mandate versioning.
Approved_Active	When the mandate reaches its start day.

Transition	Description
Approved_Draft	When a change of the <b>policy payment type</b> from Bank Transfer (OP) into <b>Direct Debit</b> is performed on a policy.
Approved_Expired	When the number of days for the activation of a mandate were exhausted.
Draft_Active	When the mandate reaches its start day. This transition is triggered automatically by a specific job that verifies if the current date is the mandate begin date and changes the status to <b>Active</b> for all eligible mandates.
Draft_Cancelled	When the system registers a notification about the mandate cancellation.
Draft_Expired	When the number of days for the activation of a mandate were exhausted. The expiration triggers the automatic change of the payment type on the policy, from Direct Debit to <b>Bank Transfer (OP)</b> .
Draft_Pending	After the instruction file for the mandate's activation is generated, and the mandate is pending approval from the bank.
Draft_VersionDraft	When the mandate versioning process starts.
Pending_Active	When the mandate reaches its start day.
Pending_Cancelled	When the system registers a notification about canceling the mandate, or a user manually cancels the mandate.
Proposal_Active	When the mandate reaches its start day.
Proposal_Draft	When the mandate is registered in the system but it is not activated yet, or its begin date is yet to come.
Proposal_Expired	When the number of days for the activation of a draft mandate were exhausted.
VersionDraft_Approved	When a version of the mandate is approved. Used for mandate versioning.
VersionDraft_VersionUnapproved	When the opened version is not approved. Used for mandate versioning.

**HINT**

To check out this business workflow open the **Innovation Studio** and go to **Fintech Automation >> Business Workflow Designer**. For more details, consult also the [Business Workflow Design](#) documentation.

## Endpoints

### FTOS\_PYMT\_DirectDebitMandate\_cancelMandate

This endpoint calls the **FTOS\_PYMT\_DirectDebitMandate\_cancelMandate** script to change the mandate business status into **Cancelled**, when clicking the **Cancel Mandate** button.

## Processors

The **FTOS\_PYMT\_DIDEProcessor** is used for setting the following parameters:

## DideConfiguration

This is an object containing the following settings:

- **numberOfDaysInAdvance** - This parameter sets the date for invoice generation with N days before the payment due date (for example, the invoice can be generated with 3 or 5 or 10 days before the payment due date).
- **runningOnBankHolidays** - This parameter determines whether the DIDE instruction file is generated on bank holidays or not (true or false). Set to true if the DIDE instruction file is to be generated on a bank holiday.
- **excludeWeekDays** - This parameter defines the week days in which the DIDE instruction file is not generated (for example, the invoice is not generated on Saturday or Sunday).
- **type** - This parameter sets the type of direct debit processing: either SEPA (for European area) or UK.

If the **type** option configured in the processor is **Sepa** - the DIDE process will take in consideration the functionality implemented for DIDE Sepa. Otherwise, the DIDE process will take into consideration the flow implemented for DIDE UK.

## Sequencers

### FTOS\_PYMT\_DIDE sequencer



This sequencer is used to set the `Order Number` value used for generating DIDE instruction files.

- Range **Min** = 1
- Range **Max** = 999,999,999

## Server Side Script Libraries

### FTOS\_PYMT\_Mandate

From the `FTOS_PYMT_Mandate` library, the following functions are used:

#### PYMT\_Buffer

This function wraps a series of functions which perform actions on the `FTOS_PYMT_DirectDebitMandateBuffer` entity or interpret data received by it. Inside the `PYMT_Buffer` function, the following functions are used:

#### updateBufferMandateId

This function updates the `directDebitMandateId` attribute on a buffer that triggers an action for a mandate (insertion, deletion or modification).

**Input** parameters:

- `bufferId` - The Id of the buffer on which the update needs to be implemented.
- `mandateId` - The Id of the mandate on which the buffer performed the specified action.

**Output** parameters: N/A.

#### performActionsOnMandate

This function detects the type of action that the buffer needs to perform on a mandate, based on the **stageId** that the buffer has. If there is no **stageId** set on the buffer, the function throws an exception, with an error message.

**Input** parameters: `bufferObj` - The object containing the data added in the buffer.

**Output** parameters: `mandateId` - The Id of the mandate which was created, modified, or cancelled.

## PYMT\_Mandate

Inside the `PYMT_Mandate` function, the following functions are used:

### getMandateStage

This function gets the stage of the mandate - that is whether the mandate is **N**(New), **M**(Modified) or **D** (Deleted).

**Input** parameters: `mandateId` - The mandate unique identifier.

**Output** parameters: N/A.

### changeMandateBusinessStatus

This function updates the mandate business status according to the logic from the `getMandateNewBusinessStatus` function (see below).

**Input** parameters: `mandateId` - The mandate unique identifier.

**Output** parameters: N/A.

## updateMandateBusinessStatusId

This function updates the mandate business status.

**Input** parameters:

- `mandateId` - The mandate unique identifier.
- `statusId` - The business status unique identifier.

**Output** parameters: N/A.

## getMandateNewBusinessStatus

This function returns the Id of the new business status according to the logic between begin date, end date and current date.

**Input** parameters:

- `beginDate` - The beginning date for the mandate.
- `endDate` - The ending date for the mandate.

**Output** parameters: Returns the `newBusinessStatusId`.

## getMandateDetailsByReference

This function returns the mandate details based on the reference.

**Input** parameters: `reference` - The mandate reference (expected policy number).

**Output** parameters: Returns the fetch.

## changeMandateStatusCancelled

This function updates the mandate business status to **Cancelled**.

**Input** parameters:

- `mandateRef` - The mandate reference (expected policy number).
- `stageId` - The mandate stage.

**Output** parameters: N/A.

## getCurrencyIdByCode

This function returns the Id of the currency based on the code.

**Input** parameters: `currencyCode` - The currency code.

**Output** parameters: The `currencyId` or `null`.

## deleteMandateFromBuffer

This function executes the following:

- It updates the business status of the mandate to **Cancelled**.
- It updates the `stageId` attribute to the one received from the buffer.
- It calls the `updateBufferMandateId` function to update the `directDebitMandateId` attribute of the buffer.
- If no mandate is found based on the reference, the function throws an error.

**Input** parameters: `bufferObj` - The object containing the data added in the buffer.

**Output** parameters: `mandateId` - The Id of the cancelled mandate.

## formatDateWithOffset

This function receives a date in a string format. Next, the function converts the date to a date object, with the correct server time.

**Input** parameters: `date` - The string containing a date.

**Output** parameters: `dateWithOffset` - The date object.

## addMandateFromBuffer

This function executes the following:

- It inserts a new mandate record, filling in with the data received from the buffer.
- It also calls the `updateBufferMandateId` function to update the `directDebitMandateId` attribute of the buffer.
- If a mandate with the same reference from the data object already exists in the system, the function throws an error.

**Input** parameters: `bufferObj` - The object containing the data added in the buffer.

**Output** parameters: `mandateId` - The Id of the newly created mandate.

## getMandateLastVersion

This function fetches from the **FTOS\_PYMT\_DirectDebitMandate** entity the Id of the last version for a specified mandate.

**Input** parameters: **mandateId** - The Id of the mandate for which the last version was requested.

**Output** parameters: **buffers** - The mandate list that has been found.

## modifyMandateFromBuffer

This function executes the following:

- Creates a new version for a mandate with a specific reference.
- Updates the mandate with the new data received from the buffer.
- Changes the business status of the new version, depending on the **beginDate** and **endDate** attributes.
- Changes the business status of the old version of the mandate with the **versionClosed** status.
- Calls the **updateBufferMandateId** function to update the **directDebitMandateId** attribute of the buffer.

**Input** parameters: **bufferObj** - The object containing the data added in the buffer.

**Output** parameters: **mandateId** - The Id of the newly modified mandate.

## changeBusinessStatusSEPA

This function changes the business status for SEPA mandates.

**Input** parameters: N/A.

**Output** parameters: N/A.

## changeSingleMandateStatus

This function changes the business status of the specified mandate.

**Input** parameters: `mandateId` - The Id of the mandate.

**Output** parameters: N/A

## getSEPAmandates

This function returns all the SEPA mandates that are in **Draft**, **Active** or **Expired** business status.

**Input** parameters: N/A.

**Output** parameters: The query object.

## getMandateDetailsById

This function gets the details of the specified mandate.

**Input** parameters: `mandateId` - The id of the mandate.

**Output** parameters: The query object.

## FTOS\_PYMT\_DirectDebitMandateConfirmation

From the **FTOS\_PYMT\_DirectDebitMandateConfirmation** library, the following functions are used:

## addPaymentConfirmation

This function inserts a payment confirmation record into the **FTOS\_PYMT\_DirectDebitMandateConfirmation** entity.

**Input** parameters: `input` - The object containing details about the new payment confirmation.

**Output** parameters: `directDebitMandateConfirmationId` - The Id of the newly inserted payment confirmation.

## dbTaskPaymentConfirmation

This function saves the contents of the uploaded payments confirmation file.

**Input** parameters:

- `fileArr` - The object containing the payments confirmation file.
- `confirmationObj` - The object containing the Id of the new file inserted and details about the user who performed the insert of the file.

**Output** parameters: N/A.

Also, inside this function, the following functions are used:

- `getTabularDataFromFile` - To get data from the payments confirmation file.
- `saveToTemporaryTable` - To save the values in a temporary table called **DirectDebitConfirmationTempTable**.

The contents of the file are saved with the **FTOS\_InsertDirectDebitConfirmationDetail** stored procedure which returns an array of payments confirmation from the **FTOS\_PYMT\_DirectDebitConfirmationDetail** entity.

## FTOS\_PYMT\_DIDE



This server side script library contains methods that help in the process of generating a direct debit payments instruction file for both SEPA and UK types.

From the **FTOS\_PYMT\_DIDE** library, the following functions are used:

## General

Inside the `General()` function, the following functions are used:

### isNullOrEmpty

This function verifies if an input value is **null** or **empty**.

**Input** parameters: `value` - The value that needs validation.

**Output** parameters: `true/ false` - The result of the validation.

### formatDateForFile

This functions transforms the format of an input date to the **ddmmyy** format. Next, it adds a number of days to the received date.

**Input** parameters:

`date`

`noOfDays` - How many days to add to the date received as an input.

**Output** parameters: Date formatted.

### getProcessorValues

This function returns the static values stored in the **FTOS\_PYMT\_DIDEProcessor** processor.

**Input** parameters: `setting` - The processor's object key name.

**Output** parameters: The object containing values from the processor:

- `destinationSort`
- `destinationAcct`
- `destinationType`
- `usersName`
- `transaction`
- `freeFormat`
- `amount`

## DIDE

This function wraps a series of functions which perform actions on the **FTOS\_PYMT\_DIDE** and **FTOS\_PYMT\_DIDE****Detail** entities, generate and validate some data or generate **.txt** files, as follows:

### runDIDEInstructionFile

This function executes the following:

- Verifies that the DIDE instruction file can be generated according the rules defined in the **FTOS\_PYMT\_DIDE****Processor** processor (`runningOnBankHolidays`, `numberOfDaysInAdvance`, and `excludeWeekDays`).
- Generates the details needed for the file type wanted.

- Saves the `statementId` and `directDebitMandateId` details into the **FTOS\_PYMT\_DIDE** entity.
- Saves the `directDebitMandateId` details in **FTOS\_PYMT\_DIDEMandateInstructionDetail** entity, according to the file type.
- Generates the .txt file.
- Saves the file into the **FTOS\_PYMT\_DIDE** entity or into the **FTOS\_PYMT\_DIDEMandateInstruction** entity.
- It also saves the file on the server **UploadEBS** folder.

This function also uses all the functions mentioned below.

**Input** parameters: `instructionFileType` - Optional parameter. This parameter has two possible values:

1. `statements` - for files dealing with payments - DIDEPaymentsInstructionFile (for **SEPA**) and, respectively, DD\_BACSPayments file (for **UK**).
2. `mandates` - for files dealing with creation of new DIDE mandates - NEW\_DIDE\_Mandates\_BACSIInstructions file (for **UK**).

**Output** parameters: N/A.

## getProcessorDetails

This function gets the configuration details of a processor.

**Input** parameters:

- `flowSettingsName` - The name of the flow setting.
- `processorSettingsType` - The type of the **Digital Processor Type**.
- `processorSettingsName` - The name of the processor setting.

**Output** parameters: `processorDetails` - An object containing the processor settings `runningOnBankHolidays`, `excludeWeekDays` and `numberOfDaysInAdvance`.

## getDayName

This function returns the name of a weekday (ex. Monday).

**Input** parameters: `date` - The date parameter.

**Output** parameters: `dayName` - The name of the week day.

## getDIDERunningDay

This function verifies if the current day is a bank holiday or a day defined in the **FTOS\_PYMT\_DIDEProcessor**. If one condition is met the function returns a true value.

**Input** parameters: `settings` - The object with the processor settings - `runningOnBankHolidays`, `excludeWeekDays` and `numberOfDaysInAdvance`.

**Output** parameters: `True` or `false`.

## getDraftMandates

This function is specific for DIDE UK. It returns all the UK direct debit mandates in **Draft** status.

**Input** parameters: `mandateTypeUk` - The `mandateTypeId` has the **UK** value.

**Output** parameters: Returns the query result.

## getPolicyDataAPI

This function is specific for DIDE UK. This function imports the `FTOS_PYMT_GetPolicyDataAPI` library in order to call the `FTOS_GetPolicyData_API` API. The function sends an object containing a list of policies as mandates references (where the `reference` is the **policy number**) and receives the information about the specified policies, in return. This function also parses the result.

**Input** parameters: `mandateObj` - The list of policies.

**Output** parameters: Information result about the specified policies.

## getStatementsDetails

This function returns statements (invoices) to be included in the DIDE instruction file, based on the following conditions:

- `businessStatus` of the statement = `Generated`
- `paymentType` of the statement = `directDebit`
- `businessStatus` of the direct debit mandate = `active`
- `scheduledDate` of the statement = `null`
- `dueDate` of the statement  $\leq$  current date + `numberOfDaysInAdvance`

- `businessStatus` of the policy = `Enforced` or `Suspended`
- `mandateTypeId` = SEPA or UK

**Input** parameters:

- `numberOfDaysInAdvance` - The number of days in advance for generating an invoice (for a policy installment), before the due date for the payment.
- `mandateTypeId` = SEPA or UK

**Output** parameters: `resultStatements` - Returns the fetch.

## processPaymentsInstructionSepa

This function is specific for SEPA direct debit.

This function executes the following:

- Inspects the results of the `getStatementsDetails` function.
- Inserts the `statementId`, `directDebitMandateId`, and `name` details into the `FTOS_PYMT_DIDE` entity.
- Populates the `CSVContentArray` with all the details necessary for the generation of the DIDE instruction file.
- Updates the `scheduledDate` from the `FTOS_PYMT_Statement` entity.

**Input** parameters:

- `result` - The fetch result of the `getStatementsDetails` (`numberOfDaysInAdvance`, `mandateTypeId`) function.
- `CSVContentArray` - An empty array.

**Output** parameters: N/A.

## processMandatesBACSInstructionUk

This function is specific for DIDE UK. This function executes the following:

- Inspects the results of the query mandates.
- Inserts the `directDebitMandateId` details into the `FTOS_PYMT_DIDEMandateInstructionDetail` entity.
- Populates the `CSVContentArray` with all the details necessary for the generation of the mandate instruction file.
- Sets the `beginDate` on the specified mandates.
- Sets the status on the mandates to **Active**.

In the `CSVContentArray` file, the information is concatenated as follows:

- **First column:** destination sort + destination acct + destination type + transaction + bank Sort Code + account number.
- **Second column:** free format + amount + users name.
- **Third column:** reference + payer last name.
- **Fourth column:** BACS Processing day.

**Input** parameters:

- **resultMandates** - The mandates that respect all conditions for being included in the file.
- **CSVContentArray** - An empty array.

**Output** parameters: N/A.

## processPaymentsInstructionUK

This function is specific for DIDE UK. This function executes the following:

- Inspects the results of the **getStatementsDetails** function.
- Inserts the **statementId**, **directDebitMandateId**, and **name** details into the **FTOS\_PYMT\_DIDE** entity.
- Populates the **CSVContentArray** with all the details necessary for the generation of the UK\_DIDE instruction file.
- Updates the **scheduledDate** from the **FTOS\_PYMT\_Statement** entity

In the **CSVContentArray** file, the information is concatenated as follows:

- **First column:** destination sort + destination acct + destination type + transaction + bank Sort Code + account number.
- **Second column:** free format.
- **Third column:** amount + users name.



- **Fourth column:** reference + payer last name.
- **Fifth column:** BACS Processing day.

**Input** parameters:

- `result` - The fetch result of the `getStatementsDetails(numberOfDaysInAdvance, mandateTypeId)` function.
- `CSVContentArray` - An empty array.

**Output** parameters: N/A.

## getInstallmentNo

This function is specific for DIDE UK. This function returns the installment number for an invoice reference, received as parameter.

**Input** parameters: `statementReference` - The invoice reference.

**Output** parameters: Returns the query result.

## generatePaymentsInstructionFile

This function executes the following:

- Generates the .txt DIDE instruction file based on `csvContentArray` populated by the `processPaymentsInstruction(result, csvContentArray)` function.
- Saves the file into **FTOS\_PYMT\_DIDE** entity.

- Saves the file on the UploadEBS folder from the portal.

**Input** parameters:

**csvContentArray** - The array with all the records for direct debit instructions.

**instructionFileType** - Optional parameter, with two possible values: 1. **statements** for DIDEPaymentsInstructionFile (SEPA) and DD\_BACSPayments (UK) file and 2. **mandates** for NEW\_DIDE\_Mandates\_BACSInstructions file (UK).

**Output** parameters:

**idInstructionFile** - The Id of the new file inserted in **FTOS\_PYMT\_DIDE** entity or **FTOS\_PYMT\_DIDEMandateInstruction** entity.

**entityName** - **FTOS\_PYMT\_DIDE** - for DIDEPaymentsInstructionFile (SEPA) and DD\_BACSPayments (UK) file OR **FTOS\_PYMT\_DIDEMandateInstruction** for the NEW\_DIDE\_Mandates\_BACSInstructions file (UK).

## updateDIDEDetails

This function searches all the records from **FTOS\_PYMT\_DIDEDetail** or **FTOS\_PYMT\_DIDEMandateInstructionDetail** entities (depending on the file type) which have **dideId** (id of the file) with **null** values and updates them with the id of the recently generated file.

**Input** parameters:

- `instructionFileObj` -
- `instructionFileId` - The id of the new generated file and `entityName` - the name of the entity.

**Output** parameters: N/A

## FTOS\_PYMT\_DirectDebitDenied

From the `FTOS_PYMT_DirectDebitDenied` library, the following functions are used.

### PYMT\_DirectDebitDenied

This function wraps smaller functions which perform actions necessary for the process of importing the payments denied file in the system.

### updateMandateStatusCancelled

This function updates the status of the mandates found by the `getMandateDetailsByReference` function to **Cancelled**.

**Input** parameters: `operationReference` - The reference of a mandate.

**Output** parameters: N/A.

### getStatementIdFromPolicyNo

This function fetches the statements (invoices) with the **Generated** or **Unpaid** business statuses from a policy, selected by its specific policy number.

**Input** parameters: `policyNo` - The number of the policy.

**Output** parameters: `statements` - The list of the statements (invoices) found.

## updateStatementsScheduleDate

This function updates the `scheduledDate` attribute of a statement (invoice).

**Input** parameters: `policyNo` - The number of the policy.

**Output** parameters: N/A.

## dideDeniedFileUpdates

This function updates the mandates and the statements (invoices) found based on the operation references received inside an object.

**Input** parameters: `dideDeniedDetails` - The object containing the operation references.

**Output** parameters: N/A.

### Scheduled Jobs

## FTOS\_PYMT\_DirectDebitMandateStatus

This job is scheduled to run every day at 5:00 AM, to update mandate statuses according to the begin date and end date of the mandates.

Schedule Services: `FTOS_PYMT_DirectDebitMandateStatusChange` - on demand server automation script.

## FTOS\_PYMT\_DIDEInstructionFile

The job is scheduled to run every day at 3:00 AM to generate the **.txt** instruction file. Depending on the configuration, it generates the **DIDEPaymentsInstructionFile** (for SEPA) and **DD\_BACSPayments** (for UK), also.

Schedule Services: **FTOS\_PYMT\_DIDEInstructionFile** - on demand server automation script.

## UK Direct Debit

This functionality handles the direct debit payment operations for UK area.

Below are the journeys, entities, libraries and scripts related to this functionality:

### Digital Journeys

## Entity FTOS\_PYMT\_DIDE\_ADDACS

### Journey DirectDebitADDACS

After importing the data from the **ADDACS** file, the user is redirected to this form. This form has two steps (tabs). In the first tab, the user finds information about the **ADDACS** file - the name of the file and the import date. In the second tab, the user can see the direct debit mandates details contained by that specific **ADDACS** file.

The form also allows the user to make changes related to the business status of the direct debit mandates. According to the updates received through the **ADDACS** file, the user can cancel, modify or reactivate existing mandates. However, these modifications can only be performed if the effective date of the record is equal with import date of the file.

#### NOTE

The generation of a new UK direct debit mandate can be done only through **FTOS\_PYMT\_GenerateMandate** endpoint which is described in the [Generate UK Direct Debit Mandate API](#) page.

### Journey DirectDebitADDACSInsert

This form is used to manually upload an **ADDACS** file, by pressing the **Import Data** upload button.

## Entity FTOS\_PYMT\_DirectDebitMandate

### **Journey FTOS\_PYMT\_DirectDebitMandateInsert**

This form is used to manually add a DIDE Mandate in the system for both SEPA and UK, so that Direct Debits can be set up for a policy. By pressing the **Insert** button, the specific Mandate form will be opened according to the DIDE type set in the Direct Debit processor.

### **Journey FTOS\_PYMT\_DirectDebitMandate\_UK**

This form is used to see all the details about the UK direct debit mandates registered - on step 1 and the history of changes for a mandate - on step 2.

## Entity FTOS\_PYMT\_DIDE\_ADDACSDetail

This form is used to see all the details related to the direct debit mandates from the imported **ADDACS** file. On this page, this entity is further called **buffer**.

## Entity FTOS\_PYMT\_DIDE

### **View FTOS\_PYMT\_DIDEReadOnly**

This view displays the list of all the DIDE UK payments instruction files generated by the [FTOS\\_PYMT\\_DIDEInstructionFile](#) scheduled job. From this view, the user can select any file, by double-clicking the **FTOS\_PYMT\_DIDEReadOnly** form, in order to see the file details.

### **Journey FTOS\_PYMT\_DIDEReadOnly form**

This form displays the details (file name and date of generation) of the DIDE UK payments instruction files generated by the [FTOS\\_PYMT\\_DIDEInstructionFile](#) scheduled job. Also, from this form the user can download the generated file.

## Entity FTOS\_PYMT\_ARUDD

### **View Default**

This view allows the user to see the list of all the UK direct debit payments files ever uploaded in the system. From this view, the user double-clicks the **FTOS\_PYMT\_ARUDD form** in order to see the details of the selected record.

### **Journey FTOS\_PYMT\_ARUDDInsert form**

This insert form allows the user to manually upload a file containing direct debit payments (not confirmed yet) by pressing the **Import Data** upload button.

### **Journey FTOS\_PYMT\_ARUDD form**

After uploading and importing the data from the ARUDD file, the user will be redirected to this form which represents the registration of the unconfirmed payments of direct debit mandates.

This form has two steps (tabs). In the first tab, the user finds the file confirmation details: the name of the ARUDD file and the import date. In the second tab, the user has a view of the payments contained by that specific ARUDD file.

## **Entity FTOS\_PYMT\_ARUDDDetail**

### **View Default**

This view displays the list with some details for all the records from the unconfirmed payments file (the ARUDD file). From this view, the user double-clicks the **FTOS\_PYMT\_ARUDDDetail\_ReadOnly form** in order to see the details of the selected record.

### **Journey FTOS\_PYMT\_ARUDDDetail\_ReadOnly form**

This form is used to see all the details about the unconfirmed payments from the ARUDD file.

## **Entity FTOS\_PYMT\_AUDDIS**

### **Journey DirectDebitAUDDISInsert**

This form is used to manually upload an AUDDIS file by pressing the **Import Data** button. The **Import Date** is read-only and is automatically completed after the import process.

#### **Journey DirectDebitAUDDIS**

After uploading the AUDDIS file, the user will be redirected to another form that contains the import date and the file name, on the first tab and all the records from the imported file, on the second tab.

If all the data from the file has been successfully imported, the correlated mandates will transition to the **Cancelled** business status only if they were in the **Active** or in **Pending** statuses. Additionally, all the invoices from the policies which the mandates refer to, will be cancelled.

## Entity FTOS\_PYMT\_AUDDISDetail

This entity is used to save all the records from the imported file with details about the direct debit mandates.

#### **Journey FTOS\_PYMT\_AUDDISDetail\_ReadOnly**

If the user clicks on any of the records visible in the second tab of the **DirectDebitAUDDIS** form, a new form opens. This form contains all the information for a particular mandate, imported from the file. As the name suggests, all the fields are in a read-only state. By clicking on the **View Mandate** button, the user will be redirected to the actual mandate page.

## Entity FTOS\_PYMT\_DIDEMandateInstruction

#### **View default**

This view displays the list of all the DIDE UK mandate instruction files generated by the **DIDEUK\_Instructions** scheduled job. From this view, the user double-clicks the **FTOS\_PYMT\_DIDEMandateInstructionReadOnly** form in order to see the details of the selected record.

#### **Journey FTOS\_PYMT\_DIDEMandateInstructionReadOnly form**



This form displays the details (file name and date of generation) of the DIDE UK mandate instruction files generated by the **DIDEUK\_Instructions** scheduled job. Also, from this form the user can download the generated file.

## On Demand Scripts

### FTOS\_PYMT\_DirectDebitNotification\_File\_Validation

On demand script that is triggered when a file is uploaded during either of the **FTOS\_PYMT\_ARUDDInsert** or **DirectDebitADDACSInsert** journeys. After the user presses the **Import Data** button, the script validates that the file is in **.txt** format. The script also restricts the upload to only one file.

**Input** parameters: **file** - The DIDE payments file that needs to be uploaded.

**Output** parameters: **fileValidation** (boolean) - The result of the validation (true/ false).

### FTOS\_PYMT\_DIDEInstructionFile

This on demand script calls the **runDIDEInstructionFileUK()** function from the **FTOS\_PYMT\_DIDE** server side script library, in order to generate the payments instructions file for direct debit payments.

**Input** parameters: N/A

**Output** parameters: N/A

### FTOS\_PYMT\_DIDE\_ADDACS

This on demand script calls the **processUKMandates** function from **FTOS\_PYMT\_Mandate** server side library in order to run different status changes for the mandates, according to the received updates. The following actions can be performed on the existing mandates: canceling, modifying or reactivating.

**Input** parameters: N/A.

**Output** parameters: N/A.

## FTOS\_DIDEUK\_Instructions

This on demand script calls the `runDIDEInstructionFileUK` function from the **FTOS\_PYMT\_DIDE** server side script library, in order to generate the payment instructions file for the UK direct debit mandates.

**Input** parameters: `mandates` - A string containing the file type.

**Output** parameters: N/A.

### Business Workflow Configuration Actions

**FTOS\_PYMT\_DirectDebitMandate** is the master business workflow that handles the different types of changes affecting a direct debit mandate during its lifetime. For more details about the mandate behavior, its states and the business workflow diagram consult the [Direct Debit](#) page.

Transition	Description
_Proposal	Initial state.
Active_Cancelled	When the system registers a notification about the mandate cancellation. The transition to this status is made according to the 0, 1, 2, 3, B, C <b>Reason Codes</b> . These values, received in the ADDACS file, are specific for mandates deletion.
Active_Expired	When the mandate reaches its end day. The expiration triggers the automatic change of the payment type on the policy, from Direct Debit to <b>Bank Transfer (OP)</b> .
Active_VersionClosed	When a version of the mandate is closed. Used for mandate versioning.
Approved_Active	When the mandate reaches its start day.
Approved_Draft	When a change of the <b>policy payment type</b> from Bank Transfer (OP) into <b>Direct Debit</b> is performed on a policy.
Approved_Expired	When the number of days for the activation of a mandate were exhausted.
Cancelled_Active	The transition to this status is made according to the <b>R</b> reason code, received in the ADDACS file. The <b>R</b> value is specific for mandates <b>reactivation</b> .

Transition	Description
Draft_Active	When the mandate reaches its start day. This transition is triggered by the <b>DIDEUK_Instructions</b> job, which verifies if the current date is the mandate begin date and changes the status to <b>Active</b> for all eligible mandates.
Draft_Cancelled	When the system registers a notification about the mandate cancellation. The transition to this status is made according to the 0, 1, 2, 3, B, C <b>Reason Codes</b> . These values, received in the ADDACS file, are specific for mandates deletion.
Draft_Expired	When the number of days for the activation of a mandate were exhausted. The expiration triggers the automatic change of the payment type on the policy, from Direct Debit to <b>Bank Transfer (OP)</b> .
Draft_Pending	After the instruction file for the mandate's activation is generated, and the mandate is pending approval from the bank.
Draft_VersionDraft	When the mandate versioning process starts.
Pending_Active	When the mandate reaches its start day.
Pending_Cancelled	When the system registers a notification (e.g. AUDDIS file) about canceling the mandate, or a user manually cancels the mandate.
Proposal_Active	When the mandate reaches its start day.
Proposal_Draft	When the mandate is registered in the system but it is not activated yet, or its begin date is yet to come.
Proposal_Expired	When the number of days for the activation of a draft mandate were exhausted.
VersionDraft_Approved	When a version of the mandate is approved. Used for mandate versioning.
VersionDraft_VersionUnapproved	When the opened version is not approved. Used for mandate versioning.

## Endpoints

### FTOS\_PYMT\_GenerateMandate

The **FTOS\_PYMT\_GenerateMandate** endpoint calls the **FTOS\_PYMT\_GenerateMandate** on demand script in order to generate a new UK direct debit mandate. This endpoint is described in the [Generate UK Direct Debit](#)

[Mandate API](#) page.

## Processors

The **FTOS\_PYMT\_DIDEProcessor** is used for setting the following parameters:

## DideConfiguration

This is an object containing the following settings:

- **numberOfDaysInAdvance** - This parameter sets the date for invoice generation with N days before the payment due date (for example, the invoice can be generated with 3 or 5 or 10 days before the payment due date).
- **runningOnBankHolidays** - This parameter determines whether the DIDE instruction file is generated on bank holidays or not (true or false). Set to true if the DIDE instruction file is to be generated on a bank holiday.
- **excludeWeekDays** - This parameter defines the week days in which the DIDE instruction file is not generated (for example, the invoice is not generated on Saturday or Sunday).
- **type** - This parameter sets the type of direct debit processing: either SEPA (for European area) or UK.

If the **type** option configured in the processor is **Sepa** - the DIDE process will take in consideration the functionality implemented for DIDE Sepa. Otherwise, the DIDE process will take into consideration the flow implemented for DIDE UK.

## DIDEPaymentsFileDefaultValues

This is an object containing static values for the UK DD Payments Instruction File. The values are:

- **destinationSort** - Destination sort
- **destinationAcct** - Destination acct

- `destinationType` - Destination type
- `userName` - Users name

## DIDEPaymentsFileDefaultValuesUk

This is an object containing static values for the DD Mandate Instructions file.  
The values are:

- `destinationSort` - Destination sort
- `destinationAcct` - Destination acct
- `destinationType` - Destination type
- `transaction` - Transaction
- `freeFormat` - Free Format
- `amount` - Amount
- `userName` - Users name

Below is an example of processor settings:

```

{
  "DideConfiguration":
  {
    "numberOfDaysInAdvance": 3,
    "runningOnBankHolidays": false,
    "excludeWeekDays": ["sunday", "saturday"],
    "type": "Uk"
  }
  "DIDEPaymentsFileDefaultValues":
  {
    "destinationSort": 202020,
    "destinationAcct": 22223333,
    "destinationType": 0,
    "usersName" : A bank name here
  }
  "DIDEPaymentsFileDefaultValuesUk":
  {
    "destinationSort": 202020,
    "destinationAcct": 22223333,
    "destinationType": 0,
    "transaction": "0N",
    "freeFormat": "000000",
    "amount": "000000",
    "usersName": A bank name here
  }
}

```

## Server Side Script Libraries

# FTOS\_PYMT\_Mandate

From the **FTOS\_PYMT\_Mandate** library, the following functions are used:

## PYMT\_Buffer

This function wraps a series of functions which perform actions on the **FTOS\_PYMT\_DIDE\_ADDACSDetail** entity or interpret data received by it. Inside the **PYMT\_Buffer** function, the following functions are used:

### updateUKBufferMandateId

This function updates the **directDebitMandateId** processed attributes on a buffer that triggered an action for a mandate (deletion, modification or reactivation).

**Input** parameters:

- **bufferId** - The Id of the buffer on which the update needs to be made.
- **mandateId** - The Id of the mandate on which the buffer performed an action.

**Output** parameters: N/A.

### performActionsOnMandateUK

This function detects the type of action that the buffer needs to perform on a mandate, based on the **stageld** that the buffer has. If there is no **stageld** set on the buffer, the function throws an exception, with an error message.

**Input** parameters:

- `bufferObj` - The object containing the data added in the buffer.
- `isJob` - (boolean) - Tells whether the function is called by a job or not.

**Output** parameters: `mandateId` - The Id of the mandate which was modified, cancelled, or reactivated.

## PYMT\_Mandate

Inside the `PYMT_Mandate` function, the following functions are used:

### updateMandateBusinessStatusId

This function updates the mandate business status.

**Input** parameters:

- `mandateId` - The mandate unique identifier.
- `statusId` - The business status unique identifier.

**Output** parameters: N/A.

### getMandateDetailsByReason

This function returns the mandate details based on **reference** or **account holder name** and **account number**.

**Input** parameters: `bufferObj` - The object containing the following attributes **reference**, or **accountHolder** and **accountNumber**.

**Output** parameters: Returns the fetch.

### getReasonDetailsByCode



This function returns the reason code details from **FTOS\_PYMT\_ADDACSReasonType** entity based on the reason code received as input.

**Input** parameters: **reasonCode** - The reason code (0, 1, 2, 3, B, C, D, E, R).

**Output** parameters: Returns the fetch.

## processUKMandates

This function makes status changes (canceling, modifying or reactivating) on the existing mandates.

**Input** parameters: N/A.

**Output** parameters: N/A.

## deleteMandateFromBufferUK

This function executes the following:

- It updates the business status of the mandate to **Cancelled**.
- It updates the **stageId** attribute to the one received from the buffer.
- It calls the **updateUKBufferMandateId** function to update the **directDebitMandateId** and **processed** attributes of the buffer.
- If no mandate is found based on the reference, the function throws an error.

**Input** parameters:

- **bufferObj** - The object containing the data added in the buffer.

- **isJob** - (boolean) - Tells whether the function is called by a job or not.

**Output** parameters: **mandateId** - The Id of the cancelled mandate.

## addMandateFromBuffer

This function executes the following:

- It inserts a new mandate record, with the data received from the **FTOS\_PYMT\_GenerateMandate** endpoint.
- If there is already a mandate with the same reference from the data object, the function throws an error.
- It calls the **updateUKBufferMandateId** function to update the **directDebitMandateId** and **processed** attributes of the buffer.

**Input** parameters: **bufferObj** - The object containing the data received from **FTOS\_PYMT\_GenerateMandate** endpoint.

**Output** parameters: **mandateId** - The Id of the mandate which was created.

## modifyMandateFromBuffer

This function executes the following:

- Creates a new version for a mandate with a specific reference.
- Updates the mandate with the new data received from the buffer.

- If no mandate is found having the same reference as the one received from the data object, the function throws an error.
- After the insert and update, the function updates the business status of the old mandate version to **versionClosed**.
- Calls the `updateUKBufferMandateId` function to update the `directDebitMandateId` and `processed` attributes of the buffer.

**Input** parameters:

- `bufferObj` - The object containing the data added in the buffer.
- `isJob` - (boolean) - Tells whether the function is called by a job or not.

**Output** parameters: `mandateId` - The Id of the modified mandate.

## reactivateMandateFromBufferUK

This function executes the following:

- Reactivates a cancelled mandate based on a specific reference, received through the buffer object.
- If no cancelled mandate or no mandate is found having the same reference as the one received from the data object, the function throws an error.
- Calls the `updateUKBufferMandateId` function to update the `directDebitMandateId` and `processed` attributes of the buffer.

**Input** parameters:

- `bufferObj` - The object containing the data added in the buffer.
- `isJob` - (boolean) - Tells whether the function is called by a job or not.

**Output** parameters: `mandateId` - The Id of the modified mandate.

## cancelMandateAndInvoices

This function sets the Id of all three business statuses that the function needs. Next, by calling the `updateFileMandateId()` function, we update the `directDebitMandateId` attribute of the buffer records. Next, the function checks if the found mandate is in one of the two statuses - either **Pending** or **Active**, and changes its business status to cancelled.

**Input** parameters:

- `bufferObj` - The object containing the data added in the buffer.
- `fileType` - The type of the file that has been imported.

**Output** parameters: N/A

## getStatementIdByPolicyNo

For a policy reference received as parameter, this function returns the statement in **Generated** or **On Grace** status and verifies if the payment method is set to direct debit.

**Input** parameters: `policyNo` - The policy number (string).

**Output** parameters: The result of the query.

## getPendingMandates

This function returns all the mandates that are in **Pending** business status.

**Input** parameters: N/A.

**Output** parameters: The query object.

## changeBusinessStatusUK

This function changes the business status for UK mandates.

**Input** parameters: N/A.

**Output** parameters: N/A.

## FTOS\_PYMT\_DIDE

From the **FTOS\_PYMT\_DIDE** library, the following functions are used:

### General

Inside the `General()` function, the following functions are used:

### isNullOrEmpty

This function verifies if an input value is **null** or **empty**.

**Input** parameters: `value` - The value that needs validation.

**Output** parameters: `true/ false` - The result of the validation.

## formatDateForFile

This functions transforms the format of an input date to the **ddmmyy** format. Next, it adds a number of days to the received date.

**Input** parameters:

`date`

`noOfDays` - How many days to add to the date received as an input.

**Output** parameters: Date formatted.

## getProcessorValues

This function returns the static values stored in the **FTOS\_PYMT\_DIDEProcessor** processor.

**Input** parameters: `setting` - The processor's object key name.

**Output** parameters: The object containing values from the processor:

- `destinationSort`
- `destinationAcct`
- `destinationType`
- `usersName`

- `transaction`
- `freeFormat`
- `amount`

## DIDE

This function wraps a series of functions which perform actions on the **FTOS\_PYMT\_DIDE** and **FTOS\_PYMT\_DIDE****Detail** entities, generate and validate some data or generate **.txt** files, as follows:

### runDIDEInstructionFile

This function executes the following:

- Verifies that the DIDE instruction file can be generated according the rules defined in the **FTOS\_PYMT\_DIDE****Processor** processor (`runningOnBankHolidays`, `numberOfDaysInAdvance`, and `excludeWeekDays`).
- Generates the details needed for the file type wanted.
- Saves the `statementId` and `directDebitMandateId` details into the **FTOS\_PYMT\_DIDE****Detail** entity.
- Saves the `directDebitMandateId` details in **FTOS\_PYMT\_DIDE****MandateInstructionDetail** entity, according to the file type.
- Generates the **.txt** file.

- Saves the file into the **FTOS\_PYMT\_DIDE** entity or into the **FTOS\_PYMT\_DIDEMandateInstruction** entity.
- It also saves the file on the server **UploadEBS** folder.

This function also uses all the functions mentioned below.

**Input** parameters: **instructionFileType** - Optional parameter. This parameter has two possible values:

1. **statements** - for files dealing with payments - DIDEPaymentsInstructionFile (for **SEPA**) and, respectively, DD\_BACSPayments file (for **UK**).
2. **mandates** - for files dealing with creation of new DIDE mandates - NEW\_DIDE\_Mandates\_BACSIInstructions file (for **UK**).

**Output** parameters: N/A.

## getProcessorDetails

This function gets the configuration details of a processor.

**Input** parameters:

- **flowSettingsName** - The name of the flow setting.
- **processorSettingsType** - The type of the **Digital Processor Type**.
- **processorSettingsName** - The name of the processor setting.

**Output** parameters: **processorDetails** - An object containing the processor settings **runningOnBankHolidays**, **excludeWeekDays** and **numberOfDaysInAdvance**.



## getDayName

This function returns the name of a weekday (ex. Monday).

**Input** parameters: `date` - The date parameter.

**Output** parameters: `dayName` - The name of the week day.

## getDIDERunningDay

This function verifies if the current day is a bank holiday or a day defined in the `FTOS_PYMT_DIDEProcessor`. If one condition is met the function returns a true value.

**Input** parameters: `settings` - The object with the processor settings - `runningOnBankHolidays`, `excludeWeekDays` and `numberOfDaysInAdvance`.

**Output** parameters: `True` or `false`.

## getDraftMandates

This function returns all the UK direct debit mandates in **Draft** status.

**Input** parameters: `mandateTypeUk` - The `mandateTypeId` has the **UK** value.

**Output** parameters: Returns the query result.

## getPolicyDataAPI

This function imports the `FTOS_PYMT_GetPolicyDataAPI` library in order to call the `FTOS_GetPolicyData_API` API. The function sends an object containing a list of policies as mandates references (where the `reference` is the **policy number**) and receives the information about the specified policies, in return. This function also parses the result.

**Input** parameters: `mandateObj` - The list of policies.

**Output** parameters: Information result about the specified policies.

## getStatementsDetails

This function returns statements (invoices) to be included in the DIDE instruction file, based on the following conditions:

- `businessStatus` of the statement = `Generated`
- `paymentType` of the statement = `directDebit`
- `businessStatus` of the direct debit mandate = `active`
- `scheduledDate` of the statement = `null`
- `dueDate` of the statement  $\leq$  current date + `numberOfDaysInAdvance`
- `businessStatus` of the policy = `Enforced` or `Suspended`
- `mandateTypeId` = SEPA or UK

**Input** parameters:

- `numberOfDaysInAdvance` - The number of days in advance for generating an invoice (for a policy installment), before the due date for the payment.
- `mandateTypeId` = SEPA or UK

**Output** parameters: `resultStatements` - Returns the fetch.

## processPaymentsInstructionSepa

This function is specific for SEPA direct debit. This function executes the following:

- Inspects the results of the `getStatementsDetails` function.
- Inserts the `statementId`, `directDebitMandateId`, and `name` details into the **FTOS\_PYMT\_DIDEDetail** entity.
- Populates the `CSVContentArray` with all the details necessary for the generation of the DIDE instruction file.
- Updates the `scheduledDate` from the **FTOS\_PYMT\_Statement** entity.

**Input** parameters:

- `result` - The fetch result of the `getStatementsDetails` (`numberOfDaysInAdvance`, `mandateTypeId`) function.
- `CSVContentArray` - An empty array.

**Output** parameters: N/A.

## processMandatesBACSInstructionUk

This function executes the following:

- Inspects the results of mandates returned by the query.
- Inserts the `directDebitMandateId` details into the **FTOS\_PYMT\_DIDEMandateInstructionDetail** entity.
- Populates the `CSVContentArray` with all the details necessary for generating the mandate instruction file.

- Sets the `beginDate` on the specified mandates.
- Sets the status on the mandates to **Active**.

In the `CSVContentArray` file, the information is concatenated as follows:

- **First column:** destination sort + destination acct + destination type + transaction + bank Sort Code + account number.
- **Second column:** free format + amount + users name.
- **Third column:** reference + payer last name.
- **Fourth column:** BACS Processing day.

**Input** parameters:

- `resultMandates` - The mandates that respect all conditions for being included in the file.
- `CSVContentArray` - An empty array.

**Output** parameters: N/A.

## processPaymentsInstructionUK

This function executes the following:

- Inspects the results of the `getStatementsDetails` function.
- Inserts the `statementId`, `directDebitMandateId`, and `name` details into the `FTOS_PYMT_DIDEDetail` entity.

- Populates the `CSVContentArray` with all the details necessary for the generation of the UK\_DIDE instruction file.
- Updates the `scheduledDate` from the `FTOS_PYMT_Statement` entity

In the `CSVContentArray` file, the information is concatenated as follows:

- **First column:** destination sort + destination acct + destination type + transaction + bank Sort Code + account number.
- **Second column:** free format.
- **Third column:** amount + users name.
- **Fourth column:** reference + payer last name.
- **Fifth column:** BACS Processing day.

**Input** parameters:

- `result` - The fetch result of the `getStatementsDetails(numberOfDaysInAdvance, mandateTypeId)` function.
- `CSVContentArray` - An empty array.

**Output** parameters: N/A.

## getInstallmentNo

This function returns the installment number for an invoice reference, received as parameter.

**Input** parameters: `statementReference` - The invoice reference.

**Output** parameters: Returns the query result.

## generatePaymentsInstructionFile

This function executes the following:

- Generates the .txt DIDE instruction file based on `csvContentArray` populated by the `processPaymentsInstruction(result, csvContentArray)` function.
- Saves the file into **FTOS\_PYMT\_DIDE** entity.
- Saves the file on the UploadEBS folder from the portal.

**Input** parameters:

`csvContentArray` - The array with all the records for direct debit instructions.

`instructionFileType` - Optional parameter, with two possible values: 1. `statements` for DIDEPaymentsInstructionFile (SEPA) and DD\_BACSPayments (UK) file and 2. `mandates` for NEW\_DIDE\_Mandates\_BACSIInstructions file (UK).

**Output** parameters:

`idInstructionFile` - The Id of the new file inserted in **FTOS\_PYMT\_DIDE** entity or **FTOS\_PYMT\_DIDEMandateInstruction** entity.

`entityName` - **FTOS\_PYMT\_DIDE** - for DIDEPaymentsInstructionFile (SEPA) and DD\_BACSPayments (UK) file OR **FTOS\_PYMT\_DIDEMandateInstruction** for the NEW\_DIDE\_Mandates\_BACSIInstructions file (UK).

## updateDIDEDetails

This function searches all the records from **FTOS\_PYMT\_DIDEDetail** or **FTOS\_PYMT\_DIDEMandateInstructionDetail** entities (depending on the file type) which have `dideId` (id of the file) with `null` values and updates them with the id of the recently generated file.

**Input** parameters:

- `instructionFileObj` -
- `instructionFileId` - The id of the new generated file and `entityName` - the name of the entity.

**Output** parameters: N/A

## FTOS\_PYMT\_ARUDD

This server side script library contains methods that help in the process of generating an ARUDD unconfirmed payments file for UK type. From the **FTOS\_PYMT\_ARUDD** library, the `ReasonForReturn` function is used. Inside the `ReasonForReturn` function, the following generic methods are used:

### formatText

This function trims, transforms into lowercase and removes the spaces from a string.

**Input** parameters: `text` - string

**Output** parameters: `text` - formatted string

### reasonForReturn

For the correlated mandates which contain these denied payments with **Reason for return code = 1- Instruction cancelled** in the file, the function triggers the **Cancellation** process - making just the

transition of the Mandate from **Active** to **Cancelled**. The invoices correlated with the mandates which contain denied payments - and changed status to **Cancelled**, become **Cancelled**, also.

For the mandates which contain denied payments with **Reason for return code = 0 - Refer to payer** in the file, the correlated invoices that contain those denied payments become **Unpaid**, also.

**Input** parameters: **records** - The records inserted in FTOS\_PYMT\_ARUDDDetail.

**Output** parameters: The query result.

## Scheduled Jobs

### FTOS\_PYMT\_DIDE\_ADDACS

This job is scheduled to run every day at 5:15 AM, to perform status changes (canceling, modifying or reactivating) on the existing mandates, based on the unprocessed records from the buffer - the **FTOS\_PYMT\_DIDE\_ADDACS** entity.

Schedule Services: **FTOS\_PYMT\_DIDE\_ADDACS** - on demand server automation script.

### FTOS\_PYMT\_DIDEInstructionFile

The job is scheduled to run every day at 3:00 AM to generate the **.txt** instruction file. Depending on the configuration, it generates the **DIDEPaymentsInstructionFile** (for SEPA) and **DD\_BACSPayments** (for UK), also.

Schedule Services: **FTOS\_PYMT\_DIDEInstructionFile** - on demand server automation script.

### DIDEUK\_Instructions



The job is scheduled to run every day at 4:00 AM to generate the **.txt** DIDE UK mandates instruction file.

Schedule Services: **FTOS\_DIDEUK\_Instructions** - on demand server automation script.

## Outgoing Payments

The **Billing and Collection** functionalities let you add, approve, and pay different payment requests, manually or automatically. Since the solution differentiates between automatic outgoing payments and manual outgoing payments, you can also configure it to deal with recurring payment requests received from different sources - like **Commissions, Broker Balances (Credit)**, and more.

**FTOS\_PYMT\_OutgoingPayment** - the entity storing the outgoing payments requests, is the source of the **Payments Instruction File** which is generated by a scheduled job, daily, in order trigger the payments from the insurer's bank account. This makes the **Billing and Collection** solution useful for recurring outgoing payments scenarios, since it can automatically approve and allocate those types of outgoing payments. For other scenarios, you have an outgoing payment request manual approval flow that lets you approve or decline such requests, according to your needs.

See more details about handling **Outgoing Payments** in the following pages:

- [Outgoing Payments Admin](#) - for details about how outgoing payments are handled by the system.
- [Outgoing Payments Allocation](#) - for details about how the outgoing payments allocation and deallocation works.
- [Manual Outgoing Payment Requests](#) - for details about how the manual outgoing payments request are handled by the system.
- [Generate Outgoing Payments API](#) - for details about how this API works.

## Outgoing Payments Admin

The **Outgoing Payments Requests** functionality lets you add, approve, and pay different payment requests, manually or automatically.

### Data model

- Entity **FTOS\_PYMT\_Payment** with all attributes
- Entity **FTOS\_PYMT\_OutgoingPayment** with all attributes.

### FTOS\_PYMT\_Payment\_EditFormOutgoing Journey

General description:

This is a user journey aimed at implementing the **Edit Outgoing Payment** functionality, in order to manually allocate or deallocate an outgoing payment. This journey is connected to the **FTOS\_PYMT\_Payment** entity.

This form is used to manually allocate or deallocate an outgoing payment.

The user clicks **Allocate** to open the **Payment Request Search** grid. The user searches for the desired outgoing payments, inside the grid. For the search, the following parameters are available:

Search parameters:	Description
referenceNo	Reference No.
outgoingPaymentTypeSearchId	Outgoing Payment Type
amountMin	Payment Amount min
amountMax	Payment Amount max
payerNameSearch	Payment Beneficiary Name
ibanSearch	IBAN
startDueDateFrom	Start Due Date From
startDueDateTo	Start Due Date To

#### NOTE

At least one criteria has to be filled in. If not, a warning message is shown: "Warning! The fields should not be empty!".

In order to show the results, the following functions are used:

## isNullOrEmpty(value)

This function verifies if an input value is null or empty.

**Input** parameters: `value`.

**Output** parameters: `true/ false`.

## getPaymentRequestSearch

This function gathers all the payment requests found based on the search criteria with the help of the `FTOS_INSP_ManualAllocatePaymentRequestSearch` automation script. It returns the results in a grid with the help of the `generateCustomGrid(viewId, viewDataSource, viewColumns)` function. The payment requests included in the results of the search comply with the following conditions:

- are in **Approved** status;
- have the same currency as the payment request;
- have the same IBAN account.

**Input** parameters:

- `paymentCurrencyId` – The payment currency.
- `paymentIban` – The payment IBAN.

**Output** parameters: N/A.

## allocateClickFunction

This function allocates a payment request on an outgoing payment, using the automation script `FTOS_PYMT_ManualOutgoingPaymentAllocation`.

**Input** parameters: `details` – (Object) – The needed data from the `generateCustomGrid(viewId, viewDataSource, viewColumns)` function, to be used for manually allocating a payment.

**Output** parameters: N/A.

## generateCustomGrid

This function generates the results grid.

**Input** parameters:

**viewId** – The name of the CSS Id.

**viewDataSource** – The data result from the **getPaymentRequestSearch** (**paymentCurrencyId**, **paymentIban**) function.

**viewColumns** – The type of columns that are displayed in the grid.

**Output** parameters: N/A.

## FTOS\_PYMT\_OutgoingPaymentFileReadOnly Form Journey

This is a user journey connected to the **FTOS\_PYMT\_OutgoingPaymentFile** entity.

The **Default View** allows the user to see a list of all the outgoing payments instruction files generated by the **FTOS\_PYMT\_OutgoingPaymentInstructionFile** scheduled job. From this view, the user can select a record, double-click on it to launch the **FTOS\_PYMT\_OutgoingPaymentFileReadOnly Form**, to see the file details. The form allows the user to see the details (file name and date of generation) of the outgoing payments instruction files generated by the **FTOS\_PYMT\_OutgoingPaymentInstructionFile** scheduled job. Also, the form allows the user to download the generated file.

## On Demand Scripts

**FTOS\_PYMT\_OutgoingPaymentInstructionFile** - this script calls the **runOutgoingPaymentInstructionFile** function from the **FTOS\_PYMT\_OutgoingPaymentFiles** server side script library in order to generate the instructions file for outgoing payments.

**Input** parameters: N/A.

**Output** parameters: N/A.

## Endpoints

The following endpoints are used for matching between outgoing payment requests and outgoing payments:

## FTOS\_INSP ManualAllocatePaymentRequestSearch

This endpoint calls the `FTOS_INSP_ManualAllocatePaymentRequestSearch` script to search the outgoing payment requests to be allocated on an outgoing payment.

## FTOS\_PYMT ManualOutgoingPaymentAllocation

This endpoint uses the `FTOS_PYMT_OutgoingPaymentAllocation_manual` library in the manual process of outgoing payment allocation.

### Server Side Script Libraries

## FTOS\_PYMT\_OutgoingPaymentAllocation\_ manual

From the `FTOS_PYMT_OutgoingPaymentAllocation_manual` library, the `PYMT_OutgoingPayment` function is used. Inside this function, the following functions are used:

### getPaymentAmount

This function gets the payment amount, the allocated amount and the remaining payment amount for a payment.

**Input** parameters: `paymentId` - The unique identifier of the payment.

**Output** parameters: Returns the results of the fetch.

### getOutgoingPayments

This function gets the payment amount, paid amount and the unpaid payment amount for an outgoing payment.

**Input** parameters: `outgoingPaymentId` - The unique identifier of the outgoing payment.

**Output** parameters: Returns the results of the fetch.

## manualOutgoingPaymentAllocation

If the remaining payment amount is greater than 0, this function calls the `insertOutgoingPaymentAllocation` function, from the `FTOS_PYMT_OutgoingPaymentAllocation` library.

**Input** parameters:

- `paymentId` - The unique identifier of the payment.
- `outgoingPaymentId` - The unique identifier of the outgoing payment.

**Output** parameters:

Returns an error message (Error in allocation process!) if there is an error in the allocation process.

Returns an error message (Payment amount fully allocated!) if the payment amount was fully allocated.

## FTOS\_PYMT\_OutgoingPaymentFiles

From the `FTOS_PYMT_OutgoingPaymentFiles` library, the `General` function is used. This function wraps up other functions which perform actions on the `FTOS_PYMT_OutgoingPaymentFile` and `FTOS_PYMT_OutgoingPaymentFileDetail` entities, generate and validate some data or generate the necessary .txt file.

## runOutgoingPaymentInstructionFile

This function executes the following:

- verifies that the outgoing payments instructions file can be generated according any defined rules - for example outgoing payments are in **Approved** status and their **Scheduled** date equals the current date.
- generates the details needed for the outgoing payments instruction file.

- saves the `outgoingPaymentId` and `name` in `FTOS_PYMT_OutgoingPaymentFileDetail` entity.
- generate the outgoing payments instruction file (in `.txt` format).
- saves the file in `FTOS_PYMT_OutgoingPaymentFile` entity.
- saves the same file on the server `OutgoingPayment` folder.

The `runOutgoingPaymentInstructionFile` function uses all the functions mentioned below.

**Input** parameters: N/A.

**Output** parameters: N/A.

## getOutgoingPaymentDetails

This function returns the outgoing payments to be included in the outgoing payments instruction file based on the following conditions: the `businessStatus` of the outgoing payment is **Approved** and the `paymentScheduledDate` is the **Current Date**.

**Input** parameters: N/A

**Output** parameters: `resultOutgoingPayments` - Returns the fetch.

## processOutgoingPaymentsInstruction

This function executes the following:

- processes the results of the `getOutgoingPaymentDetails` function.
- insert details (`outgoingPaymentId` and `name`) into the `FTOS_PYMT_OutgoingPaymentFileDetail` entity.
- populates `CSVContentArray` with all the details that are necessary for the outgoing payments instruction file.

- changes the business status from the **FTOS\_PYMT\_OutgoingPayment** entity.

**Input** parameters:

- `result` - The fetch result of the `getOutgoingPaymentDetails` function.
- `CSVContentArray` - An empty array.

**Output** parameters: N/A.

## generateOutgoingPaymentsInstructionFile

This function executes the following:

- generates the outgoing payments instruction file (in .txt format) based on the `CSVContentArray` populated by the `processOutgoingPaymentsInstruction` function,
- saves the file into **FTOS\_PYMT\_OutgoingPaymentFile** entity,
- saves the file on the **OutgoingPayment** folder from the portal.

**Input** parameters: `CSVContentArray` - An array with all the records for outgoing payments instructions.

**Output** parameters: `idInstructionFile` - The Id of the file inserted in **FTOS\_PYMT\_OutgoingPaymentFile** entity.

## updateOutgoingPaymentFileDetails

This function finds all the records from **FTOS\_PYMT\_OutgoingPaymentFileDetail** entity that have the `outgoingPaymentFileId` (the Id of the outgoing payments instruction file) with null values and updates that attribute with the Id of the newly generated outgoing payments instruction file.

**Input** parameters: `instructionFileId` - The Id of the newly generated outgoing payments instruction file.



**Output** parameters: N/A

## Scheduled jobs

# FTOS\_PYMT\_OutgoingPaymentInstructionFile

This job is scheduled to run every day at 5:00 AM to generate the outgoing payments instruction **.txt file**.

Schedule Services:

**FTOS\_PYMT\_OutgoingPaymentInstructionFile** - on demand Server Automation Script, see [above](#).

## Outgoing Payments Allocation

For [Outgoing Payments Deallocation](#) scroll down or click the link.

### Server Automation Scripts

The **FTOS\_PYMT\_OutgoingPaymentAllocation** server automation script uses the following functions:

## automaticOutgoingPaymentAllocation

Example of calling the function: `automaticOutgoingPaymentAllocation(paymentId)`.

Based on the `payment` and `outgoing payment`, this function gets and prepares all the necessary data for inserting and updating the outgoing payment and the payment.

**Input** parameters: `paymentId` - The Id of the payment.

**Output** parameters:

## getPayment

Example of calling the function: `getPayment(paymentId)`.

This function gets and returns the payment details based on the Id of the payment.

**Input** parameters: `paymentId` - The Id of the payment.

**Output** parameters: `payment[0]` - The query result containing details of the payment requested through the input parameter.

## findOutgoingPayments

Example of calling the function: `findOutgoingPayments(payment, processor)`

This function is meant to return all outgoing payments.

**Input** parameters: `payment` and `processor`.

**Output** parameters: `outgoingPaymentList` - The list of outgoing payments that comply with the references contained in the processor.

## getOutgoingPaymentReferenceValues

Example of calling the function: `getOutgoingPaymentReferenceValues(payment, processor)`

This function returns an object containing all the references to be applied.

**Input** parameters: `payment` and `processor`.

**Output** parameters: `result`.

## getOutgoingPaymentsByConditions

Example of calling the function: `getOutgoingPaymentsByConditions('paymentReferenceRule', referenceValues, processor)`.

Based on the payment type, reference rules and processor, this function returns a list of outgoing payments that are compliant with the references contained into the processor input parameter.

**Input** parameters:

- `type` - string, rule name
- `whereValuesObj` - reference rules
- `processor`

**Output** parameters: `outgoingPaymentList` - The list of outgoing payments that comply with the references contained in the processor.

## insertOutgoingPaymentAllocation

Example of calling the function: `getOutgoingPaymentsByConditions('paymentReferenceRule', referenceValues, processor)`

This function makes all the necessary updates for **FTOS\_PYMT\_OutgoingPayment** and **FTOS\_PYMT\_Payment** entities. It also inserts a record into **FTOS\_PYMT\_OutgoingPaymentRequestXPayment** entity - the relationship table between **FTOS\_PYMT\_OutgoingPayment** and **FTOS\_PYMT\_Payment** entities, and returns the Id of the inserted record.

**Input** parameters:

- `payment`
- `outgoingPayment`

**Output** parameters: `outgoingPaymentRequestXPaymentId`

## Outgoing Payment Deallocation

For deallocation, the following functions are used:

### deallocateOutgoingPayment

Example of calling the function: `deallocateOutgoingPayment(outgoingPaymentId, paymentId)`.

This function updates the **FTOS\_PYMT\_OutgoingPayment** entity and changes the business status into **Deleted** for the **FTOS\_PYMT\_OutgoingPaymentRequestXPayment** item.

**Input** parameters:

- `outgoingPaymentId` - The Id of the outgoing payment.
- `paymentId` - The Id of the payment.

**Output** parameters: N/A.

## deallocatePayment

Example of calling the function: `deallocatePayment(paymentId, paymentUpdateObj)`.

Use to update the **FTOS\_PYMT\_Payment** entity

**Input** parameters:

- `paymentId` - The Id of the payment.
- `paymentUpdateObj` - An object containing the update.

**Output** parameters: N/A.

## getOutgoingPaymentsRequestXPayment

Example of calling the function: `getOutgoingPaymentsRequestXPayment(outgoingPaymentId, paymentId)`.

This function gets and returns the **outgoingPaymentRequestXPayment** details in allocated status, based on the `outgoingPaymentId` and `paymentId`.

**Input** parameters:

- `outgoingPaymentId` - The Id of the outgoing payment.
- `paymentId` - The Id of the payment.

**Output** parameters: `fetchResult[0]` - query result containing details of the outgoing payment requested through the input parameter.

## getOutgoingPayment

Example of calling the function: `getOutgoingPayment(outgoingPaymentId)`.

This function gets and returns the outgoing payment details based on the Id of the outgoing payment.

**Input** parameters: `outgoingPaymentId` - The Id of the outgoing payment.

**Output** parameters: `outgoingPayment[0]` - The query result containing details of the the outgoing payment, requested through the input parameter.

## getPayment

Example of calling the function: `getPayment(paymentId)`.

Function meant to get and return the payment details based on the payment ID

**Input** parameters: `paymentId` - The Id of the payment.

**Output** parameters: `payment[0]` - The query result containing details of the payment, requested through the input parameter.

## Manual Outgoing Payment Requests

This functionality lets you manually add, approve or pay the **Outgoing Payments Requests** received from different sources or systems. Once added, the system automatically populates the **referenceNo** attribute for the new outgoing payment request, by using a sequencer.

## Data model

Entity **FTOS\_PYMT\_OutgoingPayment** with all attributes.

### Form Client Side Scripts

In order to show the results, the following functions are used:

## paymentReqReadOnly

This function sets some form fields to **ReadOnly**, based on the entity business status.

**Input** parameters: N/A.

**Output** parameters: N/A.

## hideStepsOnStatus

This function hides some form steps depending on the business status or if the form is in **Edit** mode.

**Input** parameters: N/A.

**Output** parameters: N/A.

## getBank

This function performs the following actions:

- Gets the bank details based on the bank code, from the **iban** parameter.
- Checks the result and throws a warning message if the validity condition (bank code) is not met.
- If there is no result the function displays a warning message.
- Upon a valid result, the function sets the **bankId**, to be used later on for completing the form field.

**Input** parameters:

- `iban` - The string of characters introduced in the IBAN form field.
- `ibanType` - The type of the IBAN, used for differentiating between the beneficiary details and the payer ones.

**Output** parameters: N/A.

## completeBankOnIban(iban)

Depending on the `ibanType`, this function calls the previous function `getBank` with a different `iban` parameter for getting the correct bank details.

**Input** parameters: `iban` - The type of the IBAN - let's you know which form field you get the value from.

**Output** parameters: N/A.

## detectBankOnIban

This function calls the previous function `completeBankOnIban` at step loading and also on a form field change, for the IBAN.

**Input** parameters: N/A.

**Output** parameters: N/A.

## callCompletePayerDetailsEndpoint

This function executes the following actions:

- Calls the `FTOS_PYMT_OutgoingPayment_CompletePayerDetails` endpoint;
- Sends an object to this endpoint, containing the `currencyId`;
- If the response from the endpoint is different than `null`, the function sets some form fields with the corresponding data.

**Input** parameters: N/A.

**Output** parameters: N/A.

## getPayerDetails

This function only calls the previous function `callCompletePayerDetailsEndpoint` on a form field change(`currencyId`).

**Input** parameters: N/A.

**Output** parameters: N/A.

## requestsButtons

This function hides or shows the proposal and cancellation buttons depending on the entity business status.

1. The **Proposal** button functionality includes the following actions:

- Saves the data introduced in the form fields.
- Calls the `ProposePaymentReturnValidation` endpoint.
- Redirects the user back to the current request, if the case.
- Reloads the page, if the form is in edit mode.

2. The **Cancellation** button calls the `FTOS_INSP_Payment_cancelPaymentReturn` endpoint and refreshes the page, after the response was received.

**Input** parameters: N/A.

**Output** parameters: N/A.

## getBeneficiaryFullName

This function gets the full name of the outgoing payment's beneficiary by calling the `FTOS_INSP_GetAccountData_UniqueIdNo` endpoint.

**Input** parameters: N/A.

**Output** parameters: N/A.

## Endpoints

### FTOS\_INSP\_GetAccountData\_UniqueIdNo



The endpoint fetches the account details from the **Account** entity based on the unique identifier - **PIN** for an individual person and **FiscalRegistrationNo** for a legal person.

After the fetch, the endpoint uses the `setData` method to send the obtained data to the client side for populating the required fields. The endpoint calls the `getAccountDataByUniqueIdNo` function from the **FTOS\_GetAccountData** server side script library.

## ProposePaymentReturnValidation

This endpoint performs two different actions on an outgoing payment:

- Change the business status into **Proposed**.
- Update the `proposalDate` attribute of the outgoing payment with the current date.

## FTOS\_INSP\_Payment\_cancelPaymentReturn

This endpoint is used for changing the outgoing payment business status.

## FTOS\_PYMT\_OutgoingPayment\_CompletePayerDetails

This endpoint is called when the **Payment No** field changes to get data from the **FTOS\_PYMT\_Payment** entity based on it's value. The endpoint uses the obtained values to populate the **Payer Details** section fields.

### Server Side Script Libraries

#### FTOS\_GetAccountData

This library contains functions for fetching data from or inserting data into the **Account** entity, based on different conditions. From this library, the following function is used:

### getAccountDataByUniqueIdNo

This function runs a fetch on the **Account** entity to retrieve some attributes, based on the unique identifier value.

**Input** parameters: `uniqueIdentifier` - The unique identifier for each account, based on the account type - **PIN** for an individual person and **FiscalRegistrationNo** for a legal person.

**Output** parameters: `acc[0]` - The first account found by the fetch containing the required details.

### FTOS\_OutgoingPaymentRequest

This library contains functions for inserting, updating and manipulating data used in the outgoing payment request process. From this library, the following functions are used:

## updateOutgoingPaymentRequest

This function updates some attributes from the **FTOS\_PYMT\_OutgoingPayment** entity for an outgoing payment with a specific Id.

**Input** parameters:

- `requestId` - The Id of the outgoing payment request.
- `paymentNo` - The value of the payment number field filled in at a new insert.

**Output** parameters: N/A.

## updateApprovedByUser

The function updates the `approvedBy` and the `approvedBy_displayname` attributes for an outgoing payment based on a specific Id.

**Input** parameters:

- `requestId` - The Id of the outgoing payment request.
- `userId` - The Id of an existing user.
- `userName` - The full name of an existing user.

**Output** parameters: N/A.

## updatePaidUnpaidAmount

The function updates the `paidAmount` and the `unpaidAmount` attributes for an outgoing payment based on the specified Id.

**Input** parameters:

- `requestId` - The Id of the outgoing payment request.
- `paymAmount` - The value of the payment amount field, received from the client-side.

**Output** parameters: N/A.

## paymScheduledDateValidation

The payment scheduled date inserted in a request form for an outgoing payment has to be greater than or equal to the current date. If the scheduled date is less than the current date, the system throws an error.

**Input** parameters: `sheduledDate` - The date object containing the details about the outgoing payment schedule.

**Output** parameters: N/A.

## fetchAccountOnId

The function fetches, based on the Id, some details from the **Account** entity for an account.

**Input** parameters: `accId` - The Id of the account.

**Output** parameters: `accounts` - The list containing the account that has been found by the fetch.

## insertAccountNew

If there is no account with a specific `uniqueIdNo`, the function gets the Id of the selected account type and inserts that Id on the **Account** entity, along with a set of attributes.

**Input** parameters:

- `paymBenId` - The Id of an account.
- `name` - The full name of the beneficiary.
- `firstName` - The first name of the beneficiary.
- `lastName` -The last name of the beneficiary.
- `beneficiaryType` -The Id of the selected optionset item .
- `uniqueIdNo` - The unique identifier for an account.

**Output** parameters: N/A.

## updateAccountIfChangedValues

This function checks if an existing account is going to be updated by comparing the stored data with the data received from the form. If the new data is different, the function goes ahead and updates the attributes for the selected account.

**Input** parameters:

- `paymBenId` -The Id of an account.
- `name` -The full name of the beneficiary.
- `firstName` -The first name of the beneficiary.
- `lastName` -The last name of the beneficiary.

**Output** parameters: N/A.

## fetchOutPaymentOnId

The function contains a fetch performed on the **FTOS\_PYMT\_OutgoingPayment** entity based on a specific Id.

**Input** parameters: **opId** -The Id of an outgoing payment.

**Output** parameters: **outgoingPayment** - The list containing the outgoing payments found by the fetch.

## fetchPaymentOnId

The function contains a fetch performed on the **FTOS\_PYMT\_Payment** entity based on a specific Id.

**Input** parameters: **pId** -The Id of a payment.

**Output** parameters: **payment** -The list containing the payments found by the fetch

## cancelOutgoingPaymentRequest

The function changes the business status of an outgoing payment with the one received as a parameter and if it finds a payment with a specific **sourcePaymentId**, it updates a couple of its attributes.

**Input** parameters:

- **paymId** -The Id of an outgoing payment.
- **paymAmount** - The numeric value of the payment amount.
- **newBS** - The new business status of the outgoing payment.

**Output** parameters: N/A.

## updateOutgoingPaymentBeneficiaryName

This function checks if an existing account name is going to be updated by comparing the stored data with the data received from the form. If the new data is different, the function goes ahead and updates the **beneficiaryName** attribute for the selected account.

**Input** parameters:

- `opReqId` - The Id of an outgoing payment request.
- `name` - The full name of the beneficiary.
- `firstName` - The first name of the beneficiary.
- `lastName` - The last name of the beneficiary.
- `beneficiaryCategoryId` - The Id of the account category type.

**Output** parameters: N/A.

## getPayerDetails

The function maps all the data inside a JSON processor.

**Input** parameters:

- `currency` - An object containing the currency details.
- `processor` - An object containing the data from the processor.

**Output** parameters: `result` - An object containing all the mapped data from the processor.

## getCurrencyDetails

The function contains the fetch performed on the `FTOS_CMB_Currency` entity based on a specific Id.

**Input** parameters: `currencyId` - The Id of a currency.

**Output** parameters: `currency[0]` - The currency found by the fetch.

## getPayerDetailsFromProcessor

This function gets the data from the `FTOS_PYMT_OutgoingPayments_PayerDetails` processor and identifies the data related to the currency code of the selected currency.

This function gets the payer data from the `FTOS_PYMT_OutgoingPayments_PayerDetails` processor based on the selected currency code.

**Input** parameters: `currencyId` - The Id of a currency.

**Output** parameters: `payerDetails` - An object containing the **payer name** and the **IBAN**.

## On Demand Scripts

### FTOS\_PYMT\_UpdateDeclineReason

This script updates the `paymentApprovalComments` attribute with the text written in the reason pop-up.

## Billing And Collection Endpoints

There are cases when you need to access the product data or Insurance Product Factory functionalities in a way alternative to the one available as a user journey, in Innovation Studio. The solution allows you to make API calls regarding different product-related aspects. For interacting with defined products, the following endpoints are available:

- [FTOS\\_PYMT\\_StatementsGenerationAPI](#) - Use this API to generate invoices for specific policies and installments. All the statement (invoice) generation business conditions will be applied when using the API as well. After calling the API, the invoice reference will be included in the response.
- [FTOS\\_PYMT\\_AddOutgoingPaymentRequest](#) - Use this API to insert a new outgoing payment request, for different types of payments. After validating all the input keys, a new request is inserted, and the reference of the payment returned in the response.

- **FTOS\_PYMT\_GenerateMandate** - Use this API to generate a new direct debit mandate for UK. The API handles the validation of the input keys as well as the logic for inserting the mandate, first into the **FTOS\_PYMT\_DirectDebitMandateBuffer** entity, and then, after processing the details, into the **FTOS\_PYMT\_DirectDebitMandate** entity.

Apart from the above mentioned endpoints, the following is the list of all the other endpoints available with the **Billing and Collection** solution:

Name	Script	Description
ButtonActionPaymentReturn	ButtonActionPaymentReturn	
FTOS_AllocationInstallmentSearch	FTOS_AllocationInstallmentSearch	
FTOS_CreateTestPayUPayment	FTOS_CreateTestPayUPayment	
FTOS_INSP_GetAccountData_UniqueIdNo	FTOS_INSP_GetAccountData_UniqueIdNo	
FTOS_INSP_ManualAllocatePaymentRequestSearch	FTOS_INSP_ManualAllocatePaymentRequestSearch	This endpoint calls the FTOS_INSP_ManualAllocatePaymentRequestSearch script to search the outgoing payment requests to be allocated on an outgoing payment.
FTOS_INSP_ManualAllocationSearch	FTOS_INSP_ManualAllocationSearch	
FTOS_INSP_Payment_cancelPaymentReturn	FTOS_INSP_Payment_cancelPaymentReturn	
FTOS_INSP_Payment_declinePaymentReturn	FTOS_INSP_Payment_declinePaymentReturn	
FTOS_PayU_ImportBdx	FTOS_PayU_ImportBdx	
FTOS_PYMT_DeallocateOutgoingPayment	FTOS_PYMT_DeallocateOutgoingPayment	
FTOS_PYMT_DeallocateStatement	FTOS_PYMT_DeallocateStatement	
FTOS_PYMT_DeallocateStatementDetail	FTOS_PYMT_DeallocateStatementDetail	
FTOS_PYMT_DirectDebitMandate_cancelMandate	FTOS_PYMT_DirectDebitMandate_cancelMandate	



Name	Script	Description
FTOS_PYMT_ DirectDebitNotification_ File_Validation	FTOS_PYMT_ DirectDebitNotification_ File_Validation	
FTOS_PYMT_ GenerateStatementOnInstal lment	FTOS_PYMT_ GenerateStatementOnInstal lment	
FTOS_PYMT_ InsertStatementDueDate	FTOS_PYMT_ InsertStatementDueDate	
FTOS_PYMT_ ManualOutgoingPaymentAll ocation	FTOS_PYMT_ ManualOutgoingPaymentAll ocation	
FTOS_PYMT_ ManualPaymentAllocation	FTOS_PYMT_ ManualPaymentAllocation	
FTOS_PYMT_ OutgoingPayment_ ApprovalAttributes		
FTOS_PYMT_ OutgoingPayment_ ChangeBusinessStatusToAp proved	FTOS_PYMT_ OutgoingPayment_ ChangeBusinessStatusToAp proved	
FTOS_PYMT_ OutgoingPayment_ CompletePayerDetails	FTOS_PYMT_ OutgoingPayment_ CompletePayerDetails	
FTOS_PYMT_ OutgoingPaymentAutomatic Allocation	FTOS_PYMT_ OutgoingPaymentAutomatic Allocation	This endpoint uses the FTOS_PYMT_ OutgoingPaymentAllocatio n_manual library in the manual process of outgoing payment allocation.
FTOS_PYMT_ PaymentAutomaticAllocatio n	FTOS_PYMT_ PaymentAutomaticAllocatio n	
FTOS_PYMT_Statement_ EditStatus	FTOS_PYMT_Statement_ EditStatus	
FTOS_PYMT_Statement_ OnGrace_and_Unpaid_ status	FTOS_PYMT_Statement_ OnGrace_and_Unpaid_ status	
FTOS_PYMT_ StatementDetail_ RemoveInstallment	FTOS_PYMT_ StatementDetail_ RemoveInstallment	
FTOS_PYMT_ ValidateExternalRevenue	FTOS_PYMT_ ValidateExternalRevenue	

Name	Script	Description
FTOS_PYMT_validatePaymentReturn	FTOS_PYMT_validatePaymentReturn	
ProposePaymentReturnValidation	ProposePaymentReturnValidation	

## Generate Statements API

This functionality adds new billing statements (invoices) in the system, based on API requests received from other **FintechOS** solutions - such as **Policy Admin** or from other systems. On the **FTOS\_PYMT\_Statement** entity, for the newly generated invoice, the API response populates the following attributes:

- Statement Id.
- Statement Reference.
- Statement Premium Amount.
- Total Taxes.
- Statement Amount.
- Statement Due Date.
- Payment Type Id.

## Generate Statements API Example

The system registers a new invoice, containing installments from three different policies.

```

1 let policiesData = {
2   policies: [{
3     "policyNo": "80000863",
4     "installmentNo": 7
5   },
6   {
7     "policyNo": "80000862",
8     "installmentNo": 11
9   },

```

```

10     {
11         "policyNo": "80000861",
12         "installmentNo": 2
13     }
14 ],
15     statementDueDate: "2021-10-30"
16 }
17
18 ebs.callActionByNameAsync('FTOS_PYMT_
StatementsGenerationAPI', policiesData).then(function(e)
{
19     console.log(e)
20 });

```

## Request Data Parameters

The following is the list of data parameters included in the request:

Parameter	Description
policies	An array with the policy request details: <code>policyNo</code> and <code>installmentNo</code> .
statementDueDate	The invoice invariant date, when the payment is due (string).

## Response

The following is an example of a response:

```

1  {
2      UIResult: {
3          ...},
4      Message: null,
5      IsSuccess: true,
6      ClientScript: null,
7      Serialized: '{"UIResult":
{"NavigateToEntityPage":false,"Navigat...ientScript":null,"
Serialized":null,"ErrorCode":0}',
8      ...
9  }
10 ClientScript: null
11 ErrorCode: 0
12 IsSuccess: true
13 Message: null

```

```

14 Serialized: {
15     UIResult: {
16         NavigateToEntityPage: false,
17         NavigateToEntityPageOnEdit: false,
18         NavigateToEntityFormName: null,
19         NavigateToEntityName: null,
20         NavigateToEntityId: null,
21         NavigateToEntityInsertDefaults: null,
22         NavigateToUrl: null,
23         DownloadFile: null,
24         ReloadPage: false,
25         Message: null,
26         IsSuccess: false,
27         Data: {
28             "isSuccess": true,
29             "result": [{
30                 "statementId": "280f5f20-676c-4f74-
b1c4-121d9bd42ec4",
31                 "statementNo": "REF0003383",
32                 "statementPremiumAmount": 10.73,
33                 "totalTaxes": 1.07,
34                 "statementAmount": 10.73,
35                 "paymentTypeId": "180921d6-345a-4d83-
a37b-278569008e7c",
36                 "dueDate": {
37                     "invariantDate": "2021-11-11"
38                 }
39             },
40             {
41                 "statementId": "0894a6f8-69b0-467c-
9991-6b6f22f19d33",
42                 "statementNo": "REF0003384",
43                 "statementPremiumAmount": 10.73,
44                 "totalTaxes": 1.07,
45                 "statementAmount": 10.73,
46                 "paymentTypeId": "180921d6-345a-4d83-
a37b-278569008e7c",
47                 "dueDate": {
48                     "invariantDate": "2021-11-11"
49                 }
50             }
51         ],
52         "errorMessage": null,
53         "errorCode": null
54     },

```

```

55     NavigateToPageNo: null
56   },
57   Message: null,
58   IsSuccess: true,
59   ClientScript: null,
60   Serialized: null,
61   ErrorCode: 0
62 }
63 UIResult:
64   Data: {
65     isSuccess: null,
66     errorMessage: null,
67     errorCode: null,
68     result: {
69       ...}
70   }
71 DownloadFile: null
72 IsSuccess: false
73 Message: null
74 NavigateToEntityFormName: null
75 NavigateToEntityId: null
76 NavigateToEntityInsertDefaults: null
77 NavigateToEntityName: null
78 NavigateToEntityPage: false
79 NavigateToEntityPageOnEdit: false
80 NavigateToPageNo: null
81 NavigateToUrl: null
82 ReloadPage: false[[Prototype]]: Object[[Prototype]]:
  Object

```

Response description:

Key	Description
errorCode	Error code.
isSuccess	Marks weather the request is successful or not.
errorMessage	Error message.

Key	Description
result	<p>Either <code>null</code> or an <code>array</code> of objects, containing the following:</p> <ul style="list-style-type: none"> <li>• Statement Id</li> <li>• Statement Reference</li> <li>• Statement Premium Amount</li> <li>• Total Taxes</li> <li>• Statement Amount</li> <li>• Statement Due Date</li> <li>• Payment Type Id</li> </ul>

## Error Messages

The following are the error messages that can be encountered during the API invoice generation process:

Code	Text	Description
ERR.BL.50101	Invalid request!	No data provided or the <code>policyData</code> doesn't have the <code>policies</code> property.
ERR.BL.50102	Invalid request! <b>PolicyNo</b> or <b>InstallmentNo</b> can not be empty!	This error appears when the provided data fields are empty.
ERR.BL.50103	Invalid request! Policy {0} inserted multiple times!	This error appears when the same <code>PolicyNo</code> is included multiple times inside the <code>policies</code> array.
ERR.BL.50104	Invalid request! One or more policies do not have valid installments!	A valid policy installment is one that has the installment amount <b>greater than 0</b> , the installment due date <b>not null</b> and the installment business status <b>OnTime</b> .

Code	Text	Description
ERR.BL.50105	The policies from the request can not be included in the same statement!	There is a mismatch between one (or some) of the requested policies and the policy grouping rule. For generating a <b>multiple policies</b> invoice, policies need to have <b>the same contractor</b> , payment type, currency, WO product configurations, and due date.
ERR.BL.50106	Invalid request! Policy {0} is in {1} status!	This error appears when the invoice generation API request is made for a policy that does not have a valid status.
ERR.BL.50107	The due date of the statement can not be from the past!	This error appears when the provided data field is filled with a past statement due date.
ERR.BL.50108	Invalid request! Policy {0} does not exist!	The requested policy is not registered in the system.
ERR.BL.50109	The statement can not be generated for policies that have broker collection as payment type!	The invoice cannot be generated for policies that have broker collection as payment type.

## Endpoint

The **FTOS\_PYMT\_StatementsGenerationAPI** endpoint is used to run the **FTOS\_PYMT\_StatementsGenerationAPI** server automation script.

## Server Automation Script

**FTOS\_PYMT\_StatementsGenerationAPI** - Script called with a data object.

This script executes the following:

- Validates the request data object.
- Searches, verifies and returns the policies that are eligible for invoice generation.

- If all the necessary conditions are met, it uses the details gathered to generate the new invoice.

The script contains the following functions:

## statementsGenerationAPI

This function validates the data object. For example if an invoice due date is provided, it must be greater than the current day. (If the invoice due date is not included in the request, the system assigns to it the installment due date's value.) After checking the **payment type** for each policy, this function calls the `getStatementGenerationRules` function from the **FTOS\_PYMT\_Statements** server automation script library in order to identify whether there is any **single rule** payment type amongst a **multiple policies** grouping. If a mismatch is found, the system throws an error (see error ERR.BL.50105, above).

**Input** parameters: `policiesData` - The request object.

**Output** parameters: `rez` - The result object.

## searchPolicies

This function verifies whether the policy is registered in the system, has the installment amount **greater than 0**, has the installment due date **not null**, has the installment business status **OnTime**. Next, the function returns an array of objects with all policies found, that are suitable for invoice generation.

This function also verifies if the request is not duplicated. If it finds that the details of the same policy are duplicated inside the same request, it throws an error.

**Input** parameters: `policiesData` - The request object.

**Output** parameters: `getPolicies` / `rez` - Depending on the query result or validation.

## searchPolicyStatus



This function verifies whether the policy has a valid status - **Proposal, Issued, Enforced** or **Suspended** and returns a result containing the policy Id, policy No, and policy status.

**Input** parameters: `policyNo` (string) - The policy number.

**Output** parameters: `policy` - The query result.

## validatePolicies

For the case of generating an invoice for more than one policy, this function verifies whether the request complies to the **multiple policies inclusion rule** - that is the policies need to have the same contractor, same payment type, same currency, same WO product configurations, and the same due date.

**Input** parameters: `policies` - An array of objects containing the policies details.

**Output** parameters: `rez` - An object containing the result.

## getSinglePaymentTypeRule

Based on the single rule name, this function returns all payment types assigned to it.

**Input** parameters: N/A.

**Output** parameters: `singlePmtTypeArr` - An array with single rule payment types Ids.

## generateStatements

This function calls the **FTOS\_PYMT\_StatementsAPI** script library and, by using all the gathered data, it generates the billing statement (the invoice) for the selected policy (or policies).

**Input** parameters:

- `policies` - An array of objects containing details from the policies for which invoices are to be generated.
- `statementDate` (string) - The date of the invoice generation.

**Output** parameters: `rez` - An object containing the result of the operation - expected invoice.

## Server Automation Script Library

The `FTOS_PYMT_StatementsAPI` library validates and creates new invoices (statements) based on the received data from the `FTOS_PYMT_StatementsGenerationAPI` script (see above). This function also calls the `generateGroupedProducts(sgDay)` and `getStatementGenerationRules` functions from `FTOS_PYMT_Statements` server automation script library in order to validate and generate the invoices.

From this library, the `StatementGeneration` script is used. This script validates and creates new invoices based on the request object received from the `FTOS_PYMT_StatementsGenerationAPI` script. This script contains the following functions:

### insertStatements

This function prepares the rules for generating invoices, as follows:

- **multiplePolicies** - multiple policies are included in the same invoice, based on **the same** contractor, currency, payment type, product and due date.
- **singlePolicies** - only one policy is included in the invoice.

**Input** parameters:

- `invariantDate` - maxDate.
- `accountId` - null.

- `sgDay` (integer) - The number of days in advance before the payment's due date.
- `policiesArrAPI` - The array of policies for which invoices are going to be generated.
- `statementDueDateAPI` - The `date` in string format or `null`.

**Output** parameters: `returnArr` - An array of objects with the newly generated invoices, containing:

- Statement Id,
- Statement Reference,
- Statement Premium Amount,
- Total Taxes,
- Statement Amount,
- Statement Due Date,
- Payment Type Id.

## getStatementByPaymentType

This functions gathers and filters all the installments according to the rules, payment type and grouped products. Next, it creates the new invoice (statement) and invoice details (statement details).

**Input** parameters:

- `insertStatementObj` - An object containing the following data `ruleName` (string), `groupByPolicy` (boolean), `groupByQuote` (boolean) and `paymentTypeArr` (array of payment types).
- `installmentNo` - `null`.

- `groupedProductsArr` - An array of objects containing the following: `minDate` - currentDate, `maxDate` - maxDate and `productIds` - [], an empty array.
- `nonBroker` - null.

**Output** parameters: `returnArr` - An array of objects with the newly created invoices.

## getpaymentTypeIdArr

This function gets the Ids of the payment types, based on each payment type name.

**Input** parameters: `paymentTypeArr` - An array containing the payment type names.

**Output** parameters: `paymentTypeIdArr` - An array containing payment type Ids.

## installmentHasStatementDetail

Based on `policyNo` and `installmentNo` this function makes a query to check if the installment has any invoice details already created. Next, it checks the installment due date period, based on the `groupedProducts` or `invariantDate`.

**Input** parameters:

- `policyNo` - The number of the selected policy.
- `installmentNo` - The number of the policy installment, that needs to be paid.
- `groupedProducts` - Grouping rule.
- `invariantDate` - The date when the payment for the installment is due.

**Output** parameters: `groupedProducts` / `false` if query result has values or not.

## insertStatement

This function is used to insert a new invoice based on an array returned by query from the `getStatementByPaymentType` function.

**Input** parameters:

- `installmentGroupResult` - array returned by query from `getStatementByPaymentType` function
- `insertStatementObj` - An object containing the following data `ruleName` (string), `groupByPolicy` (boolean), `groupByQuote` (boolean) and `paymentTypeArr` (array of payment types).

**Output** parameters: `returnObj` - The object containing the invoice.

## getStatementNo

Based on the invoice Id, this function returns the number of the invoice - `statementNo`.

**Input** parameters: `statementId` - The Id of the invoice.

**Output** parameters: `statementNo` - The number of the invoice.

## getInstallment

This function gets all the installment data necessary for invoice generation. Next, it changes the status of the invoice into **Generated**.

**Input** parameters:

- `installmentGroupResult` - An array returned by query from `getStatementByPaymentType` function.
- `insertStatementObj` - An object containing the following data `ruleName` (string), `groupByPolicy` (boolean), `groupByQuote` (boolean) and `paymentTypeArr` (array of payment types).
- `statementId` - The Id of the invoice.

**Output** parameters: N/A.

## insertStatementDetail

Based on the **installment list**, this function can generate a new `statementDetail` (invoice). When so, it updates the **Statement Due Date**, on the installment. Next, this function changes the installment business status into **Statement Issued**.

**Input** parameters:

- `installmentsList` - An array of installments objects.
- `statementId` - The Id of the invoice.

**Output** parameters: `statementDetailArr` - An array containing the Id of the statementDetail/s.

## Generate Outgoing Payments API

This functionality allows you to add outgoing payment requests in the system, through an API call. Once added, the system automatically populates the **referenceNo** attribute for the new outgoing payment request, by using a sequencer (OUGP).

When calling the API, the **defaultBusinessStatus** key is optional. However, if you fill it in, you must also fill in the **scheduledDate** key. See more details in the [Request Data Parameters](#) section, below.

**NOTE**

All new requests are registered in **Draft** status, when no value is sent for the **defaultBusinessStatus** key.

## Generate an Outgoing Payment Request - Example

A user registers a new outgoing payment request in the system:

```
1  var p = {
2  "paymentRequest":{
3  "paymentNo":"Pay0029.2",
4  "paymentType":"premiumReturn",
5  "paymentAmount":500,
6  "paymentDueDate":"2021-08-20",
7  "creationDate":"2021-08-20",
8  "proposedBy":"Jane.Doe",
9  "currency":"RON",
10 "scheduledDate":"2021-08-30",
11 "defaultBusinessStatus": "Approved", // available
    values: "Proposed", "Approved"
12 "comments":"This is the payment request."
13 },
14 "paymentBeneficiary":{
15 "beneficiaryType":"payer",
16 "beneficiaryCategory":"Individual person",
17 "uniqueID":"2920202020220",
18 "firstName":"Jane",
19 "lastName":"Doe",
20 "name":null,
21 "iban":"R069BNKB0001000000000000"
22 },
23 "payerDetails":{
24 "payerName":"CompanyName",
25 "iban":"R069BNKB0001000000000000"
26 }
27 } ;
```

```
28 | ebs.callActionByNameAsync("FTOS_PYMT_
    | AddOutgoingPaymentRequest", p).then(function(e)
    | {console.log(e)});
```

## Request Data Parameters

**IMPORTANT!** When creating a new outgoing payment record, the following information should be sent in order for the action to be successful. All fields are **mandatory**, except when declared otherwise.

Parameter	Description
paymentRequest	Array with the payment request details.
paymentBeneficiary	Array with details to identify the payment beneficiary.
payerDetails	Array with details to identify the payer.

### paymentRequest Parameter

Field	Observations
Payment No	The unique number of the outgoing payment.
Payment Type	The payment type. The option set can take the following values: <b>paymentInAdvance</b> , <b>paymentOrder</b> , <b>paymentWriteOff</b> , <b>schedule</b> , <b>paymentReturn</b> , <b>paymentExternal</b> , <b>paymentClaim</b> , <b>paymentUnallocated</b> , <b>paymentPartner</b> , <b>brokerPremiumPayment</b> , <b>brokerCommissionPayment</b> , <b>brokerClaimPayment</b> , <b>brokerClientReturnPayment</b> , <b>brokerPaymentReturn</b> , <b>outgoingPayment</b> , and <b>bankCharges</b> .
Payment Amount	The outgoing payment amount.
Payment Due Date	The outgoing payment due date.
Creation Date	The outgoing payment request creation date.
Proposed by	The user who makes the request.
Currency	The currency of the outgoing payment.
Scheduled Date	The date scheduled for the payment. Not mandatory for payments registered in <b>Draft</b> status.



Field	Observations
Default Business Status	The default business status for the payment request. The available values are: <b>Proposed</b> and <b>Approved</b> . This key is optional.
Comments	Not a mandatory field.

## paymentBeneficiary Parameter

Field	Observations
Beneficiary Type	Option set used to choose the beneficiary type. The option set values are as follows: <b>payer</b> , <b>insured</b> , <b>contractor</b> , <b>policyBeneficiary</b> , <b>broker</b> , <b>serviceProvider</b> , and <b>other</b> .
Beneficiary Category	Set to <b>Company</b> or <b>Individual</b> .
Unique ID No	The unique number of the outgoing payment beneficiary.
Name	The name of the outgoing payment beneficiary.
IBAN	The IBAN of the outgoing payment beneficiary.
Bank	Must correspond to IBAN.

## payerDetailsParameter

Field	Observations
Payer Name	The name of the payer.
Payer IBAN	The IBAN of the payer.
Payer Bank	Must correspond to IBAN.

## Response

This is an example of a response:

```

1  {
2      "UIResult": {
3          "NavigateToEntityPage": false,
4          "NavigateToEntityPageOnEdit": false,
5          "NavigateToEntityFormName": null,
6          "NavigateToEntityName": null,
7          "NavigateToEntityId": null,
8          "NavigateToEntityInsertDefaults": null,

```

```

9         "NavigateToUrl": null,
10        "DownloadFile": null,
11        "ReloadPage": false,
12        "Message": null,
13        "IsSuccess": false,
14        "Data": "
    {\"
    isSuccess\
    \":true,\"
    errorMessage\":null,\"errorCode\":null,\"result\":
    {\"outgoingPaymentRequestId\": \"394cc3cf-5059-455f-8b6b-
    93054f201dd7\", \"paymentReference\": \"REF0000090\"}}\",
15        "NavigateToPageNo": null
16    },
17    "Message": null,
18    "IsSuccess": true,
19    "ClientScript": null,
20    "Serialized": null,
21    "ErrorCode": 0
22 }

```

Response description:

Key	Description
errorCode	Error code.
isSuccess	Marks weather the request is successful or not.
errorMessage	Error message.
outgoingPaymentRequestId	The unique id of the outgoing payment request that was added.
paymentReference	The payment reference of the request that was added.

## Error Messages

The following are the error messages that can be encountered during outgoing payments generation process:

Code	Text	Description
ERR01.01	ERR01.01 - Invalid paymentType!	This payment type is not found in the FTOS_PYMT_OutgoingPaymentType option set.

Code	Text	Description	
ERR01.02	ERR01.02 - Invalid currency!	No active currency found	
ERR01.03	ERR01.03. - Invalid beneficiaryCategory!	This error appears when the beneficiary category is not found in the <b>FTOS_CMB_AccountType</b> entity or the account type selected is "Self employed individual".	
ERR01.04	ERR01.04 - First name and last name mandatory for individual person beneficiary category!	When the "Individual person" beneficiary category is selected, the first name and last name are mandatory.	
ERR01.05	ERR01.05 - Name is mandatory for legal person beneficiary category!	When the "Legal person" beneficiary category is selected, the name is mandatory.	
ERR01.06	ERR01.06 - Name should be empty for individual person beneficiary category!	The attribute name should be empty when the "Individual person" beneficiary category is selected.	
ERR01.07	ERR01.07 - First name should be empty for legal person beneficiary category!	The attribute first name should be empty when the "Legal person" beneficiary category is selected.	
ERR01.08	ERR01.08 - Last name should be empty for legal person beneficiary category!	The attribute last name should be empty when the "Legal person" beneficiary category is selected.	
ERR01.09	ERR01.09 - Payment number is mandatory!	The payment number should be completed.	
ERR01.10	ERR01.10 - Payment amount is mandatory and must be a number!	The payment amount should be filled in with a number.	
ERR01.11	ERR01.11 - Payment due date is mandatory and must be a date!	The payment due date should be filled in with a date.	

Code	Text	Description
ERR01.12	ERR01.12 - Payment creation date is mandatory and must be a date!	The payment creation date should be filled in with a date.
ERR01.13	ERR01.13 - Payment scheduled date is mandatory and must be a date!	The payment scheduled date should be filled in with a date.
ERR01.13.1	ERR01.13.1 - Payment scheduled date has to be equal or bigger than the current date!	The payment scheduled date has a validation to be equal or greater than today.
ERR01.13.2	ERR01.13.2 - Payment scheduled date must be a date!	The payment scheduled date should be filled in with a date. Only appears when defaultBusinessStatus is not completed.
ERR01.14	ERR01.14 - Invalid beneficiary Type!	This beneficiary type is not defined in the FTOS_PYMT_OutgoingPaymentBeneficiary option set.
ERR01.15	ERR01.15 - Payment beneficiary iban is mandatory!	The IBAN should be completed.
ERR01.15.1	ERR01.15.1 - Payment beneficiary iban is not valid!	Invalid payment beneficiary IBAN.
ERR01.15.3	ERR01.15.3 - No matching Bank for payment beneficiary iban!	This bank couldn't be identified based on the payment beneficiary IBAN filled in.
ERR01.16	ERR01.16 - Payer name is mandatory!	The payer name should be completed.
ERR01.17	ERR01.17 - Payer iban is mandatory!	The payer IBAN should be completed.
ERR01.17.1	ERR01.17.1 - Payer iban is not valid!	Invalid payer beneficiary IBAN.
ERR01.17.3	ERR01.17.3 - No matching Bank for payer iban!	This bank couldn't be identified based on the payer IBAN filled in.

Code	Text	Description
ERR01.18	ERR01.18 - Payment number must be unique!	There is another payment registered with this number.
ERR01.19	ERR01.19 - Invalid user for ProposedBy.	Invalid user.
ERR01.20	ERR01.20 - Invalid business status!	Invalid business status!

## Endpoint

### **FTOS\_PYMT\_AddOutgoingPaymentRequest**

Endpoint used to run the **FTOS\_PYMT\_AddOutgoingPaymentRequest** server automation script. See below.

## Server Automation Script

**FTOS\_PYMT\_AddOutgoingPaymentRequest** - Script called with a data object. This script first calls the `validateRequest` function from the **BillingCollectionAPIs** server automation script library. If the function returns the `isSuccess` answer, then the script uses the same object to call the `addOutgoingPaymentRequest` function, from the **BillingCollectionAPIs** server automation script library.

## Server Automation Script Library

The **BillingCollectionAPIs** library validates a request for an outgoing payment and creates a new request record based on the object received from the `FTOS_PYMT_AddOutgoingPaymentRequest` script.

The library contains the following functions:

### **isNullOrEmpty**

This function verifies if an input value is **null** or **empty**.

**Input** parameters: `value` - The value that needs validation.

**Output** parameters: `true/ false` - The result of the validation.

## isValidDate

This function validates the date.

**Input** parameters: `dateString` - The date string formatted like `yyyy-MM-dd`.

**Output** parameters: `true/ false`.

## getBankId

This function returns the Id of the bank from **FTOS\_PYMT\_Bank** entity.

**Input** parameters: `bankCode` - string - The code of the bank.

**Output** parameters: `bankId` - The Id of the bank, from **FTOS\_PYMT\_Bank** entity.

## getPaymentNo

This function returns the Id of the outgoing payment from the **FTOS\_PYMT\_OutgoingPayment** entity

**Input** parameters: `paymentNo` - string - The number of the payment.

**Output** parameters: `outgoingPaymentId` - The Id of the outgoing payment, from **FTOS\_PYMT\_OutgoingPayment** entity.

## getUserId

This function returns the `userId` from the **System User** entity.

**Input** parameters: `userName` - string - The name of the user.

**Output** parameters: `userId` - The Id of the user, from the **System User** entity.

## getPaymentDetails

This function returns the reference number from **FTOS\_PYMT\_OutgoingPayment** entity.

**Input** parameters: **outgoingPaymentId** - string - The Id of the outgoing payment request.

**Output** parameters: **referenceNo** - The reference number from **FTOS\_PYMT\_OutgoingPayment** entity.

## getPayerDetailsFromProcessor

This function returns the default payer details (name and IBAN), set in the processor **FTOS\_PYMT\_OutgoingPayments\_PayerDetails**. This allows the user, or a third-party app, to call the API for generating an outgoing payment request without sending the payer details.

**Input** parameters: N/A.

**Output** parameters: **payer details** - A JSON structure containing the values from the processor.

## addOutgoingPaymentRequest

This function inserts a new outgoing payment request, based on the given object from the **FTOS\_PYMT\_AddOutgoingPaymentRequest** server automation script.

**Input** parameters: **input** – object – The object received from the **FTOS\_PYMT\_AddOutgoingPaymentRequest** server automation script. The object contains the details about:

- the **payment request**: **paymentNo**, **paymentType**, **paymentAmount**, **paymentDueDate**, **creationDate**, **proposedBy**, **currency**, **scheduledDate**, **comments**;
- the **payment beneficiary**: **firstName**, **lastName** - for individual persons, **name** - for legal persons and other category, **uniqueID**, **beneficiaryType**,

`beneficiaryCategory, iban;`

- the **payer**: `payerName, iban`.

**Output** parameters:

- `id` - The Id of the newly inserted outgoing payment request.
- `referenceNo` - The reference number of the newly inserted outgoing payment request.

The `addOutgoingPaymentRequest` function also uses some of the functions presented above and the `getIdByAttrib` function, presented below.

## getIdByAttrib

This is a function from the **FTOS\_INS\_Utils** library. This function returns the `Id` of the given attribute.

**Input** parameters:

- `entityName` – string – The entity name.
- `searchAttribute` – string – The name of the attribute.
- `searchValue` – string – The value of the attribute .

**Output** parameters: The `Id` of the searched attribute. If there are no entries with the searched value, the output is `null`.

## validateOSIValues

This function validates an option set value from an option set.

**Input** parameters:



- `optionsetName` - string - The name of the option set.
- `optionsetItemValue` - string - The value of the option set item.

**Output** parameters: `true/ false`.

## validateRequestReturn

This function returns an object result with one of the following responses: `isSuccess`, `errorMessage`, `errorCode`.

**Input** parameters:

- `isSuccess` - The success message.
- `errorMessage` - The error message.
- `errorCode` - The error code.

**Output** parameters: `rez` - An object containing the result.

## validateRequest

This function validates the input object from the **FTOS\_PYMT\_AddOutgoingPaymentRequest** server automation script and uses some of the functions presented above.

**Input** parameters: `input` – object – The object received from the **FTOS\_PYMT\_AddOutgoingPaymentRequest** server automation script.

**Output** parameters: `rez` - An object containing the following details: `{isSuccess:true/false, errorMessage: "message error", errorCode: "error code"}`.

## validateScheduledDate

The function validates if the scheduled date has the correct date and format, and if it was submitted (for the case when it was marked as mandatory).

**Input** parameters:

- `rez` - The result object.
- `scheduledDate` - The scheduled date.
- `mandatory` - boolean - The variable to determine if a scheduled date is mandatory.

**Output** parameters: returns an object - `rez`, with the followings  
 {isSuccess: true/false, errorMessage: "message error", errorCode: "error code"}

## Generate UK Direct Debit Mandate API

This functionality adds new UK direct debit mandates in the system, based on API requests received from other **FintechOS** solutions, from other systems or external applications (like digital journeys). On the **FTOS\_PYMT\_GenerateMandate** entity, for the newly generated UK direct debit mandate, the API response populates the following attributes: `Account Holder First Name`, `Account Holder Last Name`, `Bank Sort Code`, `Account Number`, and `Reference` (the policy number). All mandates are registered in **Draft** status, with `mandateStage = N` (new) and **Begin Date** = `null`. See more below.

## Generate Direct Debit Mandate API Example

The system registers a new direct debit mandate request, containing details from one policyholder.

```
1 | let inputData = {
```

```

2     "accountHolderFirstName": "John",
3     "accountHolderLastName": "Doe",
4     "bankSortCode": "04-00-26",
5     "accountNumber": "55743513",
6     "reference": "80001033"
7   }
8
9   ebs.callActionByNameAsync('FTOS_PYMT_GenerateMandate',
inputData).then(function(e) {
10     console.log(e)
11   });

```

## Request Data Parameters

The following is the list of data parameters included in the request:

Parameter	Description
accountHolderFirstName	The first name of the payer.
accountHolderLastName	The last name of the payer.
bankSortCode	Sort code of the bank - it specifies the bank branch.
accountNumber	The account number used for the direct debit payment.
reference	The policy number.

## Response

The following is an example of a response:

```

1   {
2     "isSuccess": true,
3     "errorMessage": null,
4     "errorCode": null,
5     "result": null
6   }

```

Response description:

Key	Description
errorCode	Error code.
isSuccess	Marks weather the request is successful or not.
errorMessage	Error message.

Key	Description
result	An object containing the following key: <code>mandateId</code> - the Id of the newly generated mandate OR a <b>confirmation message</b> , when a mandate for the input reference already exists. If so, the following message is displayed: "There is already a mandate with the reference number... (x)... for addition!" (The message also includes the specific mandate reference number.)

## Error Messages

The following are the error messages that can be encountered during the API invoice generation process:

Code	Text	Description
ERR.BL.50201	The input parameters keys are wrong or missing!	<code>inputData</code> doesn't have all the properties or some properties are wrong.
ERR.BL.50202	There are no input parameters!	This error appears when provided empty data.
ERR.BL.50103	The input parameters should not contain null or empty values!	<code>inputData</code> has properties with null or empty values.

## Endpoint

The **FTOS\_PYMT\_GenerateMandate** endpoint is used to run the **FTOS\_PYMT\_GenerateMandate** server automation script.

## Server Automation Script

**FTOS\_PYMT\_GenerateMandate** - Script called with a data object.

This script calls the **BillingCollectionAPIs** library and, also, executes the following:

- Validates the request data object.
- If all the necessary conditions are met, it uses the input parameters to generate a new direct debit mandate.

## Server Automation Script Library

From the **BillingCollectionAPIs** library, the **GenerateDIDEMandate** function is used. This function wraps all the functions from bellow:

### validateRequest

This function validates the request fields.

**Input** parameters: **inputData** - The object containing the keys needed to call the endpoint.

**Output** parameters - An **object** containing the keys to describe the result of the validation:

- **isSuccess** - true/ false.
- **errorMessage** - **null** or **error message** as described in the error messages list.
- **errorCode** - **null** or **error code** as described in the error messages list.
- **result** = [] - An array containing the result of the request.

### generateMandate

This function inserts the data from the **inputData** object into **FTOS\_PYMT\_DirectDebitMandateBuffer** entity and calls the function **performActionsOnMandate** from the **PYMT\_Mandate** library which adds a new direct debit mandate into the **FTOS\_PYMT\_DirectDebitMandate** entity.

**Input** parameters: **inputData** - An object containing the keys needed to call the endpoint.

**Output** parameters: **result** - An object containing the following key: **mandateId** - the Id of the newly generated mandate OR a **confirmation message**, when a mandate for the input reference already exists. If so, the following message is displayed: "There is already a mandate with the reference number... (x)... for addition!" (The message also includes the specific mandate reference number.)

# Billing Notifications

This functionality helps insurers to send notifications to the customers, regarding the status of their invoices. The notifications functionality can be toggled on and off by modifying the **BillingNotification** system parameter - where 0 is **Off** and 1 is **On**.

## Server Side Script Libraries

**FTOS\_INSP\_CoreInsuranceNotification** - This library contains methods that help in the process of notifying the customers about their invoices. From this library the following functions are used:

### getCurrencyDetails

This function returns different attributes from the **FTOS\_CMB\_Currency** entity, based on the currency Id.

**Input** parameters: `currencyId` - (string) - The Id of the **FTOS\_CMB\_Currency** entity.

**Output** parameters: `query` - Array that contains an object with the following results: `Code`.

### getAccountDetails

This function returns different attributes from the **Account** entity, based on the account Id.

**Input** parameters: `accountId` - (string) - The Id of the **Account** entity.

**Output** parameters: `query` - Array that contains an object with the following results: `FirstName`, `LastName`, `Email`, `dateOfBirthInv`.

### getPolicyDetails

This function returns different attributes from the **FTOS\_INSPA\_Policy** entity, based on the statement detail Id.

**Input** parameters: **statement** - (string) - The Id of the **FTOS\_PYMT\_StatementDetail** entity.

**Output** parameters: **query** - Array that contains an object with the following results: **PolicyNo**, **policyBeginDate**, **insuranceTypeId**.

## getInsuranceDetails

This function returns different attributes from the **FTOS\_IP\_InsuranceType** entity, based on the insurance type Id.

**Input** parameters: **typeId** - (string) - The Id of the **FTOS\_IP\_InsuranceType** entity.

**Output** parameters: **query** - Array that contains an object with the following results: **Name**.

## getStatementDetails

This function returns different attributes from the **FTOS\_PYMT\_Statement** entity, based on the statement Id.

**Input** parameters: **statementId** - (string) - The Id of the **FTOS\_PYMT\_Statement** entity.

**Output** parameters: **query** - Array that contains an object with the following results: **customerId**, **statementNo**.

## addPassword

This function encrypt a .pdf file, using the date of birth from the **Account** entity. If the account doesn't have a date of birth registered, the file will not be encrypted.

**Input** parameters:

- **dob** - (Invariant Date) - The date of birth from the **Account** entity.
- **fileName** - The Id of the .pdf file.

**Output** parameters: N/A

## getPaymentConfirmationDetails

This function returns the data needed to populate the **InvoicePaymentConfirmation** email **template**, based on the context.

**Input** parameters: **context** - (object) - The context object received.

**Output** parameters: **tokens** - Object that contains the following information: **InvoiceDate**, **FirstName**, **LastName**, **InvoiceNumber**, **PolicyType**, **PolicyNo**, **BeginDate**, **Email**, **Attachments**.

## getInvoiceGeneratedDetails

This function returns the data needed to populate the **InvoiceGeneratedNotification** email **template**, based on the context.

**Input** parameters: **context** - (object) - The context object received.

**Output** parameters: **tokens** - Object that contains the following information: **InvoiceDate**, **FirstName**, **LastName**, **InvoiceNumber**, **PolicyType**, **PolicyNo**, **BeginDate**, **Email**, **Attachments**.

## getFollowUpDetails

This function returns the data needed to populate the **InvoiceFollowUp** email **template**, based on the Id of the invoice.

**Input** parameters: **statementId** - (string) - The Id of the **FTOS\_PYMT\_Statement** entity.

**Output** parameters: **tokens** - Object that contains the following information: **InvoiceDate**, **FirstName**, **LastName**, **InvoiceNumber**, **PolicyType**, **PolicyNo**, **BeginDate**, **Email**, **Attachments**.



## sendClientNotification

This function sends the notifications, based on the type of the email. If the account doesn't have an email registered, the notification will not be sent.

**Input** parameters:

- `emailDetails` - (object) - Object containing the information to populate the email templates.
- `type` - (string) - The type of notification to be sent.

**Output** parameters: N/A.

## Processors

The `FTOS_INSP_CoreInsuranceNotification` processor is used for setting the `NotificationConfiguration` parameter. This parameter is an object containing the following settings:

- `followUp` - **Follow up** notification,
- `invoiceGenerated` - **Invoice Generated** notification,
- `paymentConfirmation` - **Payment Confirmation** notification.

## Business Workflow Configuration Actions

### DRAFT\_Generated

`GenerateInvoiceNotification` Action

It calls `getInvoiceGeneratedDetails(context)` and `sendClientNotification(emailDetails, type)` functions from `FTOS_INSP_CoreInsuranceNotification` server side script library in order to get the data needed to populate the email **template** and send the notification to the customer.

It calls the `getFlowProcessorSettingsByType(flowSettingsName, processorSettingsType, processorSettingsName)` function from the **FTOS\_DFP\_FlowProcessorSettings** server side library, in order to get the processor settings.

The notification is sent if the payment type is different than the **broker collection** payment type and if the **BillingNotification** system parameter is set to **1**.

## CLOSED\_Paid

**PaymentConfirmationNotification** Action

It calls `getPaymentConfirmationDetails(context)` and `sendClientNotification(emailDetails, type)` functions from **FTOS\_INSP\_CoreInsuranceNotification** server side script library in order to get the data needed to populate the email **template** and send the notification to the customer.

It calls the `getFlowProcessorSettingsByType(flowSettingsName, processorSettingsType, processorSettingsName)` function from the **FTOS\_DFP\_FlowProcessorSettings** server side library, in order to get the processor settings.

The notification is sent if the payment type is different than the **broker collection** payment type and if the **BillingNotification** system parameter is set to **1**.

## On Demand Scripts

**JOB FTOS\_PYMT\_FollowUpInvoice** - on demand script which calls the `getFollowUpDetails(statementId)` and `sendClientNotification(emailDetails, type)` functions from the **FTOS\_INSP\_CoreInsuranceNotification** server side script library in order to get the data needed to populate the email **template** and send the notification to the customer.

It calls the `getFlowProcessorSettingsByType(flowSettingsName, processorSettingsType, processorSettingsName)` function from the **FTOS\_DFP\_FlowProcessorSettings** server side library, in order to get the processor settings.

It calls the `getValues(code)` function from the **FTOS\_PA\_FlowParameter** server side library in order to get the insurance parameter.

The notification is sent if the payment type is other than **broker collection** payment and if the **BillingNotification** system parameter is set to **1**.

**Input** parameters: N/A

**Output** parameters: N/A

## Scheduled jobs

### FTOS\_PYMT\_FollowUpInvoice -

This job is scheduled to run every day at 6:00 AM in order to send notifications to invoices that are still in status Generated x days before the invoice's due date. The number of days is determined by the **NoOfDaysFollowUpInvoice** insurance parameter.

Schedule Services - **FTOS\_PYMT\_FollowUpInvoice**, on demand server automation script.

## Security Roles

**FintechOS** security architecture is a unified security design aimed at empowering **FintechOS** clients to address the necessities and potential risks involved in a certain scenario or environment. The **Security Roles** are an inbuilt part of the **Core DPA Platform** security architecture, designed to help you mitigate cybercrime-related risks and keep data secure across all your business flows. Consequently, you use **Security Roles** to protect sensitive data and configure various organization layers to allow for better communication, collaboration, or reporting.

**NOTE**

For more details, see also the [Default Security Roles](#) documentation.

On top of the platform's default security roles, the **Billing and Collection** solution comes with three pre-defined **Security Roles** which allow you to:

- Control the level of user access to various actions, functions or operations.
- Maintain compliance with security standards with regard to processing sensitive information.
- Safeguard end-to-end ownership over your billing and collection operations.

The following are the defined security roles for the **Billing and Collection** solution:

Security role	Description
Operations user	This is the user role for <b>performing imports</b> in the system. For example you use this role for uploading bank statements, notification files from online payment processors and direct debit notification or instruction files. Please see the table below for the available access privileges for this user role.
Operations superUser	This is the user role for <b>operating with payment data</b> . For example you use this role for allocating or deallocating payments, initiating outgoing payment requests, or delete installments. Please see the table below for the available access privileges for this user role.
Operations manager	This is the user role for <b>approving payment requests</b> . Please see the table below for the available access privileges for this user role.

The following are the defined security privileges per every role:

Functionality	Operations User	Operations SuperUser	Operations Manager	Operation
Invoices				
	x	x	x	V View
				I (Insert)
		x		E (Edit)
Installments				
	x	x	x	V
				I
		x		E - can remove installment

BILLING AND COLLECTION USER GUIDE

Functionality	Operations User	Operations SuperUser	Operations Manager	Operation
Premiums collected - Payments				
	x	x	x	V
	x	x		I
		x		E
Bank statements				
	x	x	x	V
	x			I
				E
Payments - Incoming and Outgoing				
	x	x	x	V
	x	x		I
		x		E
Payment returns and Unallocated payments				
		x	x	V
		x		I
		x		E
Outgoing Payment Request Approval				
			x	V
			x	I
			x	E
Payments Allocation				
	x	x	x	V
		x		I
		x		E
Outgoing Payment Requests				
	x	x	x	V
		x		I
		x		E
DIDE Mandates				

BILLING AND COLLECTION USER GUIDE

Functionality	Operations User	Operations SuperUser	Operations Manager	Operation
	x		x	V
	x			I
	x			E
DIDE Notification file				
	x		x	V
	x			I
				E
DIDE Payment Instructions file				
	x		x	V
				I
				E
DIDE Payment Confirmations				
	x		x	V
	x			I
				E
DIDE Payment Denied				
	x		x	V
	x			I
				E

For the UK flow:

Security role	Description
Operations user	<p>This user has the rights to see the DIDE files (External Reports) menu, lists and forms, also having the possibility to import new files: ADDACS and ARUDD files.</p> <p>FTOS_PYMT_ADDACSReasonType - read                      FTOS_PYMT_DIDE_ADDACSDetail - read                      FTOS_PYMT_DIDE_ADDACS - create, read, update                      FTOS_PYMT_ARUDD - create, read, update                      FTOS_PYMT_ARUDDDetail - create, read, update                      FTOS_PYMT_DIDEMandateInstruction - read                      FTOS_PYMT_DIDEMandateInstructionDetail - read                      FTOS_DFP_FlowSettings - read</p>

Security role	Description
Operations manager	View rights for all new DIDE entities: FTOS_PYMT_ADDACSReasonType - read FTOS_PYMT_DIDE_ADDACSDetail - read FTOS_PYMT_DIDE_ADDACS - read FTOS_PYMT_ARUDD - read FTOS_PYMT_ARUDDDetail - read FTOS_PYMT_DIDEMandateInstruction - read FTOS_PYMT_DIDEMandateInstructionDetail - read

**HINT**

Apart from the **Billing and Collection Security Roles**, you can always define new roles to meet your business needs. For more details, consult the [Creating or Editing Security Roles](#) documentation.

## Digital Assets

The following are the digital assets of the **Billing and Collection** solution:

### Billing & Collection

**SDK Code:** FTOS\_INSP\_BillingCollection 2.2.0

**Description:** This digital asset contains all the important customization items defining the solution, combining data model elements with SDK scripts. It includes business entities, attributes, business entity extensions, business workflows, entity forms and entity views, covering the data model of the solution. Client-side and server-side scrips used to automate and enhance the capabilities of the solutions are also part of this asset. It also includes data import templates and **config data definition** used to populate the entities included in the solution.

**Items Contained:** It includes business entities, attributes, business entity extensions, business workflows, entity forms and entity views, covering the data model of the solution.

### Billing and Collection Menu

**SDK Code: FTOS\_INSP\_BillingCollection\_Menu 2.2.0**

**Description:** This digital asset contains a collection of menu items used to create menu entries for the most important flows included in the solution.

**Items Contained:** Menu items.

## Billing and Collection Import

**SDK Code: FTOS\_INSP\_BillingCollection\_Import 2.2.0**

**Description:** This digital asset includes default settings that can be used to customize the behavior of the solution or to automate the processes.

**Items Contained:** It includes scheduled jobs, insurance parameters, flow settings, and versioning settings for the solution entities.



# Dashboards

**Dashboards** bundle together data views that are either needed more often than others or they simply have priority, in your everyday work. **Dashboards** are also a flexible way to work with your data since they can be easily updated when needed - for example when a flow must be given priority over other flows due to an increase in the frequency of its use.

The **Billing and Collection** solution comes with a configured dashboard, displayed in your portal. The dashboard provides links to the most important flows, making it easier for you to access the [Unallocated Payments List](#), the [Direct Debit Mandates List](#) or to insert [Bank Statements](#). The dashboard also displays views of the [Invoices List](#), the [Outgoing Payments Requests List](#) and the [Payments List](#) allowing you to rapidly inspect the latest payment data listed in any of the mentioned sections.

Below, you can see the example of a dashboard:

The dashboard features a navigation bar with the following tabs: APPS, MAIN DASHBOARD, BILLING & COLLECTION (highlighted), POLICY ADMIN, REINSURANCE, CLAIMS, and LIFE AND HEALTH. Below the navigation bar are three main action cards:

- Unallocated Payments List**: Unallocated Payments
- Insert Bank Statements**: Bank Statements
- Direct Debit Mandates List**: Direct Debit Mandates Activated

Below these cards is an **Invoices List** table with the following data:

Invoice No.	Invoice Date	Contractor	Invoice Amount	Currency	Due Date	Business Status
REF0000703	03/11/2021		75.00	EUR	04/11/2021	Generated
REF0000702	03/11/2021		33.00	EUR	08/11/2021	Generated

Depending on your level of authority (security role), you might have access to different **FintechOS** insurance solutions, functionalities or processes, **Billing and**

**Collection** being just one of them. As you can see in the image above, the portal dashboard is yet another way to help you smoothly navigate between the insurance applications that you use.

Finally, since **Dashboards** is a feature that can be easily adjusted in **Innovation Studio**, you also have the means to configure the **Billing and Collection** dashboard to respond to your particular needs. So, if more widgets or fewer views are needed to help you keep your financial data organized, accessible, and comprehensible this is easily done with **Innovation Studio**.

For more details about adjusting your dashboards, consult the [Dashboards](#) documentation.

**HINT**

Instead of starting from scratch, you can start by adjusting the dashboard offered by the **Billing and Collection** solution.

# Glossary

For more terms, check also the [FintechOS Getting Started Glossary!](#)

---

## B

---

### **Bank Statement**

A list of financial transactions, issued by a bank.

### **Business Formulas**

FintechOS solution handling advanced computations mapped to different business needs - such as scoring, pricing, and more.

### **Business Workflows Processor**

FintechOS solution dedicated to processing business workflows automatically.

## D

---

### **Direct Debit Payment**

By way of a Direct Debit Instruction (Mandate), the bank transfers the premium amounts into the insurer's account, on behalf of the insured.

## E

---

### **Excess Amount**

The excess amount represents the policyholder's financial contribution to the claims made on the policy. For a policyholder, agreeing to pay an excess amount leads to a discounted total premium per policy.

**N**

---

**Northstar Insurance Suite**

This suite contains all FintechOS solutions for insurance.

**P**

---

**Payment Order**

Bank payment solution for transferring funds from an account to another.

**Product Factory**

FintechOS solution dedicated to building and managing products.

**R**

---

**Revenue**

The earning in the particular depth of insurance after paying the claims and similar other expenses.