

A woman with dark hair is shown in profile, looking down at a tablet she is holding. The background is a blurred city skyline. Overlaid on the lower left is a semi-transparent financial chart with orange bars and a line graph. The text 'Core Policy Admin 3.2.0' is centered in the upper right.

Core Policy Admin 3.2.0

User Guide

TOC

Overview	4
Security Roles for Core Policy Admin	8
Creating a Policy	11
Core Policy Admin	12
Policies	13
Policy Cancellation	14
Mid Term Adjustments	30
Types of Policy Updates	31
Master Policy Contract	35
Master Policy Generation	35
Automatic Renewal	45
Configuring Policy Automatic Renewal	45
Renewal Offers	48
Automatic Renewal Process	50
Policy Configurations	55
Business Functionalities	55
InForce Flow	57
Flow Parameters and Scheduled Jobs	60
1 Flow Parameters	60
2 Scheduled Jobs	61
Business Workflows	64
Mid Term Adjustments Journeys	74
Policy Cancellation Journeys	89
1 FTOS_INSP_PolicySurrender	89

2 FTOS_INSP_Payment	104
Other Automated Journeys	107
Policy Generation API	110
Endpoints	118
Get Policy Data API	133
PolicyStatusChange API	146
Policy Versioning	151
Business workflow configuration actions	154
Core Policy Admin Formulas	156
Installation	159
1 Prerequisites	159
2 Import Packages	160
3 Installation	160
4 After installment configurations	163
1 Prerequisites	166
2 After installment configurations	167
1 Prerequisites	167
2 After installment configurations	168
Glossary	169

Overview

The **Core Policy Admin** module provides administration services to an insurer's business operation. This is the business area that is responsible for the heavy lifting aspect of the insurance business. Once new business processing is complete, there is huge scope for the alteration of policies in issue to fulfill the obligations of flexibility, and morphing of a policy and its benefits so that it stays relevant to the policyholder's circumstances.

The module encompasses the registration of permitted policy alterations, such as an upgrade to core healthcare benefits, whilst automatically identifying and communicating the requirements and evidence necessary to recalculate and complete the alteration. Evidence and document requirements gathering is orchestrated by FintechOS and assisted by [Automation Processors](#).

Business experts directly involved in the process can use **Core Policy Admin** module to refine and deploy digital journeys so that they cover the requirements of legacy, current and planned products in an order that meets the identified priorities of their business.

The **Core Policy Admin** module covers the following typical events:

- Mid term adjustments;
- Benefit additions or increases;
- Removal or reduction of benefits;
- Addition/removal of insured persons or objects;
- Automatic premium/benefit escalations;
- Premium collection alterations leveraging FintechOS Premium Billing;
- Notifications to agents, brokers or other professional intermediaries responsible for servicing the policyholders portfolio if applicable;
- Collation and calculation of all policy benefits and reserves relevant to the alteration event

- Initial automatic definition and request for Alteration requirements and evidence specific to the Case
- Ongoing management of the alteration, enabling the definition and request of further requirements and evidence as it evolves and progresses;
- Orchestrating the involvement of:
 - Third parties – Re-insurers, service providers, medical agencies;
 - Other systems of the insurer within the business architecture, such as calculation engines, expert underwriting, re-insurance solutions, fraud and finance;
 - Users, senior underwriters, senior managers and others as required by the specific case;
- Recording and storage of the evidence received and of the decisions made during the process;
- All communications and status changes during the process;
- Generation of downstream activities such as ongoing premium payment schedules, generation of new commissions and fees.

This guide concentrates on use of Innovation Studio in the configuration of a digital journey, that will incorporate FintechOS Automation Processors and highlight the opportunities which exist to greatly improve efficiency, Policy Administration automation, audit and management capabilities and speed time to completion of administrative tasks.

Business Pain Points

The **Core Policy Admin** module helps insurers address many pain points associated with claims management, including:

- Automation of repetitive and otherwise time-consuming process decisions;
- Introduction of Straight Through Processing (STP) to address the bulk of Policy Administration tasks;

- Elimination of previously manual processing steps, reducing human error but also freeing up highly paid specialist staff to concentrate more complex cases;
- Automatic identification and elimination of requests that do not fall within a product versions terms and conditions;
- Leverages FintechOS Automation Processors to greatly improve requirements and evidence gathering and communications.

Core Policy Admin Key Features

The key features of the **Core Policy Admin** module are listed below:

- It enables a central point from which to initiate policy administration tasks that impact on multiple policies;
- It allows the automatic identification of business requirements pertaining to the changes requested.
- It allows the automatic initiation and completion of regular scheduled policy alterations, e.g. annual premium and benefit escalations;
- It offers high levels of automation but with the capability to determine where intervention of experts is required;
- Reverts to manual processing only where necessary for more complex or specialist cases.

Core Policy Admin Key Benefits

The key benefits of the **Core Policy Admin** module are presented below:

- Enables the orchestration and timely involvement of multiple parties, processes and systems:
 - Underwriters, service providers, medical agencies;
 - Re-insurers;
 - External systems, such as Insurance Blacklists, MIB;

- Other systems of the insurer within the business architecture, such as calculation engines, re-insurance solutions, fraud identification and finance;
- Users, expert administration teams, underwriters as required by the specific case.
- Provides a fully auditable end to end administrative process view with no blind spots covering:
 - All participants in the process;
 - Performance of third parties and service;
 - SLA's;
 - Response times for communications to claimants for regulatory reporting.
- Automation of high volume, scheduled or simpler alteration types;
- Introduction of a robust policy administration management process, ongoing improvement of performance, and reduction of costs;
- Regulatory compliance.

HINT Integrate **Core Policy Admin** with more **Lighthouse solutions for insurance** in order to make the best of process automation for your company, portfolios, products and clients!

Security Roles for Core Policy Admin

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.

The following security roles are available for **Core Policy Admin** allowing the users to only perform the actions attributed to them:

Security Role	Description
Policy info	<p>This user only has the right to see the Policies and Masterpolicies list and form (and the Policies + Masterpolicies menu entry) without having the possibility to edit.</p> <div>NOTE Useful for other module's users who need to have policy view rights.</div>
Policy user	<p>Read-only rights for Policies and Masterpolicies.</p> <p>Rights for policy versioning process.</p> <p>Rights to insert a Cancellation request without the possibility to approve it.</p>
Policy superUser	<p>Rights to insert Policy alterations and work on them. Reinitiate payment returns and make the payment returns (last buttons on the Cancellation form).</p>

Security Role	Description
Policy manager	Cancellation approvals and returns approvals.

The following are the defined security privileges per every role (view, insert and edit):

Functionality	Policy info	Policy user	Policy superUser	Policy manager
Masterpolicies	View	View	View	View
Policies	View	View Edit	View	View
Installments	View	View	View Edit	View
Installment payment allocation	View	View	View	View
Invoices	View	View	View	View
Policy coverages	View	View	View	View
Alterations			View Insert Edit	View
Cancellations		View Insert Edit	View Edit	View Edit
Request approval cancellation		View	View	View Edit (approval)
Premium Reimbursements		View	View Edit	View Insert
Return approvals		View	View	View Edit

The table below presents which menu items are accessible for every security role:

Menu item	Policy info	Policy user	Policy superUser	Policy manager
Masterpolicies	x	x	x	x
Policies	x	x	x	x
Alterations			x	x
Cancellations		x	x	x
Premium Reimbursements			x	x
Policy Versionning		x		

HINT

Apart from the **Core Policy AdminSecurity Roles**, you can always define new roles to meet your business needs. For more details, consult the [Creating or Editing Security Roles](#) documentation.

Creating a Policy

The policy issuance process represents the main **Core Policy Admin** functionality through which the insurance contracts are generated within the core system, laying the basis for other **Core Policy Admin** flows. From a business perspective, this functionality involves an end-to-end process that starts from the generation of an initial Policy through a specific endpoint, is updated through an update endpoint and then reaches the **InForce** status, according to the period of validity.

NOTE A **Policy** can be introduced into the system only automatically, by the policy issuance endpoint.

The policy issuance endpoint brings business value to the **Core Policy Admin** component by creating a connection between the core system and various external systems - such as websites, apps and other digital channels that you use in order to reach out to customers. Thus, once the integration with another system that sends in customer data is obtained, the policy is generated in the system according to the received information.

Within this endpoint, a policy generation object is structured to collect the necessary policy related information through the integration with an external system. Following the request made, the response body issues into the system the new data specific to the policy, in addition to those obtained during the data collection from a given external system.

For more details go to the [Policy Generation API](#) page.

Core Policy Admin

With increasing digitization, the amount of data and the number of data points is growing faster and faster – and their intelligent and fast use can be a pressure for insurers. When you use **Core Policy Admin** for collecting, storing and processing policy data, you also save time for: acquiring new customer segments, tapping into other customer needs or creating new products and services.

The **Core Policy Admin** module keeps a traceability during the life period of an insurance contract and its adjustments through time. The solution is comprised of the following key functionalities:

The **Policies** repository - for storing and managing your digital insurance policies.

The **Policy Cancellation** flow - for managing cancellation processes; from registering and validating policy cancellation requests, to calculating the amount to be returned and approving payments.

The **Mid Term Adjustments** flow - for making various changes on the policy with different impacts such as changes in the existing coverage on the policy, changes in the type of payment, changes in the frequency of payment and more.

The **Multipolicies Contract** flow - for creating bundled policy offerings, covering general insurance, healthcare and life protection.

The **Automatic Renewal** flow - for generating renewal offer policies for those which are due to expire.

HINT If needed, you can further customize your **Core Policy Admin** flows by using the **FintechOS Business Workflow Designer**.

Core Policy Admin Key Steps

Core Policy Admin is designed to offer you a streamlined route for managing different types of changes affecting an insurance policy during its lifetime. To access the **Core Policy Admin** module take the following steps:

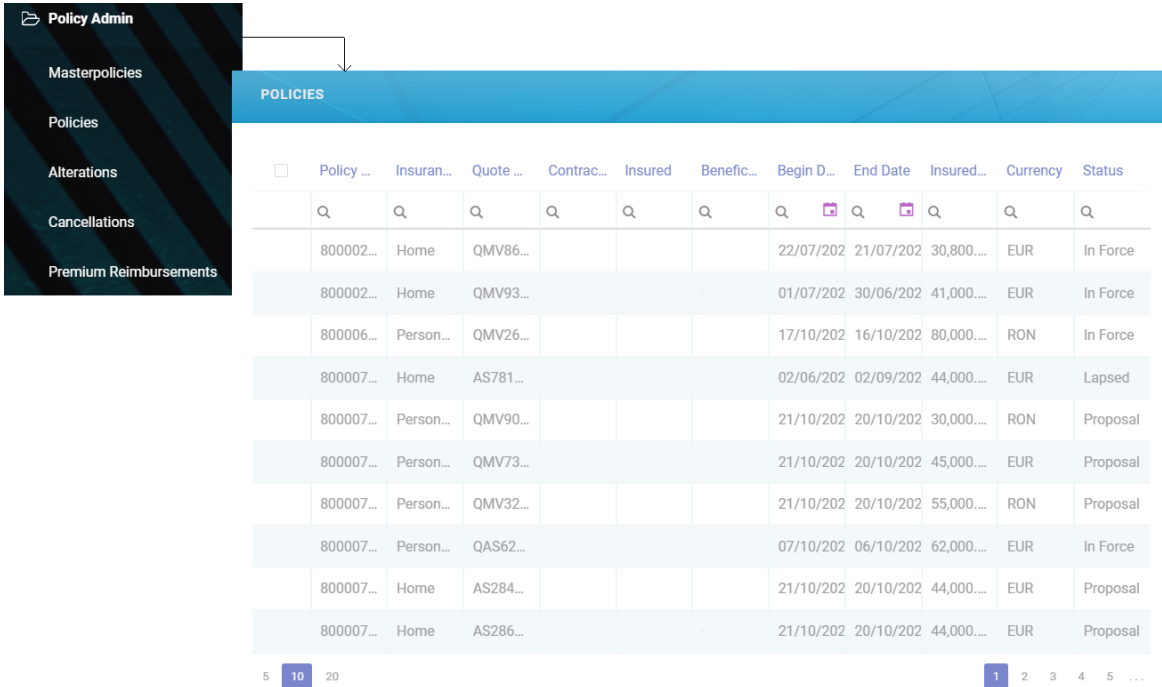
1. Go to your **FintechOS Portal** and click on the main left menu.
2. From the drop-down, click to open **Core Policy Admin**. Inside the **Core Policy Admin** drop-down:
 - If you need to probe inside your policy repository, click [Policies](#).
 - If you need to need to get a report, click [Policies Report](#).
 - If you need to need to register a new type of payment made on a policy, click [Change Payment Type](#).
 - Click [Policy Cancellation](#) to trigger the digital journey for the cancellation of a policy.

Policies

The **Core Policy Admin** solution enables you to store and manage a virtually unlimited number of digital policies. Use **Policies** when you need to have an overview of the repository of policies from your environment. It is also here where you see the latest **Policies** generated into your system. However, for an in depth analysis of your portfolio, use [Policies Report](#) to extract the needed data.

For accessing the repository, follow the next steps:

1. In your **Fintech OS Portal**, at the top left corner, click the main menu icon to open the drop-down.
2. Click **Policy Admin** and then select **Policies** from the drop-down.
3. When the **Policies** window opens, you can access the repository.



	Policy ...	Insuran...	Quote ...	Contrac...	Insured	Benefic...	Begin D...	End Date	Insured...	Currency	Status
	800002...	Home	QMV86...				22/07/202	21/07/202	30,800....	EUR	In Force
	800002...	Home	QMV93...				01/07/202	30/06/202	41,000....	EUR	In Force
	800006...	Person...	QMV26...				17/10/202	16/10/202	80,000....	RON	In Force
	800007...	Home	AS781...				02/06/202	02/09/202	44,000....	EUR	Lapsed
	800007...	Person...	QMV90...				21/10/202	20/10/202	30,000....	RON	Proposal
	800007...	Person...	QMV73...				21/10/202	20/10/202	45,000....	EUR	Proposal
	800007...	Person...	QMV32...				21/10/202	20/10/202	55,000....	RON	Proposal
	800007...	Person...	QAS62...				07/10/202	06/10/202	62,000....	EUR	In Force
	800007...	Home	AS284...				21/10/202	20/10/202	44,000....	EUR	Proposal
	800007...	Home	AS286...				21/10/202	20/10/202	44,000....	EUR	Proposal

Term Life policies are also displayed in the list, once these are generated through the Generate Policy API.

Policy Cancellation

When a termination of an insurance policy is initiated either by insured or by the insurer, the latter checks in what way the cancellation request falls under the policy scope, and fulfills the request under the agreed terms of the policy. Once approved, the insurer issues some payment, if the case, to the insured, or to an approved third party on behalf of the insured. **FintechOS** clients use the **Core Policy Admin** module to organize and automate routines of this typical scenario, in order to increase the efficiency and accuracy of their operations.

NOTE **Core Policy Admin** makes sure that the same policy is not available inside two cancellation flows, simultaneously. In turn, a policy can be accessed sequentially, by different operators, inside the same cancellation flow - for example,

when a user manages the cancellation processing but a super user must approve the **Returned Premium** payment.

Manage Policy Cancellation Requests

In order to process a policy cancellation request, you must access the **Policy Cancellation List** that displays all the available **Policy Cancellation** records from the database.

In order to do so, follow the instructions from below:

1. In the **Fintech OS Portal**, at the top left corner, click the main menu icon to open the dropdown. From the dropdown, choose **Policy Admin** and click it. The **Core Policy Admin** sub-menu opens.
2. From the sub-menu, choose **Policy Cancellation** and click it. The **Policy Cancellation List** is displayed.
3. In the **Policy Cancellation List** page:
 - To **Add** a new **Policy Cancellation** record, click **Insert**, at the top right corner of the page.
 - To **Edit** a **Policy Cancellation** record from the grid, double click it. When opening an existing **Cancellation request**, the **Edit form** becomes available, with the information previously introduced. Use the form to update the desired fields. Click **Save and Close**, at the top right corner of the page.
 - To **Delete** a **Policy Cancellation** record from the grid, select it and click

Delete, at the top right corner of the page.

The screenshot shows the 'Policy Admin' sidebar on the left with options: Policies, Policy Cancellation, Policies Report, and Change Payment Type. The main area is titled 'POLICY CANCELLATION LIST'. It contains a table with columns: Insured, Final End ..., Policy No., Insurance ..., Cancellati..., Returned ..., Policy Beg..., Policy End..., and Business The table lists several policies with their respective dates, numbers, and statuses. A search form is overlaid on the bottom right, featuring fields for Policy No., Insurance Type, First Name, Last Name, Phone No., Email, PIN / UTR, and Client No., along with Search and Reset buttons.

Insured	Final End ...	Policy No.	Insurance ...	Cancellati...	Returned ...	Policy Beg...	Policy End...	Business ...
	01/07/2021	80000247	Home	23/07/2021	33.00	01/07/2021	30/06/2022	Cancelled
	25/07/2021	80000249	Home	23/07/2021	21.25	25/07/2021	24/07/2022	Approved
	25/07/2021	80000250	Home	23/07/2021	21.25	25/07/2021	24/07/2022	Approved
	22/07/2021	80000216	Home	21/07/2021	0.00	22/07/2021	21/07/2022	Draft
		BRDAG E...	Home			02/06/2021	01/06/2022	Draft
	12/07/2021	BRDAG E...	Home	06/07/2021	0.00	12/07/2021	11/07/2022	Cancelled
	12/07/2021	BRDAG E...	Home	07/07/2021	0.00	12/07/2021	11/07/2022	Approved
		BRDAG E...	Home					
	30/06/2021	BRDAG E...	Home					
	30/06/2021	BRDAG E...	Home					

Policy Search

Sometimes, the **Policy Cancellation** record you need to access is not listed on the first page view inside the **Policy Cancellation List** page. Consequently, to find a **Policy Cancellation** record, you must search for it. In order to do so, click **Insert** at the top right corner of the page. The **Policy Search** form is displayed. Use it to input data about the policy you need to cancel and hit the **Search** button to find it.

The same **Search** form is used for both type of customers: individuals and companies. Only the policies that are in the status of **InForce** are displayed in the list of policies.

The screenshot shows the 'Search' form with the following fields: Policy No., Insurance Type (dropdown), First Name, Last Name, Phone No., Email, PIN / UTR, and Client No. There are Search and Reset buttons at the bottom.

Here are the search keys:

Field	Description
PIN/ UTR	Brings the PIN/ UTR of the Contractor.
Name	Either the first name or the last name of the client. You can use any of these keywords. The search looks into the Account entity and brings the following: Insured, Contractor or the Beneficiary.
Phone No.	Brings the phone number of the Contractor.
Email	Brings the email of the Contractor.
Policy No.	Number of the policy.
Client No.	Number of the client.

After filling in the search parameters, click **Search**. A new list of policies is displayed. Choose the policy to be closed. The **Refresh** button reloads the existing search results. If needed, reset the search process by clicking **Reset**. All the information displayed in the previous search is erased.

IMPORTANT! From the search results, only one policy can be selected for **Policy Cancellation**.

To do so, click **Choose option** near the policy record. A new form related to the cancellation process is displayed. At this stage, a new cancellation record is registered by default with Draft status. This record is later transitioned to other statuses according to the Policy Cancellation process steps. A request can't be saved if there is no owner or no policy. Proceed forward with the change request form.

Change Request and Policy Details

This is where the registration of the request for the policy's cancellation begins. The Edit form of a new Policy Cancellation record initially opens with only 2 tabs: **Change Request** and **Policy Details**. The other tabs are displayed after completing the minimum information required. Consequently, completing the **Change Request Summary** fields subsequently triggers the actual calculations for the payment amounts to be returned to the client - if the case and the calculations are displayed inside the Policy section specific fields.

1

Change Request

2

Policy Details

CHANGE REQUEST SUMMARY

Cancellation

Notification Date

Reason Type

Requested End Date

Final End Date

The following fields must be completed first:

Field	Description
Cancellation Notification Date	The date when the notification for the policy cancellation is made. This is a mandatory field for the process.
Reason Type	From the option set, choose a reason regarding the decision for policy termination. This is a mandatory field for the process. The values of the Reason Type option set are: Withdrawal, Property sold, Cancelled by Client, Decline by screening, Cancelled, Closed by Claim.
Requested End Date	The date from which the closing is expected to take effect. This is a mandatory field for the process.
Final End Date	The date when the change takes effect - when policy is closed. This is a mandatory field for the process.

Choosing a Reason Type triggers the calculation for **Requested End Date** and **Final End Date** fields as in the table below:

The screenshot displays the 'Change Request' form. The 'Reason Type' dropdown menu is open, showing the following options: Property sold, Withdrawal, Cancelled by Client, Decline by screening, Cancelled, and Closed by Claim. The 'Requested End Date' field is currently empty.

Reason Type	Requested End Date	Final End Date
Withdrawal	In this case, it is automatically completed with the Policy Begin Date .	In this case, it is automatically completed with the Policy Begin Date .
Property sold	In this case, the Requested End Date is set to any day before or including the day of the request for cancellation.	Final End Date is filled with the value from Requested End Date but with the possibility to edit.
Cancelled by Client	Here it can be manually completed with any date.	If Notification Date -Policy (Issued) Date <= 15 then this field will be automatically completed with the Policy Begin Date with the possibility to be adjusted. Otherwise this will be automatically completed with the Notification Date + 21days to go
Decline by screening	In this case, it can be manually completed with any date.	It can be filled with the value from Requested End Date but with the possibility to edit.

Reason Type	Requested End Date	Final End Date
Cancelled	Here it can be manually completed with any date, by the following rule: If the Notification Date - Policy (Issued) Date <= 15 days, then the Requested End Date is filled with the Policy Begin Date .	Here it is automatically completed with the Notification Date + 21 days to go. If the Requested End Date meets the rule explained above, the Final End Date is filled with the Policy Begin Date .
Closed by Claim	It is pre-filled with Notification Date + 21 days from the present on.	It is filled with the value from the Requested End Date but with the possibility to edit.

Following the input of the above data, on the next section with details regarding the chosen policy, the fields are automatically filled. The pre-filled details are extracted from Policy basic information section and from the results of the calculation made by selecting the information in the **Change Request Summary** section.

The screenshot displays the 'Policy Details' section of the 'Change Request' interface. The interface is divided into two main parts: a sidebar on the left titled 'CHANGE REQUEST SUMMARY' and a main content area on the right titled 'POLICY'.

The 'CHANGE REQUEST SUMMARY' sidebar contains the following fields:

- Cancellation
- Notification Date
- Reason Type
- Requested End Date

The 'POLICY' main content area contains the following fields:

- Policy No.: BRDAG EMB80000132
- Policy Date: 07/07/2021
- Policy Begin Date: 12/07/2021
- Policy End Date: 11/07/2022
- Insured: Gigi Test
- Contractor: Gigi Test
- Premium Amount: (empty field)
- Premium Currency: EUR
- Paid Amount: 0
- Paid Currency: EUR
- Earned Premium: (empty field)
- Earned Premium Currency: (empty field)
- Canceled Premium: (empty field)
- Canceled Premium Currency: (empty field)
- No of Uninsured Months: (empty field)
- Claims: (checkbox)

Field	Description
Policy No.	The number of the policy when issued.

Field	Description
Policy Date	The date when the policy was issued.
Policy Begin Date	The day when the policy becomes enforced.
Policy End Date	The day when the policy is no longer be available, according to the contract.
Insured	First and last name of the insured person on the policy.
Contractor	First and last name of the contractor on the policy.
Premium Amount	The premium amount of the policy.
Premium Currency	The currency of the policy.
Paid Amount	The total amount of the payments made by the customer on a contract.
Paid Currency	The currency in which the amount has been paid.
Earned Premium	<p>The earned premium on the Cancellation process: The amount is calculated after the following formula:</p> <ul style="list-style-type: none"> If there are no paid or opened claims and <ol style="list-style-type: none"> If $[\text{Cancellation Date} - (\text{Policy Begin Date} + 1)] \leq 15$, then the Earned Premium is equal to 0; If $[\text{Cancellation Date} - (\text{Policy Begin Date} + 1)] > 15$, then the amount is filled in with the following: $\text{Premium Amount} / 12 * (12 - \text{No. of uninsured months})$; If there are no paid or opened claims the Earned Premium is equal with the Premium Amount. <p>The amount calculated should be returned to the client.</p>
Earned Premium Currency	The specific currency of the earned premium is, by default, the premium currency.
Canceled Premium	The cancelled premium.
Canceled Premium Currency	The currency of the cancelled premium.
Uninsured Period Type	The values on a monthly or daily basis according to the Prorata type configured in the system.

Field	Description
Uninsured Period	Calculated as the number of months or number of days from the Policy Begin Date to the Policy End Date according to the uninsured period type set.
Claims	Check the box for existing claims - it should be checked if there are any claims on the policy.

The process continues with filling in the information regarding the customer to whom the payment must be made. In order for the payment to take effect, information must be provided for all the fields in this section. However, this section is left blank when there is no amount to be returned to the customer.

Field	Description
Payment Beneficiary	<p>The Policy Admin user can choose from the list who is the beneficiary of the payment: the Policy Beneficiary, the Insured person or someone else.</p> <p>For the options Policy Beneficiary and Insured, the First, Last Name and PIN are automatically pre-filled.</p>

Field	Description
Payment Beneficiary First Name	The first name of the payment beneficiary.
Payment Beneficiary Last Name	The last name of the beneficiary.
Payment Beneficiary PIN	The PIN of the payment beneficiary.
Bank	The name of the bank towards which the payment for the beneficiary is made.
IBAN Account	The IBAN Account of the payment beneficiary .

After the payment information requested above is introduced, the application is registered in the system in order to go for approval. Canceling the request is also possible, provided you offer a cancellation reason. If you continue with the current cancellation request, clicking the **Register Request** button automatically opens a tab with a new **In Progress** status, while the system calculates the amount to be returned to the beneficiary. Also, the **Comments** field becomes available if there is any observation regarding the request.

Field	Description
Comments	Complete this field with relevant information regarding the cancellation process. This field is optional.
Resolution Reason	When choosing to cancel the registration of the policy termination request, provide relevant information about doing so. This field is mandatory. The values of the option set are: Product reasons, Other reasons, Company reason, Withdraw.

1 Change Request 2 Policy Details

CHANGE REQUEST SUMMARY

Cancellation

Not

Policy

Rec

Rec

Dat

Policy No.

Policy Begin Date

Insured

Premium Amount

Paid Amount

Earned Premium

Canceled Premium

No of Uninsured Months

Beneficiary Last Name

Beneficiary PIN

Bank

IBAN Account

Select a value...

[none]

Product reasons

Other reasons

Company Reason

Withdraw

Comments

Resolution reason

[none]

Register request

Cancel

When clicking **Register request**, you are automatically switched to the next tab, which represents the third step of the process.

This is where you find two predefined lists that contain the valid installments and the claims that have been opened on the current policy. On this tab there is no possibility to insert or delete records. You can only edit a specific record by double-clicking the desired row, which redirects you to the specific **Edit** form of the installment or claim.

1 Change Request
2 Policy Details

INSTALLMENTS

<input type="checkbox"/>	Installment Number	Due Date	Amount	Payment Schedule	Status	Allocation Amount
	1	30/06/2021	120.94	Policy Schedule	Paid	120.9400
	2	30/12/2021	120.95	Policy Schedule	OnTime	

CLAIMS

<input type="checkbox"/>	Policy	Claim No	Open Date	Close Date	Business Status
No data					

Premium Returned

This is a step where all details are automatically filled in as follows:

Field	Description
Policy Start Date	The day when the policy becomes enforced.
Policy End Date	The day when the policy is no longer be available according to the contract.
Last Payment Date	The date when the last installment payment was made.
Interval Until Due Date	The remaining days until the next installment payment.
Interval Type	The type of the interval between the installment payments.
Premium Amount	The premium amount of the insurance policy.
Premium Currency	The currency of the insurance policy.
Paid Amount	The total amount of the payments made by the client on a contract.
Paid Currency	The currency of the paid amount.

Field	Description
Returned Premium Amount	The amount to be returned to the customer. Is equal to the Unearned Premium Amount result, but with the possibility for the user to edit the amount with a desired value if necessary.
Returned Premium Currency	The currency of the returned premium amount.

1 Change Request
2 Policy Details
3 Premium Returned

RETURNED PREMIUM AMOUNT

Policy Start Date
29/06/2021

Policy End Date
28/06/2022

Interval Until Due Date

Interval Type
Month

Premium Amount

Premium Currency
EUR

Paid Amount
0

Paid Currency
EUR

Returned Premium Amount

Returned Premium Currency

Comments

Resolution reason
[none]

Propose change request
Cancel

When the information regarding the value to be returned to the customer is displayed, decide whether the process continues towards the approval phase or towards closing the request. Click **Propose change request** to continue with the cancellation process and send the request to approval or click **Cancel** and complete the **Resolution Reason** field. This field is mandatory.

Requesting approval opens a new tab, while automatically triggering the transitioning from **In Progress** status to **In Approval** status. Also, you can use the **Comments** field to transmit any observations regarding the current request.

Request Approval

Approving the request for policy cancellation requires the intervention of a **Policy Admin Super User** who holds the necessary security rights to approve or reject such cases.

Field	Description
Proposal Date	The date when the cancellation request was scheduled for approval. It is completed with the current date.
Approval Date	The date when the request was approved. It is completed with the date the cancellation application is approved.
User	It is automatically completed with the name of the Policy Admin Super User who is logged in at the approval moment.
Comments	It can be filled with relevant information regarding the approval process or the cancellation flow.
Resolution Reason	In case of declining, the Policy Admin Super User must offer information on why the cancellation application was not approved.

The screenshot displays the 'Request Approval' step of the policy cancellation process. The interface includes a navigation bar with four steps: 1 Change Request, 2 Policy Details, 3 Premium Returned, and 4 Request Approval. The 'Request Approval' step is currently active. Below the navigation bar, there are two panels. The top panel, titled 'RETURNED PREMIUM AMOUNT', contains fields for Policy Start Date (12/07/2021), Policy End Date (11/07/2022), Interval Until Due Date, Interval Type (Month), Premium Amount (435), and Premium Currency (EUR). The bottom panel, titled 'Request Approval', contains fields for Proposal Date (07/07/2021), Approval Date (07/07/2021), User, Comments, and Resolution reason ([none]).

When the case is **Approved**, the flow continues on the approved branch as follows:

1. If the Returned Premium Amount is equal to 0 then the process ends and the status changes from **In Approval** to **Approved**.
2. If the Returned Premium Amount is not equal to 0 then the process continues with the **Payment Return** flow, explained in the [Billing and Collection](#) user guide. In addition, the status changes from **In Approval** to **Approved**. The system redirects the user to the tab where the date of the scheduled payment is displayed.

When the case is **Declined**, the status changes from **In Approval** to **Declined**. The process reaches the end without the possibility of starting again.

Policy Cancellation Status Transitions

The **Policy Cancellation** statuses are as follows:

Status name	Description
Draft	When you open the policy, the Cancellation record is by default in the draft status.
In Progress	When you input the notification date. You can move forward by pressing Register request .
In Approval	Following the registration of the request, for approval use Propose change request button.
Approved	The cancellation request is approved by the Policy Admin super user.
Declined	The cancellation request is declined by the Policy Admin super user.
Cancelled	The cancellation request is cancelled.

Conditions for Policy Cancellation

A policy is terminated and is not considered active if:

- The first installment wasn't paid. If so, after the prescribing period, the policy becomes **Withdraw**.
- The customer wants to terminate the policy before 14 days from issuance. If so, the policy becomes **Withdraw on Client Request**.
- The policy reached **Maturity**.

- The customer didn't pay the installment. If so, after the grace period, the policy becomes **Lapsed**.
- The customer wants to terminate the policy even the payments are up to date. If so, the policy becomes **Cancelled**.
- The customer's risk class was changed into unacceptable risk. If so, all the customer's active policies are terminated with **Decline by Screening** status.
- The insurer reserves the freedom to decline the policy for any **Other** reason.

The above conditions currently shape the policy termination flows but **Core Policy Admin** is a highly customizable solution and you are able to change it according to your needs. For doing so read the [Policy Configurations](#) page.

Policy Cancellation Notification

The system automatically informs the client of the Policy Cancellation process for their insurance contract through a specific notification sent to them.

The policies which are included in the notification are those which have the **Policy Status** = Decline by screening/ Closed by Claim/ Withdraw on client's request/ Cancelled, transitioned due to a Cancellation process.

The notification is sent when the policy status is changed to a specific cancellation status, on the effective date.

The following are the tokens used in the notification:

- Contractor name
- Policy number
- Insurance type
- Policy End Date (new End Date) - the Effective date from the Cancellation process which will be updated on the last active version of the policy

- Email template name : HomeInsurance_ClientPolicyCancelledNotification.

Lapsed Policy Notification

Similarly to the cancellation process, the system informs the client of the Policy Lapsing process for their insurance contract through a specific notification sent to them.

The Policies which included in the notification are those which have the **Policy Status** = Lapsed.

The following are the tokens used in the notification:

- Contractor name
- Policy number
- Insurance type
- Email template: homeInsurance_ClientPolicyStatusLapsed

Mid Term Adjustments

The Policy Alteration functionality allows you to make changes to the active policies, according to the client's request. Thus, through this functionality, you can make various changes on the policy with different impacts such as changes in the existing coverage on the policy, changes in the type of payment, changes in the frequency of payment and more. Each change in the policy also generates a new version of the policy so that the information is up to date and correct, according to customer requirements. Also called mid-term adjustments, the Policy Alterations functionality has an impact on the value of the insurance premium, as through changes in coverage, the insurance premium may increase or decrease.

Types of Policy Updates

After registering the Mid Term Adjustment (MTA) request, the new policy version is displayed in a new tab, so the client can see the updated information.

After registering an MTA request by clicking the **Register** button, a new tab appears next to the first one containing the MTA request. The new tab displays the updated policy after MTA changes, showing the entire policy form, as it is for the **Core Policy Admin** module, and all the fields are read-only.

The displayed updated policy is the current policy version:

- In Version **Draft** status for versioning before approving or declining the MTA request.
- In Version **Unapproved** when declining the MTA status.
- After approving the MTA request, the **Approved** policy version is displayed.

Registering and Canceling an MTA Request

After registering an MTA request, the user can register or cancel it, either after performing the alterations on it and validating them, or before attempting to work on the request.

There are 2 buttons on the bottom of the page:

- **Register:** This button is used to register the request and proceed with the MTA process.
- **Cancel:** This button is used to cancel the request if the user has changed his mind.

Click the **Register** button to trigger the following actions:

- The change policy request transitions from **Draft** to **Registered**.
- All fields become read-only.
- The **Updated Policy** tab becomes available displaying the entire form of the updated policy.

- Core Policy Admin calculates the additional premium. For the discounted amount, a negative amount is displayed, as per the formula explained below.
- The policy alteration type is determined. If at least one included alteration requires the issuance of an MTA, the value is MTA needed.

After updating a policy alteration request, the coverages on a policy by editing the Amount Insured, removing or adding new coverages, the impact in the premium is calculated sending a `getPrices` API to receive the new annual premium amount calculated for each coverage (item). The request is sent after the validation of each alteration type (**Validate** button).

The calculations are given below:

- ***The additional coverage premium amount*** = $(\text{new annual premium} - \text{initial annual premium}) / 12 * \text{No. of uninsured months}$
- ***Updated coverage premium amount*** = $\text{Initial premium amount} * (12 - \text{No of uninsured months}) / 12 + \text{The additional coverage premium amount}$
- ***Additional policy premium amount*** = $\text{Sum (The additional coverage premium amounts)}$
- ***Updated policy premium amount*** = $\text{Sum (Updated coverage premium amount)}$

The adjustments are made according to the new amount, the rest of the installments which are unpaid and having the status **OnTime**.

The rounding in case of inaccurate calculations is done at the first unpaid installment and without an issued statement.

For the following actions on an MTA request, the request is made in order to return the monthly premium amount:

- Coverage addition;
- Coverage removal;
- Amount Insured edited;
- Change frequency type.

Approving and Declining an MTA Request

An MTA user approves or declines an MTA request according to the client's decision regarding the modifications on the policy.

Beside the displayed policy information, the **Change Policy Request** tab contains the **Accepted** or **Declined** buttons, which trigger specific status transitions.

Click the **Accepted** button to trigger the following actions:

- The policy transitions from **Registered** to **Accepted** status.
- The MTA No is updated from the first tab in the policy alteration summary.
- The specific transition for policy versioning is made to **Approved**.

By clicking the **Declined** button, the following actions are triggered:

- The policy transitions from the **Registered** to the **Declined** status.
- The specific transition for policy versioning is made to **Unapproved**.

NOTE The actions and the buttons are displayed only for the change policy requests in **Registered** status having displayed the **Updated Policy** tab, otherwise, the buttons are not visible and the actions not possible.

MTA Refactoring According to the Pro Rata Type

The Premium calculations made in the Policy Alteration processes take into consideration the type of pro rata which will be used in the calculations.

The type of pro rata is configured in the pro rata type insurance parameter.

The parameter contains 2 types of pro rata:

- Daily pro rata as $1/365$ from the premium amount of a contract;
- Monthly pro rata as $1/12$ from the premium amount of a contract.

Parameter location: Settings - Insurance parameters

Parameter name: Prorata type

Parameter code: PRT

Parameter type and value: Option set: either Daily or Monthly

For the premium adjustments, either the daily pro rata or the monthly pro rata can be chosen.

If Prorata type = daily

- ***AdditionalCoveragePremiumAmount*** = $(\text{coverageNewPremium} - \text{initialPremiumAmount}) / 365 * \text{uninsuredPeriod}$
- ***UpdatedCoveragePremiumAmount*** = $\text{inita}l\text{PremiumAmount} * (365 - \text{uninsuredPeriod}) / 365$
- ***FreqAdditionalCoveragePremiumAmount*** = $(\text{newPremiumAmount} - \text{initialPremiumAmount}) / 365 * \text{uninsuredPeriod}$
- ***FreqUpdatedCoveragePremiumAmount*** = $(\text{initialPremiumAmount} * (365 - \text{uninsuredPeriod}) / 365) + (\text{newPremiumAmount} / 365 * \text{uninsuredPeriod})$

If Prorata type = monthly

- ***AdditionalCoveragePremiumAmount*** = $(\text{coverageNewPremium} - \text{initialPremiumAmount}) / 12 * \text{uninsuredPeriod}$
- ***UpdatedCoveragePremiumAmount*** = $\text{inita}l\text{PremiumAmount} * (12 - \text{uninsuredPeriod}) / 12$
- ***FreqAdditionalCoveragePremiumAmount*** = $(\text{newPremiumAmount} - \text{initialPremiumAmount}) / 12 * \text{uninsuredPeriod}$
- ***FreqUpdatedCoveragePremiumAmount*** = $(\text{initialPremiumAmount} * (12 - \text{uninsuredPeriod}) / 12) + (\text{newPremiumAmount} / 12 * \text{uninsuredPeriod})$

New variables are created at the Policy Alteration level:

- **uninsuredPeriodType** = monthly/daily according to the pro rata type parameter set
- **uninsuredPeriod** = calculated as number of months or number of days from Policy Begin Date to Policy End Date according to the uninsured period type set
- **coveredValidityType** = monthly/daily according to the pro rata type parameter set
- **coveredValidityPeriod** = calculated as number of months or number of days from Policy Begin Date to Policy End Date according to covered validity type set

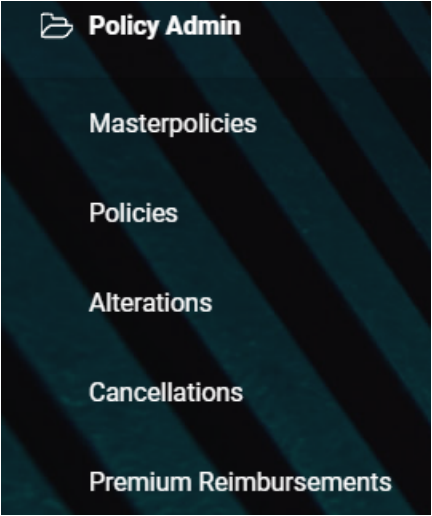
Master Policy Contract

This functionality enables insurers to create their bundled policy offerings, covering general insurance, healthcare, and life protection. It offers the ability to manage a mix and match of insurance coverages that can be extended to family members, affinity groups, employees and other logical groups.

Master Policy Generation

You can manually generate a Master Policy in the system through a journey, in order to administer a product where there are multiple policies issued under the same Master Policy.

In FintechOS Portal, in the menu, under the **Quote and Apply** section, the **Policy Admin** section is found, containing the **Masterpolicies** subsection.



Click the **Master Policies** subsection. A grid called **Master Policies List** is displayed, which contains all the created master policies.

MASTER POLICIES						
<input type="checkbox"/>	Master Policy No	Contractor	Start Date	Premium Amount	Currency	Business Status
Q	Q	Q	Q	Q	Q	Q
	90000	John Doe	30/09/2021	21,000.00	EUR	Inforce
	ZX1123334	Jane Doe	01/10/2021	12,000.00	EUR	Proposal
	SA123232	Jack Smith	01/10/2021	13,444.00	EUR	Issued
	354353	Drucila Winters	30/09/2021	11,222.00	EUR	Cancelled
	1212132131	Sally Green	30/09/2021	4,433.00	EUR	Draft
	12121219	Daniel Haner	01/08/2021	12,000.00	EUR	Issued

The grid contains the following columns:

Name	Description
Master Policy No	The number of the Master Policy.
Contractor	The full name of the contractor.
Start Date	The start date of the Master Policy.
Premium Amount	The premium of the Master Policy.
Currency	The currency of the Master Policy.
Business Status	The business status of the Master Policy.

You can manually add a new master policy by clicking the **Insert (+)** button on the top right corner of the screen. Fill in the newly opened form:

- The **Master Policy** tab contains the following sections:
 - **Contractor**, containing the following fields:

Name	Description
Name	The full name of the contractor, concatenated.
PIN	Personal Identification Number.
Email	The email of the contractor.
Phone	The phone number of the contractor.

Choose a **Contractor** from the drop-down to automatically fill in the fields with their details.

CONTRACTOR

Contractor

John Doe

Name

John Doe

PIN

12345

Email

john.doe@gmail.com

Phone

0040712345678

- **Intermediary**, containing the following fields:

Name	Description
Agent Name	The full name of the agent.
Broker Name	The full name of the broker.

Only one of the **Agent** or **Broker** name fields can be filled. If you try to fill both, the following message is displayed: "A Master Policy can have either an Agent or a Broker".

INTERMEDIARY

Agent

Jane Doe

FTOS_IP_InsuranceBrokerId

Jill Smith

Agent Name

Jane Doe

Broker Name

Jill Smith

- **Master Policy**, containing the following fields:

Name	Description
Master Policy No	The unique number of the master policy.
Quote Number	The unique number of the quote.
Validity Type	Drop-down field with the following possible values: Days, Months, or Years.
Validity	The validity of the master policy expressed in the number of days, months, or years.
Start Date	The start date of the policy.
End Date	The end date of the policy.
Payment Type	Drop-down field, with the following possible values: Bank Transfer or PayU.
Payment Frequency	Drop-down field with the following possible values: annually, semiannually, quarterly, monthly.
Currency	The currency used for the policy.
No of Installments	The number of installments for the policy.
Premium Amount	The amount of the premium.
IPT Amount	The amount of the insurance premium tax.
Master Policy Document	The master policy file to be uploaded.
Renewed Master Policy	Option set to choose the master policy to be renewed.

- **Mentions**, where the user can add some free text, with mentions regarding the Master Policy.

- Fill in these details and save the record. Two more tabs are displayed in the form:
 - Tab 1, first called **Master Policy** and now called **Master Policy Summary**, containing the above fields, plus, below them, 2 new sections called **Premium Payments Schedule** and **Invoices**.
 - These sections are empty at first, and populated with values after the associated policies are generated:
 - The **Premium Payments Schedule** section, the list with the installments, contains:

Name	Description
Installment No	The number of the installment.
Amount	The premium amount for that record, for that installment.
Due Date	The due date for that installment.
Business Status	The status of the installment.

Statement No	Amount	Currency	Due Date	Business Status
1	437.00	USD	11/10/2021	OutTime
2	437.00	USD	01/10/2022	OutTime
3	437.00	USD	01/10/2022	OutTime
4	437.00	USD	01/10/2022	OutTime
5	437.00	USD	11/10/2022	OutTime
6	437.00	USD	01/10/2023	OutTime
7	437.00	USD	01/10/2023	OutTime
8	437.00	USD	01/10/2023	OutTime
9	437.00	USD	11/10/2023	OutTime
10	437.00	USD	01/10/2024	OutTime

- The **Invoices** section, the list with the statements, contains the following columns:

Name	Description
Statement No	The number of the statement.
Amount	The premium amount for that record, for that statement.
Due Date	The due date of that statement.
Status	The status of the statement.

- Tab 2, called **Policies List**, is empty at first, and is populated with values after the associated policies are generated through the PolicyGenerationAPI endpoint.
 - The grid contains the following columns:

Name	Description
Insured Name	The full name of the insured person.
PolicyNo	The number of the policy.
Insurance Product	The product type.
Policy Start Date	The start date of the policy.
Policy End Date	The end date of the policy.
Premium Amount	The amount of the premium.
Currency	The currency used for the policy.
Business Status	The status of the policy.

All the above fields are mandatory, except Renewed Master Policy and the **Mentions** section.

Master Policies Versioning History

A **History** tab is available for each Master Policy with all the versions related to it, in order to keep track of the version number, starting from 1.

1. The **History** tab is displayed for the first time when the Master Policy is in the **Proposal** status, and the first version status of the Master Policy displayed in the **History** tab is **Proposal**.

Label	Attribute Version Date	Attribute Version	Modified by user
VersionUnapproved	12/06/2021 09:47	2	user
Proposal	12/06/2021 02:00	1	user

2. After the Master Policy is issued, the version displayed in the **History** tab is **Issued**. The previous version is not displayed anymore.
3. When the Master Policy is in **Inforce** status, this is the version displayed in the **History** tab. The previous versions are not displayed.
4. When a version is approved, the previous version is closed , having the **Version Closed** status. This automatically starts with the effective date of the approved version. Unapproved versions are also displayed in the **History** tab.
5. There cannot be 2 or more **Draft** or **Pending** versions at the same time. Also, there cannot be a **Pending** and a **Draft** version at the same time.

The **History** tab contains a grid with the following columns:

Column Name	Description
Label	Contains the name (status) of the versions.
Attribute Version Date	Contains the date and hour when the version is created in Draft status.
Attribute Version	Contains the number of the version. The versions are displayed in reverse chronological order.
Modified by User	Displays the name of the user editing the policy.

generateMasterPolicy Endpoint

The generateMasterPolicy endpoint is used to provide another way of adding a Master Policy into the system.

Request:

- Contractor (ID);
- Name;
- Personal Identification Number;
- Email;
- Phone;
- Agent (ID);
- Agent Name;
- Broker (ID);
- Broker Name;
- Currency;
- Renewed Master Policy;
- Quote Number;
- Start Date;
- Validity Type;
- Validity;
- Payment Frequency;
- Payment Type.

Response:

- Master Policy ID;
- Master Policy No.

NOTE

The only validation for the endpoint is that a Master Policy cannot have both an agent and a broker, but only one of the two.

The End Date of the Master Policy is calculated based on the Start Date and Validity.

PolicyGenerationApi Endpoint Update

The PolicyGenerationApi endpoint is updated in order to take into account the associated master policy terms and to calculate the premium amount and installments amounts of the master policy based on the allocated policies.

Generate Policy Endpoint - Initial Request

The system needs to take into consideration the information from the master policy for the generated policies.

For the policies which are included in the initial version of the master policy, when the master policy is initially issued, the Generate Policy endpoint must be updated in order to take into consideration the master policy number. The master policy number must be added to the request.

If there is a policy that is generated in the context of a master policy, the policy attributes are inherited from the associated master policy. Otherwise, for normal policies, the master policy ID is null.

NOTE

The validity of a policy that has a master policy is the difference between the Master Policy End Date and the Policy Begin Date.

For the generation of a master policy, the system sends in the request some info. The same info is used for the Generate Policy endpoint for each policy in case there is a master policy number.

Master Policy update - Premium Calculation and Payment Schedule Update

The first update is made after the initial request, when the first set of policies is generated when calling the Generate Policy endpoint.

After the associated policies are generated by calling the Generate Policy endpoint initial request, the premium amount, IPT amount and the installments amounts of the master policy are updated and calculated based on the allocated policies.

The master policy transitions from the **Draft** into the **Proposal** status at update time.

NOTE

The payment schedule of the master policy is updated. For the master policy, the system uses the same entities as for the policy, meaning: FTOS_INSQB_PaymentSchedule and FTOS_INSQB_PaymentScheduleDetail. Therefore, a lookup attribute, Master Policy ID, must be added for these entities.

PolicyMaturity Job Update for Master Policy

The PolicyMaturity job is updated in order to have the same behavior for the Master Policy.

The FTOS_PA_PolicyMaturity job runs daily at 23:59 and is set in order to automatically transition policies' status from **In Force** or **Suspended** to **Maturity** status when the policies' Current Date is greater than the End Date.

The logic is extended also for a Master Policy regarding the transition from **In Force** to **Maturity**.

The job is updated in order to automatically transition Master Policies' status from **In Force** to **Maturity** when Master Policies Current Date is greater than the End date.

IssuedToEnforced Policy Job Update for Master Policy

The current job, called FTOS_INSPA_Policy_IssuedToEnforced, which runs daily at 01:00 AM, is set in order to automatically transition the policies' status from **Issued** to **InForce** when the policies' Begin Date is equal to the Current Date.

The logic is also extended for a Master Policy.

The job must be updated in order to automatically transition Master Policies' status from **Issued** to **InForce** when the Master Policies Begin Date is equal to the Current date.

Automatic Renewal

The Policy Automatic Renewal solution helps insurers to continue providing coverage to the customer once the initial policy period has passed, at the end of the term period, so the beneficiary never goes without coverage in any field of insurance. You can opt for this automatic renewal once you initiates your first insurance policy, so as to get rid of the worry of manual renewal once the policy reaches maturity. What is important to note is that such automatic renewals may also include changes in the premium insured by rate increases or decreases.

Configuring Policy Automatic Renewal

In order to configure whether the contracts under a specific Insurance Product should be automatically renewed or not, you must set the **Renew Type** on the Insurance Product level.

The screenshot displays the 'PRODUCT CONFIGURATIONS' interface for a policy named 'Holiday Home Insurance'. The interface is divided into two main sections: 'POLICY COVERAGE' and 'POLICY ADMIN'. In the 'POLICY ADMIN' section, the 'Renew Type' dropdown menu is highlighted with a red box. The dropdown menu is open, showing four options: '[none]', 'No', 'Automatic renewal', and 'Renewal offers'. Other configuration fields visible include 'Grace Period' (1), 'Total Indemnity Limit' (56,000), 'Free Withdrawal Period Limit' (14), 'Type of Suspension' (Specific), and 'Premium Update due to Suspension' (Monthly).

You may choose between:

- No - no renewal applied for the current product and subjected insurance policies;
- Automatic renewal - the process of automatic renewal is applied according to some extra configurations triggered by this option (see below);
- Renewal offers - a renewal offer is generated as a JSON object for the policies which have to be renewed under the current product.

If you configure your product to Automatic renewal, then you have to fill out the following fields:

PRODUCT CONFIGURATIONS

POLICY COVERAGE

Grace Period: 1 | Grace Period Type: Months

Total Indemnity Limit: 56,000

POLICY ADMIN

Free Withdrawal Period Limit: 14

Renew Type: Automatic renewal

Renewal Validity: Yearly | Renewing Policy: Same policy

Renewal Tariff: Same tariff | No of Days Before Renewal: 5

Renewal Validity - the new validity which is applied for the correlated contracts of current product:

- Yearly - the new validity is set for 12 months (a year) in order to be automatically renewed next year, so that the new End Date of the renewed policy is the Start Date + Policy validity where the value is retrieved according to Renewal validity;
- Monthly - the new validity is set for x month in order to be renewed, so that the new End Date of the renewed policy is the Start Date + Policy validity where the value is retrieved according to Renewal validity;
- Same Validity - the new validity is calculated as for the previous policy to be renewed, so that the new End Date of the renewed policy is the Start Date + Policy validity where the value is retrieved according to Renewal validity; also, for this calculation, the values retrieved from the old policy are kept in the Policy Validity and Validity type attributes.

Example 1:

For the previous policy: Policy Validity = 400 and Validity type = days

For the new policy: Start Date = Renewed policy End Date + 1 and End Date = Start date + 400 days

Example 2:

For the previous policy: Policy Validity = 13 and Validity type = months

For the new policy: Start Date = Renewed policy End Date + 1 and End Date = Start date + 12 months

Renewing Policy - throughout the automatic renewal process, policies can be renewed as new policy records in the system, keeping the old policy records with a final status, or they can be renewed by creating a new version of the same old policy records:

- Same policy - policy new version, keeping the same record in the database;
- New policy - brand new policy as new record in the database.

Renewal Tariff - the calculation of renewed policies can be either kept as for the previous policy that have been renewed or it can be calculated according to the current insurance product tariff version

- Same tariff - keeping the same tariff as for the old policy which have been renewed without taking into consideration the current insurance product tariff version;
- Actual tariff - taking into account the current insurance product tariff version and calculate the premium amount with the new values configured on product level.

IMPORTANT!

In order to take into account the new insurance product tariff configuration, the insurance product must be an approved version!

No. of Days Before Renewal - tis stores the number of days before renewal, more precisely, with how many days before the policy End Date, a policy should be renewed; the automatic renewal scheduled job looks after this value in order to renew the policies which meet this renewal condition x days before End Date.

Example:

If No. of days before renewal = 2, the policies after which the automatic renewal scheduled job looks after are the policies with

Policy End Date (old policy Initial End Date) - No. of days before renewal = Current Date and applies the renewal process to them

15.01.2022 (old policy Initial End Date) - 2 days (No. of days before renewal) = 13.01.2022 (Current Date) → the policies with this End Date is renewed.

NOTE

Policy Initial End Date ≤ Current Date + x days before renewal and Policy Initial End Date > Current Date

Renewal Offers

If at product level, the **Renewal Type** is set to Renewal offer, then the policies found to be renewed generate a JSON object as an insurance renewal offer containing information about the new policy renewed.

```

1  {
2  "startDate": "2022-12-19",
3  "validity": 12.0,
4  "validityType": "Months",
5  "issuedDate": "2021-12-17",
6  "totalIndemnityLimit": null,
7  "isRenewal": true,
8  "renewedPolicyId": "3bdfb0b6-5a91-48ac-854f-59e5e1650622",
9  "mentions": "Insert comment here",
10 "quoteNo": "User0063",
11 "noOfRenewals": 1.0,
12 "insuranceTypeName": "Personal Accidents",
13 "productCode": "PA",
14 "agent": {
15   "agentId": null,
16   "agentType": "Individual person"
17 },
18 "broker": {
19   "brokerId": null,
20   "distributionChannel": null
21 },
22 "contractor": {

```



```

23  "uniqueIdentifier": "1911110223344",
24  "firstName": "Praslea",
25  "lastName": "NoMail",
26  "type": "Other"
27  },
28  "insured": {
29    "uniqueIdentifier": "1911110223344",
30    "firstName": "Praslea",
31    "lastName": "NoMail",
32    "type": "Other"
33  },
34  "beneficiary": {
35    "uniqueIdentifier": "1911110223344",
36    "firstName": "Praslea",
37    "lastName": "NoMail",
38    "type": "Other"
39  },
40  "currency": "RON",
41  "paymentType": "brokerCollection",
42  "paymentFrequency": "monthly",
43  "renewedPolicyNo": "80001342",
44  "insuranceProductItemList": [
45    {
46      "code": "MEACC",
47      "insuredAmount": 20000.0,
48      "finalPremiumAmount": 691.2
49    },
50    {
51      "code": "ICPA",
52      "insuredAmount": 25000.0,
53      "finalPremiumAmount": 792.0
54    },
55    {
56      "code": "PDA",
57      "insuredAmount": 35000.0,
58      "finalPremiumAmount": 1209.6
59    },
60    {
61      "code": "DPA",
62      "insuredAmount": 50000.0,
63      "finalPremiumAmount": 144.0
64    }
65  ]
66  }

```

Automatic Renewal Process

The daily scheduled job, FTOS_PA_PolicyRenewal, is running in order to find all the policies from the system which have to be renewed according to their end date and the parameter set for the number of days before renewal mentioned above.

The job verifies all the policies which have the Policy End Date (policy to be renewed Initial End Date) - No. of days before renewal (parameter set) = Current Date, and applies the renewal process to them.

The policy automatic renewal types are described below:

New Policy/Same Validity/Actual Tariff

The system generates renewal offer policies for those which are due to expire and have the following product configuration:

- **Types of Renewal:** New Policy;
- **Renewal Validity:** Same Validity;
- **Renewal Tariff:** Actual Tariff.

The policies which need to be renewed, need to generate insurance offers which need to include the following information:

- Status of newly renewed policy - Proposal;
- Validity - Renewed Policy validity;
- Begin Date = Renewed policy End Date + 1;
- End Date is calculated according to the Begin Date + Validity;
- Same parties;
- Same insured object;
- Same agent/broker and distribution channel;
- Same quote number;

- Same payment type and the same payment frequency;
- Same coverages and indemnity limits;
- The premium has to be calculated based on the currently approved product version;
- The Payments Schedule is generated according to the standard logic;
- The old policy has to be mapped as renewed and have completed the Renewed by attribute with the new Policy number;
- For the new policy, save the Renewed Policy number of the old policy.

Renewal Offers/Same Validity/Actual Tariff

The system generates renewal offer policies for those which are due to expire and have the following product configuration:

- **Types of Renewal:** Renewal Offers;
- **Renewal Validity:** Same Validity;
- **Renewal Tariff:** Actual Tariff.

The policies found through the FTOS_PA_PolicyRenewal job which need to be renewed, need to generate insurance offers which need to include the following information:

- Renewed policy;
- Same Product and Insurance Type;
- Validity: Renewed Policy validity;
- Begin Date: Initial Policy End Date + 1;
- End Date is calculated according to the Start date + Validity;
- Same parties;
- Same insured object;

- Same agent or broker and distribution channel;
- Same payment type and the same payment frequency;
- Same coverages and indemnity limits;
- The total premium has to be calculated based on the currently approved product version.

New Policy/Same Old Tariff/Same Validity

The system automatically renews policies having the same premium amount. These are updated as new policies to update the system's contract. The configuration is given below:

- **Renewing Policy** = New Policy;
- **Renewal Validity** = Same Validity;
- **Renewal Tariff** = Same Tariff.

The policies found through the FTOS_PA_PolicyRenewal job which need to be renewed, for products with the above configuration are renewed as follows:

- Status: Proposal;
- Validity: Renewed Policy validity;
- Begin Date: Initial Policy End Date + 1;
- End Date is calculated according to the Start date + Validity;
- Same Parties;
- Same insured object;
- Same agent or broker and distribution channel;
- Same Quote;

- Same payment type and the same payment frequency;
- Same coverages and indemnity limits;
- The premium has to be calculated based on the product version valid on the issuance date of the initial policy;
- The **Payment Schedule** is generated according to the standard logic;
- The old policy has to be mapped as renewed (existing attribute) + have completed the Renewed by attribute with the new policy no;
- For the new policy, we need to save the Renewed Policy no of the policy (existing attribute);
- for the new policy, Start Date = Renewed policy End Date + 1 and End Date is calculated according to the Start date + Policy validity (to be taken from Renewal Validity attribute).

Renewal Offers/Same Validity/Same Tariff

The system generates renewal offer policies for those which are due to expire and have the following product configuration:

- **Types of Renewal:** Renewal Offers;
- **Renewal Validity:** Same Validity;
- **Renewal Tariff:** Same Tariff.

The policies found through the FTOS_PA_PolicyRenewal job, which need to be renewed, need to generate insurance offers including the following information:

- Renewed policy;
- Same Product and Insurance Type;
- Validity - Renewed Policy validity;
- Begin Date = Initial Policy Begin Date + 1;

- End Date is calculated according to the Start date + Validity;
- Same Parties;
- Same insured object;
- Same agent/broker and distribution channel;
- Same payment type and the same payment frequency;
- Same coverages and indemnity limits;
- The total premium is calculated based on the product version valid at the issuance date of the renewed policy.

Policy Configurations

The **Core Policy Admin** module keeps a traceability during the life period of an insurance contract and its adjustments through time. Check the following pages to find out more about how this solution works:

[Business Functionalities](#) - for details about manual and automatic flows.

[Flow Parameters and Scheduled Jobs](#) - for details about the flow parameters and scheduled jobs.

[Business Workflows](#) - for details about statuses and transitions.

[Mid Term Adjustment Journeys](#) - for details about the mid term adjustment journeys.

[Policy Cancellation Journeys](#) - for details about the policy cancellation journeys.

[Other Automated Journeys](#) - for details about other automated journeys.

[Policy Generation API](#) - for details about the Policy Generation API.

[Get Policy Data API](#) - for details about the Get Policy Data API.

[Policy Status Change API](#) - for details about the Policy Status Change API.

[Policy Versioning](#) - for details about the versioning mechanism.

[Core Policy Admin Formulas](#) - for details about the Core Policy Admin Formulas.

Business Functionalities

Here are the **Core Policy Admin** functionalities, from a business perspective:

Automated Flows

When they satisfy some given conditions for changing their business status, policies can automatically be handled by the solution. **Core Policy Admin** has the capability to meet the following policy administration needs, without intervention from an operator:

Policy Origination

The Policy issuance process represents the main **Core Policy Admin** functionality through which the insurance contracts are generated within the Core system, on the basis of which the other **Core Policy Admin** processes are based.

From a business perspective, this functionality involves an end-to-end process that starts from the generation of the initial policy through a specific endpoint, is updated through an update endpoint and then moves to **InForce** business status, according to the contractual **Start Date**.

Policy origination endpoint

The policy issuance endpoint brings business value to the **Core Policy Admin** component by creating a connection between various external systems and the core system. Thus, once the integration with another system in order to take over the information is achieved, the policy is generated in the system according to the information received with reference to the policy to be created.

Within this endpoint, a policy generation object is structured which contains all the information that must be taken over within an integration with an external system.

Following the request made, the response body executes the populating of the system with new data specific to the policy, in addition to those obtained during the data collection from an external system.

Other Automated Flows:

Policy lapsing

When they satisfy some given conditions for lapsing, policies can automatically be moved from **InForce** to **Lapsed** status. Lapsing occurs when there is no payment, for an agreed period of time, of the latest installment on the contracted policy. The lapsing process of a policy represents an **automatic process** performed by means of

configuration items specific to the lapsing process: the DAUDD insurance flow parameter and the FTOS_PA_Policy Lapsed [scheduled job](#).

Policy maturity

When they satisfy some given conditions for termination, policies can automatically be moved from **InForce** to **Maturity** status. The solution has an automated job in place in order to help with moving the policies to **Maturity** status.

Policy renewal

When they satisfy some given conditions for renewal, policies can automatically be moved from **Maturity** to **Renewed** status. The solution has an automated job in place in order to help with moving the policies to ready for renewal procedures.

InForce Flow

The enforcement process is a straightforward process that aims to activate the policies in the system that have an initial status, meaning that the policies are currently active and providing insurance coverage in return for premiums paid as agreed. The phrase “in force” refers to the policy at the time it is evaluated.

With their enforcement, the policies become valid and various other operations can be performed within them, such as: inclusion in reports, activating the statement generation process and the possibility of paying them for the policies in question, going through termination processes and more.

During its lifetime, until the **End Date** set on the contract level, the policy is **InForce** as long as the policy holder has been paying his insurance as per the payment agreement or has paid their premiums in full amount.

When a policy is deemed **InForce**, the policyholder is entitled to the policy's benefits. Depending on the type of insurance and policy purchased, the **InForce** time and date may vary. In addition, the conditions for when the policy lapses due to missed payments or what causes the policy to be voided are also configurable.

Process description

The enforcement process involves changing the status of the policies in the system - having the **Issued** status, once they reach the **Policy Begin Date**. There are 2 possibilities to trigger this process:

1. **Automatically** - if the current product is configured with the **Insurance Product Factory** to allow the **Automatic InForce process**.
2. **Specific request** in order to make the policy **InForce**.

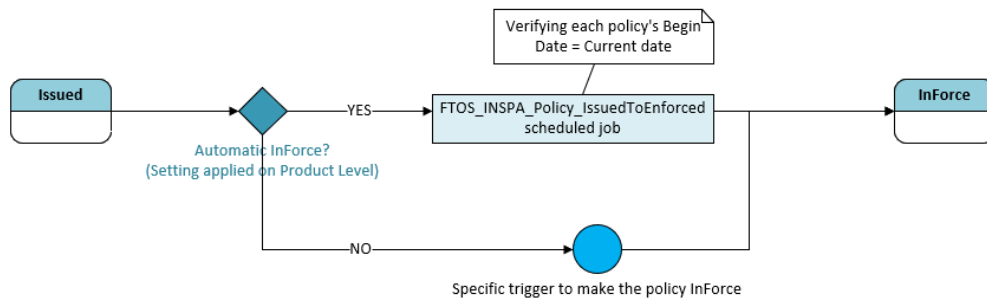
In the picture below: **Automatic InForce** configuration at the level of the product, made with the **Insurance Product Factory** solution.

The screenshot displays the 'INSURANCE PRODUCT' configuration page. At the top, there's a header bar with navigation icons and a status section showing 'CURRENT STATUS: APPROVED' and 'NEXT STATUS: CLOSED'. Below this, a table lists product details: NAME (Household NN), START DATE (01/01/2020), CURRENCY (RON), and TYPE (Home). The main configuration area includes fields for Insurance Type (Home), Insurance Product Code (HH0), Name (Household NN), Currency (RON), Start Date (01/01/2020), End Date (01/01/2050), Grace Period (0), Grace Period Type (Months), Maximum Discount (0), Maximum Commission, Total Indemnity Limit, Allow Renew (checked), RenewType (Renewal offers), and a red-bordered checkbox for 'Automatic InForce' which is also checked.

1. Automatic InForce = True

In order to automate this process, within the system there is the related configuration item, more precisely the **scheduled job FTOS_INSPA_Policy_IssuedToEnforced** which involves a daily run in order to check the condition of the Policy Begin Date being the Current Date for all policies in the system, in **Issued** status. Once this condition is met, for the related policies, the status is automatically changed to **InForce**.

In the picture below: the policy InForce flow



2. Automatic InForce = False

If a method other than **Automatic InForce** is employed, this process may occur following an external request that triggers the policy change from **Issued** to **InForce**.

Thus, such a trigger can come either from a specific call through policy update endpoint, or through an automatic trigger according to specified business requirements such as the integration of Policy Confirmation Files that, once arrived and processed in the system, it also changes the status of the policy - if the **Policy Begin Date** is the **Current Date**.

Policy Cancellation

The Cancellation flow is used to manage the process of policy termination. It allows the user to find a desired policy based on some searching parameters and to create and manage a policy termination request. The solution allows policy operators to achieve efficiency in a timely manner.

A **Core Policy Admin** user registers any customer termination request based on the following steps:

- Identify the policy and the customer
- Calculate Returned Premium based on Returned Premium rules
- Send request to approval
- Continue with the Payment Return flow

A **Core Policy Admin** super user should approve the termination request if **Returned Premium** is greater than 0. The system creates payments for each approved request with **Returned Premium** greater than 0.

Explore more details about the [business steps and transitions](#) and the [conditions, parameters and jobs](#) related to the above mentioned automatic flows that already accommodated inside the **Core Policy Admin** module.

Flow Parameters and Scheduled Jobs

The following flow parameters and scheduled jobs are used with the **Core Policy Admin** solution.

1 Flow Parameters

Parameter Name	Policy Automatically Withdraw
Details	Type: Integer; Code: PWDAY
Component	Policy Admin
Correlated with	FTOS_INSPA_Policy_AutomaticallyWithdraw scheduled job
Description	This parameter sets the Automatically Withdrawal Day of a policy after a number of days since the First Installment on that policy moved to Unpaid status. When the parameter is fulfilled, the policy is transitioned to Withdraw status.

Parameter Name	maturityNotification
Details	Type: Integer; Code: MTN
Component	Policy Admin, Notifications
Correlated with	N/A

Parameter Name	maturityNotification
Description	This parameter sets the number of days for notifying the policy Maturity status to the policy holder, after the policy reaches that specific status. For example: 5 days after the policy reaches Maturity status, a maturity notification is sent to the contract holder informing that the policy reached its end date.

Parameter Name	Days After Unpaid Due Date
Details	Type: Integer; Code: NDBR
Component	Policy Admin, Billing & Collection
Correlate with	FTOS_PA_PolicyLapsed scheduled job
Description	This parameter sets the number of days after an unpaid installment's due date, registered on a policy. When the DAUDD parameter is fulfilled, the policy is transitioned to Lapsed status. For example: the Lapsing Day of a policy is triggered after the passing of the chosen number of days since the Last Installment on that policy moved to Unpaid status.

Parameter Name	No of Days Before Renewal
Details	Type: Integer; Code: NDBR
Component	Policy Admin
Correlated with	FTOS_PA_PolicyRenewal scheduled job
Description	This parameter correlates the starting date of the Renewal process with a given number of days before the End Date of a policy. For example: it is required to start the Renewal process within 5 days before policy reaching its End Date . As a result the parameter is set to 5 days.

2 Scheduled Jobs

Job Name	FTOS_PA_PolicyLapsed
Scheduled	At 04:00 AM, daily run

Job Name	FTOS_PA_PolicyLapsed
Description	<p>For policies with last installments in Unpaid status. This job automatically moves policies from Enforced to Lapsed status, when the DAUDD parameter is fulfilled.</p> <p>When in Lapsed, the policy is modified as follows:</p> <ul style="list-style-type: none"> • The End Date becomes the Due Date of the unpaid installment. • If there are no claims registered, the Premium Amount becomes equal to the Paid Premium (sum of paid installments). If there are claims registered on the policy, the Premium Amount doesn't change. • If the Premium Amount is changed, the installment schedule is updated including just the paid installments. <p>The system also checks the Master Policies payment and, if there will be any installments still unpaid, being in the Unpaid status) after X days for a Master Policy (X days since the due date of the last unpaid installment of the Master Policy), then that Master Policy changes its status from Inforce to Lapsed.</p> <p>After a Master Policy is moved in the Lapsed status, that Master Policy is modified as follows:</p> <ul style="list-style-type: none"> • The end date becomes the due date of the unpaid installment. • The new premium amount is equal to the paid premiums (the sum of paid installments). • Because the premium amount is changed, the payment schedule of the Master Policy is updated and only includes the paid installments. <p>The number of days is set in the "Days after unpaid due date" existing and generic parameter, set at portfolio/contract level.</p>

Job Name	FTOS_INSPA_Policy_AutomaticallyWithdraw
Scheduled	At 02:00 AM, daily run

Job Name	FTOS_INSPA_Policy_AutomaticallyWithdraw
Description	This job moves policies from Proposal to Withdraw status when they satisfy the following conditions: their first premium is still in Unpaid status after a given number of days - PWDAY parameter, after their first due date.

Job Name	FTOS_INSPA_Policy_DeleteAfter10m
Scheduled	At 01:00 AM, daily run
Description	This job deletes from the system the policies with no business transition in the last x minutes - number of minutes set in the NMACLBWT parameter.

Job Name	FTOS_PA_PolicyRenewal
Scheduled	At 03:00 AM, daily run
Description	<p>This job looks into the system for the End Date and the NDBR parameter in order to identify the policies in InForce status that are suitable for automatic or manual renewal - for renewal offers.</p> <p>The job returns the policies with Policy End Date (Initial End Date) - No of days before renewal (parameter set) = Current Date.</p>

Job Name	FTOS_INSPA_Policy_IssuedToEnforced
Scheduled	At 01:00 AM, daily run
Description	<p>For the policies that are in the Issued business status, have the same Begin Date as the Current Date and have the Automatic InForce checkbox selected, at Product level.</p> <p>This job verifies whether the above conditions are fulfilled and moves policies from Issued to InForce status.</p>

Job Name	FTOS_PA_PolicyTerminationProcess
Scheduled	Every 50 minutes, starting at 2 minutes past the hour
Description	This job verifies whether the Policy End Date is equal to the Current date and moves InForce policies to Maturity status.

Business Workflows

Business workflows help organizations coordinate tasks between people and synchronize data between systems, with the ultimate goal of improving efficiency and responsiveness.

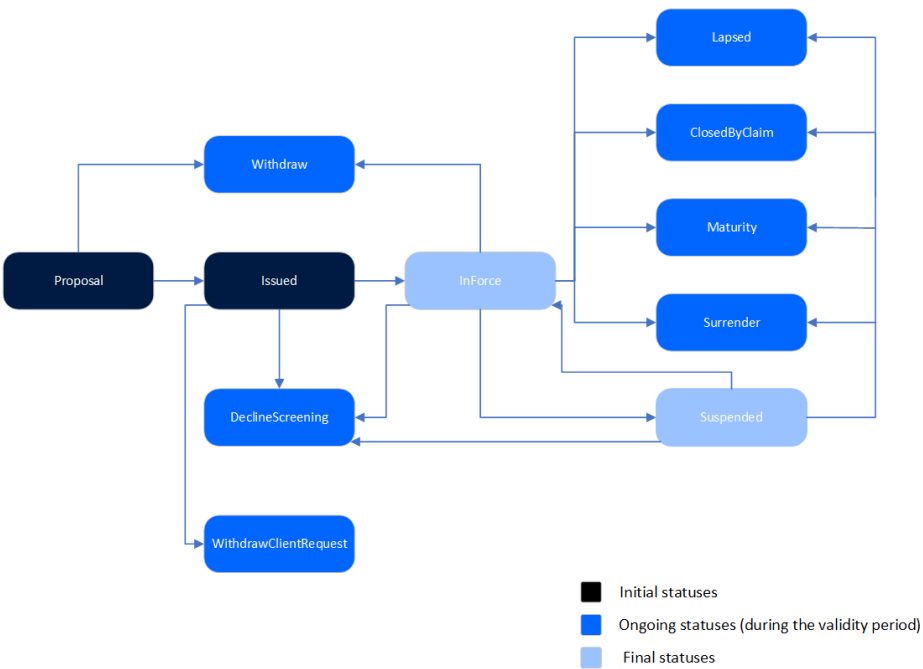
Workflow stages show the status of a record in the workflow and provide processing that must occur for the record to move to the next phase. The tasks, information or documents are passed from one status to another (from one participant to another for action) either manually or triggered by business rules or actions specified for that specific status.

Policy Workflow

The Core Policy Admin management system allows policy operators to achieve efficiency, gain flexibility and organize their insurance policy data for analysis and operational purposes in an easy way.

Policy State Machine

Here is a diagram with the transitions managed through the **Core Policy Admin** module:



Policy Statuses

The **Core Policy Admin** module accommodates the following business states for a policy:

Status name	Type	Description
Proposal	Initial	The first status for every policy generated into the system. Can be tailored to apply to policies waiting for their first installment to be paid.
Issued	Initial	For policies that passed their first Paid premium.
InForce	Ongoing	For policies passed their Begin Date that also meet all their contract terms - for ex. current date is not the End Date of the policy, payments made on time and so on.

Status name	Type	Description
Withdraw	Final	For policies closed before reaching the Issued status due to the following reasons: first premium not paid in a certain period following the proposal, conflicting parameter configurations between the insurance product and the policy or a system error - for ex. duplicated policy.
Cancelled	Final	For policies closed by the insurer for reasons other than those exposed in the . This status is reached through the cancellation flow only.
WithdrawClientRequest	Final	For policies closed following the policy holder request. This status is reached through the cancellation flow only.

Status name	Type	Description
ClosedByClaim	Final	For policies which exhausted their coverage to claims made by the policy holder. This status is reached through the cancellation flow only.
DeclineScreening	Final	For policies being pursued by a high risk customer. This status is reached through the cancellation flow only.
Lapsed	Final	For policies on which premiums were not paid, after the Payment Grace Period expired.
Maturity	Final	For policies passed their End Date .

Policy Status Transitions

Here is a description of the transitions managed through the **Core Policy Admin** module:

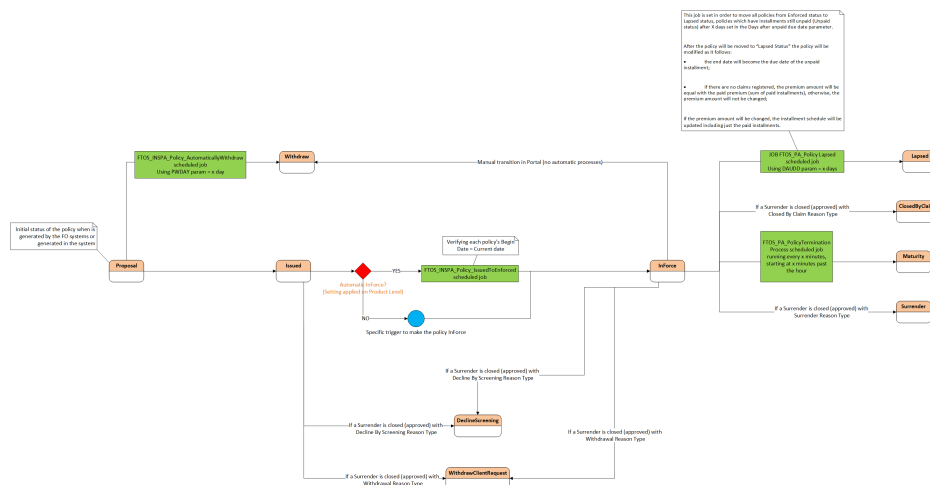
Transition	Description
_Proposal	The first status for every policy generated into the system.
Proposal_Issued	Automatic transition after policy generation in the system. Mention: This transition is properly used in a delivery project where the first paid installment triggers this status change.

Transition	Description
Proposal_Withdraw	Automatic transition before reaching the Issued status due to the following reasons: first premium not paid in a certain period following the proposal, conflicting parameter configurations between the insurance product and the policy or a system error - for ex. duplicated policy. This transition is triggered by FTOS_INSPA_Policy_AutomaticallyWithdraw scheduled job using the PWDAY parameter value.
Issued_InForce	This transition can be triggered in 2 different ways: Automatically - when the policy includes an insurance Product on which Automatic InForce is set to True , at product level, the FTOS_INSPA_Policy_IssuedToEnforced scheduled job moves the policies meeting the conditions for enforcement from Issued to InForce status. Manually - triggered by a specific request - for example Policy related endpoints, on policies that include Products with Automatic InForce set to False , at Product level.
Issued_DeclineScreening	Automatic transition triggered by choosing the Decline By Screening reason type, during the Cancellation flow.
Issued_WithdrawClientRequest	Automatic transition triggered by choosing the Withdrawal reason type, during the Cancellation flow.
InForce_Withdraw	Manual transition that can be made when the insurer decides that the policy needs to be cancelled and a Cancellation flow is not necessary - for example when policy duplication occurs due to operational mistakes.

Transition	Description
InForce_WithdrawClientRequest	Automatic transition triggered by choosing the Withdrawal reason type, during the Cancellation flow.
InForce_Surrender	Automatic transition triggered by choosing the Surrender reason type, during the Cancellation flow.
InForce_Maturity	Automatic transition for policies that reach their contractual Policy End Date . FTOS_PA_PolicyTerminationProcess scheduled job verifies whether the condition Policy End Date is Current Date applies to the policy and triggers the status transition. After transitioning, the Core Policy Admin keeps the policy status updated.
InForce_Lapsed	Automatic transition for policies on which premiums were not paid for a number of days, after the Payment Grace Period expired. FTOS_PA_PolicyLapsed scheduled job verifies each policy having this condition and triggers the status transition.
InForce_DeclineScreening	Automatic transition triggered by choosing the Decline By Screening reason type, during the Cancellation flow.
InForce_ClosedByClaim	Automatic transition triggered by choosing the Closed By Claim reason type, during the Cancellation flow.

Policy Status Transitions Map

Below is a map showing the transitions managed through the **Core Policy Admin** module in the **FintechOS** environment:



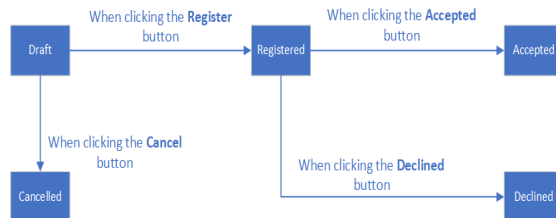
[Click here to download the above diagram in Visio format \(.vsdx\)](#)

Change Policy Request Business Workflow

The business statuses for a change policy request are described below:

- **Draft:** This is the initial status when inserting a new change policy request. In this status, all fields are editable, with a few exceptions.
- **Registered:** This is the intermediary status for an MTA request when the user clicks the **Register** button. In this status, all the fields become read-only. In this situation, from a business perspective, is the decision of the MTA user if continues with the MTA request or not.
- **Cancelled:** This is the final status if the user clicks the **Cancel** button in the interface. In this status, all the fields become read-only. In this situation, from a business perspective, is the decision of the MTA user if they continue with the MTA request or not.
- **Accepted:** This is the final status if the user clicks the **Accepted** button. In this status, all the fields become read-only. In this situation, from a business perspective, is the decision of the MTA user if they continue with the MTA request or not.

- **Declined:** This is the final status if the user clicks the **Declined** button. In this status, all the fields become read-only. In this situation, from a business perspective, is the decision of the MTA user if they continue with the MTA request or not.



Multipolicies Contract Business Workflow

- The first status of the master policy is **Draft**, after the user clicks the **Insert** button.
- The status moves from **Draft** into **Proposal** automatically when the associated policies are generated.
- From **Proposal**, the status can move to:
 - **Issued**, through a call from the **Quote&Bind** module.
 - **Withdraw**, automatically if the master policy status does not move from **Proposal** into **Issued** within a specific number of days.
- From **Issued**, the status can move to **Inforce**, when the current date is the same as the start date of the master policy.
- From **Inforce**, the status can move to:
 - **Cancelled**, due to different aspects, such as a duplicated policy, or on a client request but after the cooling off period.
 - **Lapsed**, when the installments are not paid until the due date.
 - **Maturity**, when the current date is the same as the end date of the master policy.
 - **WithdrawClientRequest**, on a client request for personal reasons, within the cooling off period.

Master Policy Payment Schedule Business Workflow

The trigger for the Master Policy installments status transitions are the installments statuses of the policies associated with said Master Policy.

There is a specific job, for Master Policies, which runs daily and checks the installments statuses of the policies that are included in a Master Policy and have the same due date.

Master Policy Payment Schedule Status Transitions

Here is a description of the transitions for the master policy payment schedule.

Transition	Description
_on time	When generated, all the Master Policy installments are generated in this status for the entire period. They are available, but without an issued statement.
on time_ statement issued	When the statement is generated, the statement is issued and all the installments are correlated on that statement, if all the installments of the associated policies have the Statement Issued status, then the Master Policy installment status is also Statement Issued .
statement issued_paid	If all the associated policies are paid before the due date, and they have the Paid status, then the Master Policy installment is also Paid .
statement issued_ unpaid	If at least one associated policy is not paid before the due date and has the Unpaid status, then the Master Policy installment status is also Unpaid .

Transition	Description
paid_statement issued	When a payment for at least an associated policy is deallocated, the statement of that policy is not covered anymore, so it becomes generated instead of closed or paid. The result being the Statement Issued status for the Master Policy installment also.
paid_unpaid	When a payment for at least an associated policy is deallocated, so the installment status for that policy becomes Unpaid , the installment status for the Master Policy also becomes Unpaid .

Policy Lapsed Job for the Master Policy

The current job, called **FTOS_PA_Policy Lapsed**, which runs daily at 04:00 AM, is set in order to move all policies from **Enforced** to **Lapsed** status, policies that have installments still unpaid (**Unpaid** status) after X days set in the "Days after unpaid due date" parameter.

After a policy is moved in **Lapsed** status, that policy is modified as follows:

- The end date becomes the due date of the unpaid installment;
- If there are no claims registered, the premium amount is equal to the paid premium (the sum of paid installments), otherwise, the premium amount is not changed;
- If the premium amount is changed, the payment schedule is updated and only includes the paid installments.

The system also checks the Master Policies payment schedule, as it does for policies, and, if there are installments that are still unpaid (**Unpaid** status) after X days for a Master Policy (X days since the due date of the last unpaid installment of the Master Policy), then that Master Policy changes its status from **Inforce** to **Lapsed**.

After a Master Policy is moved in **Lapsed** status, that Master Policy is modified as follows:

- The end date becomes the due date of the unpaid installment;
- The new premium amount is equal with the paid premiums (sum of paid installments);
- Because the premium amount is changed, the payment schedule of the Master Policy is updated and only includes the paid installments.

The number of days is set in the "Days after unpaid due date" existing parameter, which is a generic parameter, set at portfolio or contract level.

HINT Use the low-code capabilities of **Innovation Studio** to create a new business status that reflects a particular business need regarding your policy administration flows. In order to do so, please follow the instructions from the [Business Workflows Processor Guide](#).

Mid Term Adjustments Journeys

Here are the journeys and scripts used for the **Mid Term Adjustments** flow:

Digital Journey

The following journey is available for **Mid Term Adjustments**:

MTA_ChangePolicyRequestSearch (MTA - Search)

This custom driven flow helps you to search and find a policy in order to start a new MTA (Mid-Term Adjustment) request. On Step 1 of the form driven flow, you can find the following actions:

- **Search** button (#btnSearch) on click: This function gathers all the data added on the search fields and performs a search based on it with the help of the FTOS_INSPA_ChangePolicyRequestSearchMTA server automation script.
- **Reset** button (#btnReset) on click: This function reset/empty all the fields from the form.
- **Choose Option** button (.btn-option) on click: This function calls the FTOS_INSPA_ChangePolicyRequestInsertMTA server automation script with an object as parameter in order to insert a new MTA request.

Object parameters:

- policyId - (string) - The policy ID.
- paymentTypeId - (string) - The payment type ID.
- paymentFrequencyId - (string) - The payment frequency ID.

The following function is used:

`generateCutomGrid();` - This function generates the table for the search results from the #btnSearch function presented above.

Input parameters:

- viewId - (string) - The ID (CSS ID) for the table.
- viewDataSource - (Array) - An array with the results from the #btnSearch function.
- viewColumns - (variable) - The type of columns for the DxDataGrid. This can have multiple options. For more information about this, please refer to [DevExpress columns configuration](#).

Output parameters:

- N/A

MTA_ChangePolicyRequest

The following functions are available for this journey:

`formScope.hideButtonsAndSections()`: formScope function that hide Buttons and sections based on business conditions;

Input parameters:

- N/A

Output parameters:

- N/A

`formScope.setRequiredAttributes()`: formScope function that sets some attributes required;

Input parameters:

- N/A

Output parameters:

- N/A

`formScope.setChangePolicyAttributesReadOnly()`: formScope function that sets some attributes from the MTA read only;

Input parameters:

- N/A

Output parameters:

- N/A

`formScope.setPolicyAttributesReadOnly()`: formScope
function that sets some virtual attributes from the Policy read only;

Input parameters

- N/A

Output parameters

- N/A

`setOptionSetItems(paramX, resource)`: formScope
function that sets some virtual attributes from the Policy read only;

Input parameters:

- paramX - array with optionsetsItems that we need to display
- resource - an html id

Output parameters:

- N/A

Server Automation Scripts

The following server automation scripts are available for Mid Term Adjustments:

ETOS INSPA
ChangePolicyRequestInsertMTA

This script creates a new change policy request based on the policy ID, the payment frequency ID and the payment type ID. Also, to create a new change policy request, this script runs 2 database queries (Fluent query) to perform the following:

1. The first query search/checks the database to make sure that there is no other change policy request in **Draft** or **Registered** status. If yes, it throws an error with the queried change policy request number.
2. The second query is not used for the moment.

FTOS_INSPA_ChangePolicyRequestSearchMTA

This script performs a search/query to retrieve the policies for the MTA - Search form.

FTOS_INSPA_ChangePolicyRequest_AfterInsert

This script creates records into the FTOS_INSPA_ChangePolicyCoverage entity for the current MTA. The insurance items existing on the policy for which the MTA is inserted are added with a true onPolicy flag, and the remaining items from the product are added with a false onPolicy flag.

FTOS_INSPA_ChangePolicyCoverage_AfterUpdate

This script updates the premium amount for the Change Policy Coverage entries from the Change Policy Request form (MTA_ChangePolicyRequest - page 0/Step 1) and sets the attribute coverageValidated to false. Also, this script uses the `getClientAge()` function which is presented into the MTA_GetPriceHelper server script library. The following functions are used:

```
getPaymentFrequency();
```

This function performs a query to return the new payment frequency for the current change policy request.

Input parameters:

- `cprId` - (string) - The change policy request ID.

Output parameters: an object with the following props:

- `changeReqName.PolicyId` - (string) - Policy ID associated with the queried Change policy request.
- `changeReqName.FrequencyValidated` - (bool) - True or false.
- `payFreqName.Name` - (string) - The name of the current payment frequency.
- `newPayFreqName.Name` - (string) - The name of the new payment frequency.

`insuranceProductItemId();`

This function returns the insurance product item name based on the insurance product item ID.

Input parameters:

- `ipId` - (string) - The insurance product item ID.

Output parameters:

- `ipiName.Name` - (string) - The insurance product item name.

`getInsuranceProductItems();`

This function returns different values based on the Policy ID and Change Policy Coverage ID.

Input parameters:

- id - (string) - Policy id
- cpclId - (string) - Change policy coverage id

Output parameters: an object with the following props:

- FTOS_IP_InsuranceProduct.InsuranceProductCode - (string) - The insurance product code
- FTOS_IP_InsuranceProductItem.Code - (string) - The insurance product item code
- FTOS_IP_InsuranceType.Name - (string) - The insurance type name

`changePolicyPaymentType();`

This function runs for each result from `getInsuranceProductItems();` function and calls the FTOS_IP_PremiumAmountAPI script in order to update the premium amount.

Input parameters:

- mandate - (object) - An object with the following data:
 - insuranceTypeName - (string) - Insurance type name
 - productCode - (string) - Insurance product code
 - insuranceProductItemDetails - (array) - An array with one or more objects>The object/s contains the following props:
 - code - (string) - Insurance product item code
 - calculationDetails - (object) - The object contains the following props:

- coverage - (string) - Insurance product item name
- frequency - (string) - This prop can be:
 - paymentFrequency optionSet name - if the frequencyValidated attribute is null
 - newPaymentFrequency optionSet name - if the frequencyValidated attribute is validated(true)
- age - (number) - The insured client age
- sumInsured - (number) - The insured amount

Output parameters: there can be 3 different outputs:

- Error - If some of the data sent are incomplete or wrong.
- Error - If the authentication details are wrong or missing.
- Success - If the API call is successful, it returns the new premium amount and makes an update on the FTOS_INSPA_ChangePolicyCoverage entity with the new premium amount.

buttonActionsCoverageUpdate

This script receives an ID through context data and a string describing the type of the action desired as follows:

- for type “remove”: it removes the coverage requested through the ID from the new policy coverages.
- for type “add”: it adds the coverage requested through the id to the new policy coverages.
- for type “validateCoverage”: it changes the status of all new policy coverages to **Validated**.

MTA_ButtonActions

This script contains all the actions that are made on buttons on MTA flow, ex: Cancel, decline, accepted, register etc. The following functions are used:

`function getFrequencyUpdatedItems(policyId, ipiId):`
function that gets some informations from the FTOS_INSPA_Policy and FTOS_INSPA_PolicyInsuranceItem entities.

Input parameters:

- policyId- unique identifier of the FTOS_INSPA_Policy entity.
- ipiId - unique identifier of the FTOS_INSPA_PolicyInsuranceItem entity.

Output parameters:

- fluentQuery -> an object with the results.

`getProductItemsForFrequency(newPolId):` function that gets the Items from which the frequency changes.

Input parameters:

- newPolId - unique identifier of FTOS_INSPA_Policy entity, but the new version of it.

Output parameters:

- query - an object with the results.

`queryRegisterCoverage(entityId):` function that gets the registered coverages from the FTOS_INSPA_ChangePolicyRequest entity.

Input parameters:

- entityId - unique identifier of FTOS_INSPA_ChangePolicyRequest entity.

Output parameters:

- query - an object with the results.

getInfoForAddProductItem(policyId): function that gets informations from more entities, function that is necessary when adding the items on the new version of policy.

Input parameters:

- policyId - unique identifier of FTOS_INSPA_Policy entity.

Output parameters:

- query - an object with the results.

addProductItemMTA(insertValues): the function gets all the necessary details of the product item ready to be added on the policy. After manipulating some data received on a policy generation, the function inserts a new product item in the FTOS_INSPA_PolicyInsuranceItem entity.

Input parameters:

- insertValues - object containing all the required details for finding and adding a new product item on the new policy.

Output parameters:

- N/A.

getInstallmentsOnPolicy(policyId): function that gets the installments from a specified Policy.

Input parameters:

- policyId - unique identifier of the FTOS_INSPA_Policy entity.

Output parameters:

- ids - array with unique identifiers of FTOS_INSQB_PaymentScheduleDetail entity.

`updateInstallmentsOnPolicy(newPolId, updatedPolicyPremiumAmount)`: function that updates the installmentAmount on the FTOS_INSQB_PaymentScheduleDetail entity.

Input parameters:

- newPolId - unique identifier of the FTOS_INSPA_Policy entity, but the new version of it.
- updatedPolicyPremiumAmount - the calculated premium amount.

Output parameters:

- N/A.

`getInitialPremiumAmount(policyId, insuranceItemId)`: function that gets the premium amount from the FTOS_INSPA_Policy entity.

Input parameters:

- policyId - unique identifier of the FTOS_INSPA_Policy entity.
- insuranceItemId - unique identifier of FTOS_INSPA_PolicyInsuranceItem entity.

Output parameters:

- initialPremium - the premium.

MTA_FillCustomVirtualAttributes

Server script that fills all the custom virtual attributes on MTA process. The following function is used:

`function getPolicyInfo(policyId):` function that gets the informations from the FTOS_INSPA_Policy entity.

Input parameters:

- policyId - unique identifier of FTOS_INSPA_Policy entity.

Output parameters:

- fetch - object with the results.

FTOS_INSPA ChangePolicyRequestAddSequencerMTA

This script sets a new number for a new change policy request based on the PolicyChangeRequest sequencer.

FTOS_MTA_GetUpdatedPolicy

This script is used to fetch/render the new version of the policy into MTA_ChangePolicyRequest/Section Step2/afterGenerate.js.

Server Automation Script Library

The following libraries are available for Mid Term Adjustments.

MTA_GetPriceHelper

This library contains different functions that help calculate the new premium amount.

`getClientAge()`; this function returns the insured client age based on the Change Policy Request ID.

Input parameters:

- `cprId` - (string) - Change policy request ID.

Output parameters:

- `accountAge` - (number) - The insured client age or 0 if the client does not have a DOB set in the db.

`getPaymentFrequency()`; this function returns the payment frequency name based on the payment frequency ID.

Input parameters:

- `payFreqId` - (string) - Payment frequency ID.

Output parameters:

- Payment frequency name or null if the ID does not exists.

`getInsuranceProductItems()`; this function returns different attributes from the FTOS_INSPA_ChangePolicyCoverage entity.

Input parameters:

- `cprId` - (string) - Change policy request ID.

Output parameters: an object with the following data:

- `cpcData.FTOSINSPChangePolicyCoverageId` - (string) - Change policy coverage ID.
- `cpcData.CoverageTypeId` - (string) - Coverage type ID.

- `cpcData.InsuranceProductItemId` - (string) - Insurance product item iID.
- `cpcData.InsuredAmount` - (numeric) - Insured amount.

`getUpdatedPrices()`; this function updates/recalculate the premium amount based on an object.

Input parameters:

- `queriedObject` - (object) - This object contains the following props:
 - `insuranceTypeName` - (string) - Insurance type name.
 - `productCode` - (string) - Insurance product code.
 - `insuranceProductItemDetails` - (array) - An array with one or more objects. The object/s contains the following props:
 - `code` - (string) - Insurance product item code.
 - `calculationDetails` - (object) - The object contains the following props:
 - `coverage` - (string) - Insurance product item name.
 - `frequency` - (string) - This prop can be:
 - `paymentFrequency` optionSet name - if the `frequencyValidated` attribute is null.
 - `newPaymentFrequency` optionSet name - if the `frequencyValidated` attribute is validated (true).
 - `age` - (number) - The insured client age.

- `sumInsured` - (number) - The insured amount.

Output parameters: there can be 3 outputs:

- Error - If some of the data sent are incomplete or wrong.
- Error - If the authentication details are wrong or missing.
- Success - If the API call is successful, returns the new premium amount.

FTOS_INSPA_MTA

Library that contains different functions that helps on MTA process.

`function getAlterationTypes(policyId)`: function that gets the alteration types for a policy.

Input parameters:

- `policyId`- unique identifier of the `FTOS_INSPA_Policy` entity.

Output parameters:

- `fetch`- object with the results.

`function getPolicyInsuranceItems(policyId)`: function that gets the insurance items for a policy.

Input parameters:

- `policyId` - unique identifier of the `FTOS_INSPA_Policy` entity.

Output parameters:

- `query`- object with the results.

Policy Cancellation Journeys

Here are the journeys, entities, libraries and endpoints related to the **Policy Cancellation** functionality:

Data Model

- Entity `FTOS_INSP_PolicySurrender` with all attributes;
- Entity `FTOS_INSP_Payment` with all attributes. This is a sub-entity of the `FTOS_PYMT_Payment` entity, which stores payment data from other **FintechOS** insurance solutions as well.

1 FTOS_INSP_PolicySurrender

FTOS_INSP_PolicySurrenderSearch Journey

This is an user journey aimed at implementing the **Search Policy** functionality. This journey is connected to the `FTOS_INSP_PolicySurrender` entity.

General Description

This journey is marked as **Default** for insert on `FTOS_INSP_PolicySurrender` entity.

The journey uses the following methods from **FintechOS Client Side SDK**:

After identifying the policy, when pressing **Choose option**, the record is saved using `ebs.saveEditForm` method. The flow is redirected to the **Default for Edit** form driven flow of `FTOS_INSP_`

PolicySurrender entity, using `ebs.goToUrl` method. The results grid is generated using `ebs.generateGrid` method from **FintechOS** Client Side SDK.

Client Side Libraries

From the FTOS_INS_Utils library, the following function is used:

Function `isNullOrEmpty()` - The function checks if a value is null or empty.

Input parameters: `value` - The parameter that the user needs to check.

Output parameters: `Bool` - The parameter returned as a Boolean value (true or false) to describe whether the input value is null or not.

Endpoints

The following endpoints are used for gaining access to policy data:

FTOS_INSP_PolicySurrenderSearch

This endpoint searches for Policies that are eligible for the Cancellation process.

Function `getFetch()` searches for Policies with the following statuses: **InForce** or **Issued** and also for the attributes that are filled in by the user on the **Search Policy** view - for example Policy No, First Name, Last Name, Pin, Email etc.

Input parameters: N/A.

Output parameters: N/A.

FTOS_INSP_PolicySurrenderFetchPolicy

This endpoint fetches all necessary Policy related information in order to display it on the Cancellation entity.

Function `getFetch(policyId)` checks whether the Policy is already subject to a cancellation flow - in its final steps, possibly initiated by some other user.

- **Input** parameters: `policyId` - This is the unique identifier of a record from **FTOS_INSPA_Policy** entity.
- **Output** parameters: returns an array of the Cancellations that are in status **Rejected** or **Cancelled** on the specified Policy.

Function `getLastPaymentDate(policyId)` returns the list of **Installments** on the specified Policy, ordered by **dueDate** ascendant.

- **Input** parameters: `policyId` - This is the unique identifier of a record from **FTOS_INSPA_Policy** entity.
- **Output** parameters: N/A.

FTOS_INSP_GetPolicy

This endpoint retrieves the necessary information, from the chosen Policy, that corresponds with the business conditions mandatory for executing the Cancellation process.

FTOS_INSP_PolicySurrenderUpdate Journey

This is an user journey aimed at implementing the updates occurring during the **Policy Cancellation** flow. This journey is connected to the **FTOS_INSP_PolicySurrender** entity.

General Description

This journey is marked as **Default** for insert on FTOS_INSP_PolicySurrender entity. The journey uses the following functions:

Function	Input parameters	Output parameters
<code>initializedValues()</code> Sets all the attributes on required level.	N/A.	N/A.
<code>setPaymentBeneficiaryData()</code> Fills the Payment Beneficiary section with information based on the payment beneficiary Type selected by user.	N/A.	N/A.
<code>makeFieldsReadonly()</code> Makes all the fields Read Only.	N/A.	N/A.
<code>updateSurrenderData()</code> Updates specific data of the Cancellation process.	<code>changedFieldName</code> - The name of the field that is changed.	N/A.
<code>showHideByStatus()</code> Shows/ hides tabs based on the Cancellation status.	N/A.	N/A.
<code>setPremiumReturnedTabData()</code> Fills the Premium Returned tab with data.	N/A.	N/A.

Client Side Libraries

From the FTOS_INS_Utils library, the following function is used:

Function: `isNullOrEmpty()` - The function checks if a value is null or empty.

Input parameters: `value` - The parameter that the user needs to check.

Output parameters: `Bool` - The parameter returned as a Boolean value (true or false) to describe whether the input value is null or not.

Endpoints

The following endpoints are used for gaining access to Policy data.

FTOS_INSP_PolicySurrender_ BankIdentification

This endpoint matches an IBAN account to a certain Bank. It calls an on demand server side script named `FTOS_INSP_PolicySurrender_BankIdentification`.

Input parameters: `iban`- IBAN value, filled in by the user.

Output parameters: `FTOS_PYMT_Bank record id` - The unique identifier of a record from `FTOS_PYMT_Bank` entity.

FTOS_INSP PolicySurrenderClaimUsed

This endpoint checks whether the Policy is subject to a Claim flow or not. If true, the endpoint flags the Policy to inform that a Claim flow is used on the environment. The endpoint calls an on demand server side script named `FTOS_INSP_PolicySurrenderClaimUsed`.

FTOS_DFP FlowProcessorSettingsByType

This endpoint extracts data from `FTOS_DFP_FlowSettings` entity. It calls an on demand server side script named `FTOS_DFP_FlowProcessorSettingsByType`.

Input parameters:

- `flowSettingsName` - The name of the flow setting.
- `processorSettingsType` - The type of the processor setting.
- `processorSettingsName` - The name of the processor setting.

Output parameters: N/A.

FTOS_INSP PolicySurrenderStatusChange

This endpoint changes the status of the policy record, after applying validation. It calls an on demand server side script named `FTOS_INSP_PolicySurrenderStatusChange`.

Input parameters:

- `surrenderId` - Id of the cancellation record, from `FTOS_INSP_PolicySurrender` entity.
- `newStatus` - Displays name of the new business status.
- `returnedMoney` - Amount to be paid back to the policy holder.
- `reasonType` - The reason type for the policy cancellation.
- `policyId` - Id of the policy that is subject to cancellation.
- `finalEndDate` - Final date for policy termination.
- `premiumAmount` - The Premium for that Policy.

- **areClaimsOnPolicy** – checks whether the cancellation is pending due to claims opened on the Policy.
- **resolutionReason** - the resolution regarding the Policy cancellation.

Output parameters: N/A.

FTOS_INSP PolicySurrenderInsertInPayment

This endpoint makes inserts into the FTOS_PYMT_Payment and FTOS_INSP_Payment entities, based on certain conditions. Also, this endpoint changes the FTOS_INSPA_Policy status in accordance with the **reasonType** picked by the user during the Cancellation process. The endpoint calls an on demand server side script named **FTOS_INSP_PolicySurrenderInsertInPayment**.

Input parameters:

- **surrenderId** – Id of the cancellation record, from **FTOS_INSP_PolicySurrender** entity.
- **newStatus** – Displays the name of the new business status.

Output parameters: N/A.

FTOS_INSP PolicySurrenderUpdateUserJourney

This endpoint makes the necessary updates on a FTOS_INSP_PolicySurrender record, when Surrender Notification Date, Reason Type, Requested End Date or Final End Date fields values are changed. It calls an on demand server side script named **FTOS_INSP_**

PolicySurrenderUpdateUserJourney.**Input** parameters:

- **reasonType** – Id of the selected reason type.
- **changedFieldName** – Name of the attribute that changed its status.

Output parameters: N/A.

FTOS_INSP_GetAccountData

This endpoint retrieves data from the **Account** entity, about the beneficiary of the payment in order to populate the following virtual attributes Payment Beneficiary **Last Name**, Payment Beneficiary **First Name**, Payment Beneficiary **PIN**. The endpoint calls an on demand server side script named **FTOS_INSP_GetAccountData**.

Input parameter: **accountId** – The Payment Beneficiary id attribute values.

Output parameters: data object from the Account table, with the following keys **Email**, **FirstName**, **FullName**, **LastName**, **MobilePhone**, **UniqueID**, **a_Accountid**, **clientCode**.

FTOS_INSP PolicySurrenderDeleteSurrenders

This endpoint deletes Cancellation processes after X days if their status is still **In Progress**. It changes the business status to **Cancelled**.

Input parameters: N/A.**Output** parameters: N/A.

FTOS_INSP PolicySurrenderUpdateResolutionReason

This endpoint updates the Resolution Reason attribute according to the values introduced for the custom virtual attributes. It uses the FTOS_INSP_PolicySurrender library.

Input parameters:

- **id** - The unique identifier of the record.
- **resolutionReasonV1** - The resolution reason selected by the user on the **Change Request** step (by using the **FTOS_INSP_PolicySurrenderUpdate** form).
- **resolutionReasonV2** - The resolution reason selected by the user on the **Premium Returned** step (by using the **FTOS_INSP_PolicySurrenderUpdate** form).
- **resolutionReasonV3** - The resolution reason selected by the user on the **Request Approval** step (by using the **FTOS_INSP_PolicySurrenderUpdate** form).

Output parameters: N/A.

Business Workflow Configuration Actions

Before: from **Draft** to **In Progress** – this script verifies that the **reasonType** attribute is filled in.

After: from **In Approval** to **Approved** – this script verifies that the **finalEndDate** and **premiumAmount** attributes are updated on the FTOS_INSPA_Policy entity, based on the reason type selected on the FTOS_INSP_PolicySurrender entity.

Server Side Script Libraries

Here are the libraries for the functions performed on the server:

FTOS_DFP_FlowProcessorSettings

From this library, the following function is used:

`getFlowProcessorSettingsByType`.

FTOS_GetAccountData

From this library, the following functions are used:

Function	Input parameters	Output parameters
<code>findObjectByKey()</code> Finds an object by keys and values.	<code>array</code> - An array with the object . <code>key</code> - The key used for search (string). <code>value</code> - The value used for search.	<code>array</code> - An array with all the resulting elements.
<code>getIntegrationByCode()</code> Gets the FTOS_IntegrationProcessId by the <code>code</code> parameter.	<code>code</code> - The code attribute from FTOS_IntegrationProcess .	<code>rez</code> - The result of the query made in the function.
<code>getPolicyAccounts()</code> Gets information about all the Accounts from the Policy entity.	<code>policyId</code> - The uniqueidentifier of a record from the Policy entity.	<code>policyData</code> - An array with all the information detected.
<code>getAccountDataById()</code> Gets the id information for the Accounts.	<code>accountId</code> - The unique identifier from the Account entity.	<code>rez</code> - The result of the query made in the function.

Function	Input parameters	Output parameters
getAccountDataByCode() Performs similar action as above but the subject key for the search is the code.	clientId - The unique identifier from clientId attribute in Account entity.	rez - The result of the query made in the function.
insertUpdateClient() Checks if the account id exists already, if not a new registration is made.	accountId - The unique identifier from the Account entity. input - An object with data for the insert action.	accountId and newAccountId

FTOS_INSP_PolicySurrender

From this library, the following functions are used:

Function	Input parameters	Output parameters
treatAsUTC() Formats a date to UTC time zone.	date - The date that needs to be formatted.	result - The date in UTC time zone.
dateDiffInDays() Calculates the difference (in days) between 2 dates.	d1 - Date 1. d2 - Date 2.	dateDiffInDays - The result of the difference.
monthDiff() Calculates the difference (in months) between 2 dates.	d1 - Date 1. d2 - Date 2.	months - The result of the difference.
getSurrenderDetails() Gets all the information from a cancellation based on the entity id.	entityId - The cancellation entityId, unique identifier of the FTOS_INSP_PolicySurrender entity.	queryResult - The result of the query made in the function.

Function	Input parameters	Output parameters
getUPRContractual() Gets the UPR (unearned premium) and contractual values.	N/A.	uprDays - Parameter from FTOS_DFP_ProcessorSettings entity. contractualDays - Parameter from FTOS_DFP_ProcessorSettings entity.
setChangeDateValue() Updates the FTOS_INSP_PolicyChange entity with information about the changed values.	dateValue - The changeDate value. surrenderDetails - An array with details from the Cancellation entity.	N/A.
changeDateRules() Calls also the setChangeDateValue (dateValue, surrenderDetails) function and makes updates.	surrenderDetails - An array with details from Cancellation entity. contractualDays - Parameter from FTOS_DFP_ProcessorSettings entity.	N/A.

Function	Input parameters	Output parameters
withdrawBehavior() Sets the withdrawal reason type to Withdrawal .	surrenderDetails - An array with details from Cancellation entity. updateObj - An object with updates. contractualDays - Parameter from FTOS_DFP_ProcessorSettings entity.	updateObj - An object with updates.
otherReasonBehavior() Sets the withdrawal reason type to Other Reason .	surrenderDetails - An array with details from Cancellation entity. updateObj - An object with updates. contractualDays - Parameter from FTOS_DFP_ProcessorSettings entity.	updateObj - An object with updates.
companySurrenderBehavior() Sets the withdrawal reason type to Company Surrender .	surrenderDetails - An array with details from Cancellation entity. updateObj - An object with updates. contractualDays - Parameter from FTOS_DFP_ProcessorSettings entity.	updateObj - An object with updates.

Function	Input parameters	Output parameters
changeDateRulesRequestedEndDateChanged() During the Cancellation flow, when the reason type attribute is changed other attributes may be affected, for example the change date . This function sets the rules for change date attribute due to reason type changes.	surrenderDetails - An array with details from Cancellation entity.	N/A.
onChangeLogic() Sets the logic from changes made in Cancellation flow.	changedFieldName - The name of the attribute that is changed.	N/A.
updateSurrender() Updates the cancellation flow.	entityId - The unique identifier from FTOS_INSP_PolicySurrender entity. reasonType - The selected the reason type.	N/A.

Function	Input parameters	Output parameters
updateResolutionReason() Updates the resolution reason attribute with the values from the custom virtual attributes used on different steps of the Cancellation flow.	id - The unique identifier of the record. resolutionReasonV1 - The resolution reason selected by the user on the Change Request step. resolutionReasonV2 - The resolution reason selected by the user on the Premium Returned step. resolutionReasonV3 - The resolution reason selected by the user on the Request Approval step.	N/A.

Scheduled Jobs

FTOS_INSP_PolicySurrender runs daily at 12:00 PM and uses the **FTOS_INSP_PolicySurrenderDeleteSurrenders** endpoint.

Style Sheets

FTOS_CLAIM_MainStyle

2 FTOS_INSP_Payment

General Description

This is an user journey aimed at storing the data about payments made through the **Policy Cancellation** process. This journey is marked as **Default** (for insert) on FTOS_INSP_Payment entity. The journey uses the following functions:

Function	Input parameters	Output parameters
<code>approvedHide()</code> Makes the selected attributes hidden.	N/A	N/A
<code>showPaymentData()</code> Shows the payment data area.	N/A	N/A
<code>fillPolicyDetails()</code> Gets data from the Policy entity based on policyId.	<code>policyId</code> - The unique identifier of a record from FTOS_INSPA_Policy entity.	N/A
<code>setPaymentBeneficiaryData()</code> Fills the Payment Beneficiary section with payment information based on the payment beneficiary Type selected by user.	N/A	N/A
<code>formScope.setPaymentBeneficiaryReadOnly()</code> Makes the selected attributes Read Only.	N/A	N/A
<code>formScope.showDefaultValuesPaymentBeneficiary()</code> Populates the Payment Beneficiary section with default details, based on the payment beneficiary Type .	<code>paymentBeneficiary</code> - The beneficiary of the payment. <code>policyId</code> - The unique identifier of a record from FTOS_INSPA_Policy entity.	N/A

Client Side Libraries

- From the **FTOS_GLUsed** library, the following function is used:
Function `showGL()` - The function shows the GL (General Ledger) tab.
Input parameters: `step` - The step where the GL tab is available.
Output parameters: N/A.
- From the **FTOS_INS_Utils** library, the following function is used:
Function `setAttributesReadonly()` - The function sets the input attributes to Read Only.
Input parameters: `attribute list` - The list of attributes.
Output parameters: N/A.

Endpoints

The following endpoints are used for gaining access to Policy data:

`FTOS_INSP_validatePaymentReturn` - Validates if a Payment is ready to be returned.

`FTOS_PYMT_Payment_ReturnUnallocatedAmount` - This endpoint uses the `FTOS_PYMT_PaymentReturn` library to return an Unallocated Amount on a Policy.

Server Side Script Libraries

From the `FTOS_PYMT_PaymentReturn` library, the following functions are used, in the return flow:

Function	Input parameters	Output parameters
<code>generateReturn()</code>	<code>policyId</code> - The unique identifier of a record from FTOS_INSPA_Policy entity. <code>type</code> - The type of the payment. <code>paymentTypeName</code> - The name of the payment type.	<code>newPaymentId</code> - The unique identifier of the new payment insertion.

Function	Input parameters	Output parameters
getPaymentsToReturnByPolicy() Retrieves data about all unsettled payments, with status different than Paid or Closed .	policyId - The unique identifier of a record from FTOS_INSPA_Policy entity.	result - An array with the results.
getPaymentDetails() Gets all the details from a specified payment.	paymentId - The unique identifier of a record from FTOS_INSP_Payment entity.	payment - An array with the payment details.
getPaymentTypeIdByName() Gets the payment type id by searching for the payment type name.	paymentTypeName - The name of the payment type.	paymentTypeId - The unique identifier of a record from the Payment Type optionset.
insertPaymentReturn() Inserts the payment return.	policyId - The unique identifier of a record from FTOS_INSPA_Policy entity. paymentDetails - The details of the payment. returnPaymentAmount - The amount of the payment. returnCurrencyId - The id of the currency. type - The type of the payment. sourcePaymentId - The id of the payment source. paymentTypeName - The name of the payment type.	result - An array that contains the insertPaymentPymt and insertInPayment IDs.
getBankId() Gets the FTOS_PYMT_Bank id.	bankCode - The code of the bank.	bankId - The unique identifier of a record from FTOS_PYMT_Bank entity.

Function	Input parameters	Output parameters
getAllPaymentsDetails() Gets specific details about the payment.	returnDate - The date of the return. paymentId - The unique identifier of a record from FTOS_INSP_Payment entity.	payments - An array with the details about the payment.
returnUnallocatedAmount() Returns the unallocated amount.	paymentId - The unique identifier of a record from FTOS_INSP_Payment entity. paymentTypeName - The name of the payment type	insertPaymentReturnResult - An object with keys from FTOS_INSP_Payment .

Scheduled Jobs

The **FTOS_PYMT_Payment_ReturnUnallocatedAmount** job uses **FTOS_PYMT_Payment_ReturnUnallocatedAmount** endpoint.

Style Sheets

FTOS_CLAIM_MainStyle

Other Automated Journeys

FTOS_INSPA_Policy_IssuedToEnforced

The purpose of the job is to make the transition of a specific policy to the **InForce** status automatically. It runs on a daily basis at 01:00 AM and it calls the **FTOS_INSPA_Policy_IssuedToEnforced** endpoint.

Endpoint

The `FTOS_INSPA_Policy_IssuedToEnforced` endpoint excludes from the automatic flow the policies that do not have **Automatic In Force** bool attribute set to true at the product level.

Server Side Scripts Library

From the `FTOS_INS_PolicyAdmin` library, the following function is used:

Function `jobScript`- The function selects all the policies that are in the **Issued** business status, have the same begin date as the current date and have the product with the **Automatic InForce** checkbox selected. On the policies that have been found, the script goes ahead and changes the business status to **InForce**.

Input parameters: N/A

Output parameters: N/A

FTOS_PA_PolicyWithdraw

The purpose of the job is to make the transition of a specific policy from the Proposal to the Withdraw Status automatically. It runs on a daily basis at 02:00 AM and it calls the `FTOS_PA_PolicyWithdraw` endpoint.

Endpoint

The `FTOS_PA_PolicyWithdraw` endpoint transitions the policy from the **Proposal** to the **Withdraw** business status, after a number of days since the first installment moved to **Unpaid** status. The number of days is set up in the `PWDAY` withdraw parameter.

Server Side Scripts Library

From the **FTOS_INS_PolicyAdmin** library, the following functions are used:

Function `withdrawPolicies` - The function selects all the policies that are in the **Proposal** business status and have an installment that moved to the **Unpaid** status with a set number of days before the current date. On the policies that have been found, the script goes ahead and changes the business status to **Withdraw**.

- **Input** parameters: N/A
- **Output** parameters: N/A

Function `convertDateFromInvariant` - The function converts an **Invariant** date to a string format by adding the time next to it and then it makes a `newDate` object with it. On the policies that have been found, the script goes ahead and changes the business status to **Withdraw**.

- **Input** parameters: `date` - The invariant date.
- **Output** parameters: `dateObj` - The new date object.

Function `getInterval` - first imports the library required for getting the parameter's value, from `FTOS_PA_FlowParameter` and then by calling the method `(getFlowParameterValueByCode)` it obtains the specific value of the parameter. The function goes on and calculates two interval ends necessary for checking if any installment has transitioned to the **Unpaid** status. `minDate` is calculated by subtracting the parameter's value from the current date and `maxDate` is calculated by adding a day to the `minDate`.

- **Input** parameters: N/A
- **Output** parameters: N/A

Policy Generation API

The **Policy Generation API** is responsible for generating generic policies into the core system. Requests for policy generation can be received from various external systems. Once the integration with an external system is achieved, some information should be received from the external system in order for the action to be successful - namely, policies are generated into the core system according to the information received from the external system.

Below you can find examples about generating a new policy by an insurance [broker](#) and by an insurance [agent](#).

Example Home Policy - Broker

A Broker registers a new generic Policy into the system.

```
1  var p = {
2      policyList: [
3          {
4              insuranceTypeName: "Home",
5              productCode: "BA_EMB",
6              insuranceProductItemList: [{
7                  code: "HHR0",
8                  insuredAmount: 4000.0,
9                  finalPremiumAmount: 64.25
10             }, {
11                 code: "HA",
12                 insuredAmount: 4000.0,
13                 finalPremiumAmount: 64.53
14             }],
15             issuedDate: "2021-07-16",
16             startDate: "2021-07-17",
17             renewedPolicyNo: null,
18             quoteNo: "John21",
19             totalIndemnityLimit: 10000,
20             validityType: "Months",
21             validity: 12,
22             agent: {
23                 agentId: null,
```

```

24         type: null
25     },
26     broker: {
27         brokerId: "Broker1"
28     },
29     contractor: {
30         uniqueIdentifier: "00000000000000",
31         type: "Individual person",
32         firstName: "John",
33         lastName: "Smith",
34         email: "john.smith@gmail.com",
35         phone: "0720202020"
36     },
37     insured: {
38         uniqueIdentifier: "00000000000000",
39         type: "Individual person",
40         firstName: "John",
41         lastName: "Smith",
42         email: "john.smith@gmail.com",
43         phone: "0720202020"
44     },
45     beneficiary: {
46         uniqueIdentifier: "00000000000000",
47         type: "Individual person",
48         firstName: "John",
49         lastName: "Smith",
50         email: "john.smith@gmail.com",
51         phone: "0720202020"
52     },
53     currency: "EUR",
54     paymentType: "BrokerCollection",
55     paymentFrequency: "monthly",
56     mentions: "Insert comment here"
57 },
58 ],
59 };
60 ebs.callActionByNameAsync("PolicyGenerationAPI", p)
61 .then(
62     function(e){
63         console.log(e.UIResult.Data)
64     }
65 );

```

Example Home Policy - Agent

An Agent registers a new generic Policy into the system.

```

1  var p = {
2      policyList: [
3          {
4              insuranceTypeName: "Home",
5              productCode: "BA_EMB",
6              insuranceProductItemList: [{
7                  code: "HHR0",
8                  insuredAmount: 4000.0,
9                  finalPremiumAmount: 64.25
10             }, {
11                 code: "HA",
12                 insuredAmount: 4000.0,
13                 finalPremiumAmount: 64.53
14             }],
15             issuedDate: "2021-07-16",
16             startDate: "2021-07-17",
17             renewedPolicyNo: null,
18             quoteNo: "John21",
19             totalIndemnityLimit: 10000,
20             validityType: "Months",
21             validity: 12,
22             agent: {
23                 agentId: "23435788",
24                 type: "Individual person"
25             },
26             broker: {
27                 brokerId: null
28             },
29             contractor: {
30                 uniqueIdentifier: "0000000000000",
31                 type: "Individual person",
32                 firstName: "John",
33                 lastName: "Smith",
34                 email: "john.smith@gmail.com",
35                 phone: "0720202020"
36             },
37             insured: {
38                 uniqueIdentifier: "0000000000000",
39                 type: "Individual person",
40                 firstName: "John",
41                 lastName: "Smith",
42                 email: "john.smith@gmail.com",
43                 phone: "0720202020"

```



```

44         },
45         beneficiary: {
46             uniqueIdentifier: "000000000000",
47             type: "Individual person",
48             firstName: "John",
49             lastName: "Smith",
50             email: "john.smith@gmail.com",
51             phone: "0720202020"
52         },
53         currency: "EUR",
54         paymentType: "BrokerCollection",
55         paymentFrequency: "monthly",
56         mentions: "Insert comment here",
57         renewType: "Manual"
58     },
59 ],
60 };
61 ebs.callActionByNameAsync("PolicyGenerationAPI", p)
62 .then(
63     function(e){
64         console.log(e.UIResult.Data)
65     }
66 );

```

Request Data Parameters

Here is the list of data parameters included in the request:

Parameter	Description
insuredObject	Object containing keys to describe the insured object.
address	Object describing address of the insured object.
apartmentNo	Apartment number.
buildingNo	Building number.
city	City name of a record from City entity or null. City name value must exist in the city nomenclature.
subcity	Subcity name.

Parameter	Description
districtCode	District code of a record from District entity or null. District code must exist in the district nomenclature.
entrance	Entrance.
floorNo	Floor.
postalCode	Postal code.
street	Street name.
streetNo	Street number.
streetType	Street type – one of the values existing in StreetType entity
policyList	List of policies to generate; Multiple policies can be generated at once
insuranceTypeName	The name of an insurance type configured in the system, in FTOS_IP_InsuranceType entity.
productCode	Insurance product code.
insuranceProductItemList	List of items to be included on the policy.
code	Item code.
insuredAmount	Item insured amount.
finalPremiumAmount	Item premium amount.
excessType	Deductibles for coverages.
excessValue	Value of deductibles.
issuedDate	Date of issuance, basic format ISO 8601 YYYY-MM-DD.
startDate	Policy begin date, basic format ISO 8601 YYYY-MM-DD.
renewedPolicyNo	Old policy number, in case of renewal.
quoteNo	Quote number.
totalIndemnityLimit	Total indemnity limit on the policy.
validityType	Type of validity. Months value supported in the first version of the API.

Parameter	Description
validity	How many years/months/days we want this policy to be valid for. 12 value supported in the first version of the API.
agent	Object containing the agent details.
agentId	Agent Id.
type	Type of the agent issuing the policy ("Individual person" or "Legal person").
broker	Object containing broker details.
brokerId	Broker ID.
contractor	Area containing contractor details.
uniqueIdentifier	CNP or CUI, depending on the contractor's type.
type	Contractor type ("Individual person" or "Legal person").
firstName	Contractor first name.
lastName	Contractor last name.
email	Contractor email.
phone	Contractor mobile phone.
insured	Area containing insured details.
uniqueIdentifier	CNP or CUI, depending on the insured type.
type	Insured type ("Individual person" or "Legal person").
firstName	Insured first name.
lastName	Insured last name.
email	Insured email.
phone	Insured mobile phone.
beneficiary	Area containing beneficiary details.
uniqueIdentifier	CNP or CUI, depending on the beneficiary type.
type	Beneficiary type ("Individual person" or "Legal person").

Parameter	Description
firstName	Beneficiary first name.
lastName	Beneficiary last name.
email	Beneficiary email.
phone	Beneficiary mobile phone.
currency	Currency.
paymentType	Payment type. Values: <ul style="list-style-type: none"> • OP for bank transfers • PayU for PayU • PayU-on time for PayOnTime • brokerCollection for Broker Collection.
paymentFrequency	Payment Frequency. Values: <ul style="list-style-type: none"> • full for Full(entire payment at once) • annually for Annually • semiAnnually for Semi-Annually • quarterly for Quarterly • monthly for Monthly.
mentions	Special mentions at Policy level.
renewType:	The tipe of the renewal. It can be manual or automatic etc.

Response

This is an example of a response:

```

1  {
2      "errorMessage": null,
3      "errorCode": null,
4      "isSuccess": true,
5      "result": {
6          "policyData": [{
7              "policyBeginDate": "2021-07-17",
8              "policyEndDate": "2022-07-16",
9              "policyId": "8707f5a4-4fae-40b2-b899-
10             e27735c5b98e",
11              "policyNumber": "80000480",
12          }]
13     }

```

Response description:

Key	Description
Error code	Error code.
Error message	Error message.
isSuccess	Marks if the request was successful or not.
result	Array of objects containing details about the identified policies.
policyData	Array of details for each generated policy.
policyBeginDate	Policy Start Date.
policyEndDate	Policy End Date.
policyId	GUID identifying the policy inside the Core Insurance Master system.
policyNumber	Policy Number.

Error Messages

The following are the error messages that can be encountered during policy generation process:

Code	Text	Description
ERR.PA.50101	Broker data cannot be identified!	Broker does not exist

Code	Text	Description
ERR.PA.50102	Invalid issued date!	Issue date value provided on issuedDate key is less than current date
ERR.PA.50103	Invalid start date!	Start date value provided on startDate key is less than or equal with issuedDate value
ERR.PA.50104	Invalid paymentType!	Value provided on paymentType key is not valid (not part of the accepted values)
ERR.PA.50105	Invalid currency!	Currency code provided on currency key is not identified
ERR.PA.50106	Invalid Renew Type!	Renew type invalid
ERR.PA.50107	Quote Number missing!	Quote number is mandatory when creating new policies (but not for renewal)
ERR.PA.50108	Existing policy for the same quote and insurance type!	Another policy of the same insurance type is already registered

Endpoints

PolicyGenerationAPI

The endpoint is responsible for generating generic policies in the Core system. When generating a policy, some information should be sent in order for the action to be successful. This endpoint also aligns the policies payment schedule with the one from the associated Master Policy.

In the moment a new policy is added on the Master Policy (this happens when the policy is generated, meaning the policy is in the **Proposal** status), the payment schedule for that policy is set up taking into consideration the unpaid installments of the associated Master Policy. More precisely, the system verifies the Master Policy payment schedule and retrieves the number of unpaid installments of that Master Policy. The resulting number is the number of the installments for that newly added policy.

Example:

Say the **Master Policy Payment Frequency** is set as Quarterly.

In the fourth month from the **Begin Date** of the Master Policy, a new policy is added, where the **End Date** of the policy is the same as the **End Date** of the Master Policy. In this moment, on the Master Policy, the system checks the number of unpaid installments for that Master Policy and finds that there are 2 unpaid installments.

1. The new policy is generated with 2 installments of the newly issued policy. These have the same due date as the 2 unpaid installments of the Master Policy.
2. The total premium amount of the policy is divided into 2.
3. The system updated the following elements of the Master Policy:
 - The **Payment Schedule** is updated by adding the 2 installment sums of the policy to those 2 unpaid installment sums of the Master Policy;
 - The **Payment Amount** is updated by adding to the old premium amount of the Master Policy the newly added policy premium amount;
 - The **IPT Amount** is updated by adding to the old IPT amount of the Master Policy the newly added policy IPT amount.

Server Side Script Libraries

PolicyAdminAPIs - This library contains all the necessary functions for manipulating, updating or inserting data whenever a policy generation endpoint is being called. It is split in three objects for

grouping the methods based on what type of process they cover.

Function	Input parameters	Output parameters
getItemCoveredRisk (insuranceProductItemId) Function returns all the covered risks that have been configured for an insurance product item.	insuranceProductItemId - the id of each insurance product item found on the requested insurance product.	cr - list with objects for all the covered risks that have been found for the insurance product item.
getInsuranceProductItemModuleList (insuranceProductItemId) Function returns all the insurance product items that are part of a specific module - the fetch inside the function selects all the product items that have the same item parent id.	insuranceProductItemId - the id of each insurance product item found on the requested insurance product.	fetchModules - list with objects for all the insurance product items found.
getPolicyByQuote (quoteNo, insuranceTypeId) Function checks if there is any existing policy that has the same quote number and insurance type. This is important for validating the generation process so there are no duplicates in the system.	quoteNo - the quote number obtained from the generation object. insuranceTypeId - the id of the insurance type requested.	policyData - the existing policy found in the system.
getInsuranceTypeIdByName (insuranceTypeName) Function returns the id of an insurance type based on the name given at generation.	insuranceTypeName - the insurance type name obtained from the generation object.	typeId - id of the insurance type found.

Function	Input parameters	Output parameters
isNullOrEmpty(value) Function checks if a value is invalid.	value - what needs to be validated.	True or false.
getProductByFilters(filters) Function selects from FTOS_IP_InsuranceProduct all the insurance products that meet the condition given through a chosen filter.	filters - object on which the filtering is being done. It can either be an insurance type name, a product code or an insurance product id.	rezProduct - list with objects for all the insurance products that have been found.
getSubcityId(subcityName, cityid) Function returns from the Subcity entity the id of the subcity that has a specific id and name.	subcityName - name of the subcity obtained from the generation object. cityid - id of the city for which the subcity is part of.	result[0] ["a_Subcityid"] - the id of the first subcity found.
getCityIdByName(cityName, districtid) Function returns from the City entity the id of the city that is part of a specific district and has a certain name.	cityName - name of the city obtained from the generation object districtid - id of the district for which the city is part of.	result[0] ["c_Cityid"] - the id of the first city found.

Function	Input parameters	Output parameters
compareValues(key, order) Function compares the values of two properties, when sorting an object. If the second parameter is missing, the order is set as being ascending. Then it goes on and returns a function that compares a property of the first element with the same property of the second element. Based on that comparison, the function returns 1 or -1, value to be used when sorting the object on a function call.	key - value of the object property on which the sorting is made. order - string stating the order of sorting (ascending/ descending).	comparison - value of 1 or -1 depending on which value is bigger.
getIdByAttrib(entityName, searchAttribute, searchValue) Function builds a fetch object using it's parameters. It returns the id attribute of an entity, based on an attribute and value given on a function call.	entityName - name of the entity the search is made on. searchAttribute - the attribute on which we want the condition to be made. searchValue - the value that the searchAttribute needs to have.	result[0] ["a_" + entityName + "id"]- Id of the first element found.
calculateEndDate(Validity, validityType, startDate) The function firstly checks what is the type of validity and then, based on that, it adds to the start date the number of validity units (years/ months/ days).	validity - number of validity units. validityType - type of validity selected for the policy. startDate - policy begin date.	endDate - the calculated end date for the policy.

Function	Input parameters	Output parameters
dateDiffInMonths(d1, d2) Function calculates the difference in months between two dates.	d1 - first date. d2 - second date.	months - months between the two dates.
validateInput(input) Function contains a set of rules used for validating the policy generation object represented by the input parameter. The conditions that the function has to pass in order to go ahead with the policy generation process are based on the issued date, start date, payment type, currency, quote number and insurance type.	input - object containing all the details necessary for generating a policy.	rez - object containing the generation status, an error message, an error code and a result list.
insertAccountObject(obj) If there isn't an existing account with the same uniqueIdentifier, based on the account type, the function inserts a new person in the Account entity and then returns the id of that entry.	obj - object containing all the account details.	objId - Id of the new account added.
checkExistingAccount(uniqueIdentifier) Function goes in the Account entity and checks if there is an existing account with the same unique identifier as the one given through the function parameter.	uniqueIdentifier - value used in the query as a condition for finding existing accounts.	acc[0].AccountId - Id of the first account found by the query.

Function	Input parameters	Output parameters
<p>addAccountsOnPolicy (policyId, contractorObj, insuredObj, beneficiaryObj)</p> <p>This function is in charge of updating the generated policy with the new ids for the contractor, insured and beneficiary.</p>	<p>policyId - id of the newly generated policy.</p> <p>contractorObj - object containing all the contractor details.</p> <p>insuredObj - object containing all the insured details.</p> <p>beneficiaryObj - object containing all the beneficiary details.</p>	<p>It returns an object containing all three ids.</p>
<p>addPolicyInsItemXCoveredRisk (policyInsuranceItemId, insuranceProductItemId, insuredAmount, currencyId)</p> <p>The function adds the covered risks of the insurance item on the policy. It calls the getItemCoveredRisk function with the id of the insurance product item as a parameter to get the covered risks and then it inserts all the details in the FTOS_INSPA_PolicyInsItemXCoveredRisk entity.</p>	<p>policyInsuranceItemId - id of the policy insurance item.</p> <p>insuranceProductItemId - id of the insurance product item.</p> <p>insuredAmount - the amount that is set as valueLimit if there is none on the covered risk.</p> <p>currencyId - the currency set for the risk value.</p>	<p>N/A</p>

Function	Input parameters	Output parameters
addProductItem (insertValues) The function gets all the necessary details of the product item ready to be added on the policy. After processing the data received during policy generation, the function inserts a new product item in the FTOS_INSPA_PolicyInsuranceItem entity.	insertValues - object containing all the required details for finding and adding a new product item on the new policy.	N/A
adjustSchedule (paymentScheduleId, PremiumAmount, installmentId, brokerCommission, tax) Function selects the specific installment that needs to be adjusted and then it updates it with the correct amount so that the sum of all the installments amount would be equal with the policy premium amount.	paymentScheduleId - id of the payment schedule which the installment is part of. PremiumAmount - total premium amount of the policy. installmentId - id of the installment that needs adjusting. brokerCommission - commission of the broker. tax - tax rate of the broker.	diff - the difference between the policy premium amount and the total of the other installments premium amount.
calculateFirstInstallmentDueDate(startDate) Function that calculates the first installment due date by subtracting one day from the start date.	startDate - date from which the day gets subtracted.	firstDateFinal - final version of the date.

Function	Input parameters	Output parameters
<p>calculateInstallments (firstInstallmentDueDate, installmentsNo, PremiumAmount, validity, policyValidityType)</p> <p>It calculates, based on the policy validity type, how many installments should be created for a particular schedule and it increments the due date accordingly. It also sorts the list of installments in the desired order and returns it.</p>	<p>firstInstallmentDueDate - due date of the first installment.</p> <p>installmentsNo - how many installments should be generated.</p> <p>PremiumAmount - policy premium amount.</p> <p>validity - number of the validity units.</p> <p>policyValidityType - type of the policy validity.</p>	<p>result - list of installment objects sorted by the due date.</p>

Function	Input parameters	Output parameters
<p><code>generateSchedule</code> <code>(policyId, param,</code> <code>policyValidityType,</code> <code>firstInstallmentDueDate,</code> <code>adjustFirstInstallment,</code> <code>adjustLastInstallment)</code></p> <p>This function generates a schedule of installments sorted by the due date on the new policy. It needs to be specified on a function call, by using the <code>adjustFirstInstallment</code> and <code>adjustLastInstallment</code> boolean parameters, which installment needs to be adjusted. If both are true the function throws an error.</p>	<p><code>policyId</code> - id of the newly generated policy.</p> <p><code>param</code> - object with parameters used for calculating the installments details.</p> <p><code>policyValidityType</code> - type of the policy validity.</p> <p><code>firstInstallmentDueDate</code> - due date of the first installment.</p> <p><code>adjustFirstInstallment</code> - boolean value on which the first installment is adjusted or not.</p> <p><code>adjustLastInstallment</code> - boolean value on which the last installment is adjusted or not.</p>	<p><code>installmentsList</code> - list of installment objects sorted by the due date.</p>
<p><code>getBrokerDataByName</code> <code>(brokerName)</code></p> <p>The functions returns from the FTOS_IP_InsuranceBroker entity a broker with a given name received as a parameter. If nothing was found, the function returns null.</p>	<p><code>brokerName</code> - the name of the broker on which the filtering is done.</p>	<p><code>fetchResult[0]</code> - the broker found by the query.</p>

Function	Input parameters	Output parameters
addInsuredObject (policyId, objectDetails) When this function gets called it performs three inserts, on three different entities: FTOS_INSQB_InsuredObject, FTOS_INSQB_Address and FTOS_INSQB_InsuredObjectProperty. With the help of other functions, it manipulates the data in order to insert it in the related table.	policyId - id of the newly generated policy. objectDetails - details of the object that gets insured.	It returns an object with all three ids of the new entries.
getProductItemsByProductId(productId) The function selects the product items configured on a product from the FTOS_IP_InsuranceProductItem entity, filtered by the product id. If any product item was found, the function goes on and gets all the modules configured on each item. If no items were found, the function returns null.	productId - id of the product on which the items are configured.	rez - object with all the product items found by the query together with their modules.
calculatePolicyPremiumAmount(itemList) This function calculates the policy premium amount, which is a sum between all the finalPremiumAmount attributes from all the product items.	itemList - list of all product items.	totalAmount - policy's total premium amount.

Function	Input parameters	Output parameters
calculateTotalInsuredAmount(itemList) This short function calculates the total insured amount, which is a sum between all the insuredAmount attributes the product items have.	itemList - list of all product items.	totalAmount - policy's total insured amount.
generatePolicy(inputValues) This function collects and manipulates all the required data for generating a new policy. It receives the generation object with all the policy details as a parameter and it builds different objects with them, used for inserting or updating data in the required entities. The main one is the objInsert object, which contains most of the policy details that get inserted in the FTOS_INSPA_Policy entity. The second one is the insertProductItemObj object used for adding all the product items on the newly generated policy. And last one is installmentsBrokerParameters containing details for generating the schedule. After all the data was added or updated, the function returns an object with some information about the new policy.	inputValues - object with the policy configuration details used in the policy generation process.	response - object with information about the new policy.

FTOS_INSPA_getModuleData

The endpoint is used for selecting all the data for a module that has the Id matching to the one obtained from the context JSON object.

Function: `getModuleData()` - The function contains a fluent query that returns the required attributes from the **FTOS_INSPA_PolicyInsuranceltem** entity for a module with a specific Id. After retrieving the results, the function calls the `setData` method for passing the object to the result parameter of the client-side callback function.

Input parameters: N/A.

Output parameters: `moduleList` - JSON object containing the module data.

FTOS_INSPA_getCoveredRisk

The endpoint is used for selecting all the data for a policy covered risk that has the Id matching to the one obtained from the context JSON object. When calling the endpoint with the `ebs.callActionByNameAsync` method, an object containing the `formData.id` gets passed as a parameter.

Function: `getCoveredRisk()` - The function contains a fluent query that returns the required attributes from the **FTOS_INSPA_PolicyInsItemXCoveredRisk** entity for a covered risk with a specific id. After retrieving the results the function calls the `setData` method for passing the object to the result parameter of the client-side callback function.

Input parameters: N/A.

Output parameters: `coveredRiskList` - JSON object containing the covered risk data

LH_PolicyGenerationAPI

This endpoint is used to generate a term life guaranteed/jet issue policy.

Input parameters:

- `insuranceTypeName: "Term Life"`
- `productCode: "TLF"`

- termLifeType: option set
- insuranceProductItemList
 - code: "TLF0"
 - sumInsured: numeric
 - regularPremiumBase: numeric
 - annualizedPremiumBase: numeric (attribute to be added, numeric)
 - code: "CIO"
 - isCriticalIllnessIncluded: (attribute to be added, Boolean type)
 - regularPremiumCriticalIllness: numeric
 - annualizedPremiumCriticalIllness: numeric (attribute to be added)
 - regularPremiumTotal: numeric
 - annualizedPremiumTotal: numeric (attribute to be added)
 - policyFee: numeric
- issuedDate (invariant Date type)
- startDate (invariant Date type)
- renewedPolicyNo: null (text type)
- renewTypeld: null (option set)
- quoteNo: text
- validityType: option set (days, months, years)
- validity: numeric
- policyTerm: numeric

- frequency: option set
- currency: currency code
- paymentType: option set (FTOS_INSPA_PolicyPaymentType, with "directDebit" our only option here)
- agent:
 - FTOS_INS_Agentid (lookup)
- broker:
 - brokerId: null (attribute to be added, lookup)
- contractor (policyholder):
 - Name
 - PIN
 - dateOfBirth
 - email
 - phone
- insured:
 - Name
 - PIN
 - dateOfBirth
 - email
 - phone

Response:

- policyBeginDate;
- policyEndDate;

- policyId;
- policyNumber.

Validations:

- If the issue date value provided on `issuedDate` is less than current date, the following error message is displayed: `"Invalid issued date!"`;
- If the start date value provided on `startDate` is less than or equal with `issuedDate` value, the following error message is displayed: `"Invalid start date!"`;
- If the value provided on `paymentType` is not valid, not part of the accepted values, the following error message is displayed: `"Invalid payment Type!"`;
- If the currency code provided on `currency` is not identified, the following error message is displayed: `"Invalid currency!"`;
- If another policy of the same insurance type is already registered, the following error message is displayed: `"Existing policy for the same quote and insurance type!"`.

Get Policy Data API

The `FTOS_GetPolicyDataAPI` script is called with an object as data and calls the `getPolicyData` function from the "policyDataAPIs" server automation script library.

An example of calling this function is given below:

```

1 | var p = {
2 |     policyNo: "80000753",
3 |     validityDate: '2021-10-24'
4 | }
5 |
6 | ebs.callActionByNameAsync('FTOS_GetPolicyData_API', p)
7 | .then(function(e) {

```

```

8 | console.log(e.UIResult);
9 | });

```

Endpoints

FTOS_GetPolicyData_API - This endpoint is used to run the FTOS_GetPolicyData_API server automation script.

Request data parameters:

- policyNo: policy number (mandatory);
- validityDate: The request date (optional).

Response:

```

1 | {
2 |   "isSuccess": true,
3 |   "errorMessage": null,
4 |   "errorCode": null,
5 |   "result": {
6 |     "PolicyIssuedDate": "2021-10-05",
7 |     "PolicyBeginDate": "2021-10-06",
8 |     "PolicyEndDate": "2022-10-09",
9 |     "Coverages": [
10 |       {
11 |         "Code": "DPA",
12 |         "BeginDate": "2021-10-10",
13 |         "EndDate": "2022-10-09",
14 |         "AmountInsured": 50000.0,
15 |         "PremiumAmount": 96.0,
16 |         "Currency": "RON",
17 |         "IndemnityLimit": 50000.0,
18 |         "WaitingPeriod": 0.0,
19 |         "WaitingPeriodType": null,
20 |         "Modules": [
21 |           {
22 |             "Code": "DPA1",
23 |             "AmountInsured": 50000.0,
24 |             "Currency": "RON",
25 |             "Risks": [
26 |               {
27 |                 "Name": "Personal Accident",
28 |                 "ImplicitReserve": 2000.00,
29 |                 "ValueLimit": 50000.00,

```

```

30         "Currency": "RON"
31     }
32 ]
33 }
34 ]
35 },
36 {
37     "Code": "ICPA",
38     "BeginDate": "2021-10-10",
39     "EndDate": "2022-10-09",
40     "AmountInsured": 25000.0,
41     "PremiumAmount": 528.0,
42     "Currency": "RON",
43     "IndemnityLimit": 25000.0,
44     "WaitingPeriod": 0.0,
45     "WaitingPeriodType": null,
46     "Modules": [
47         {
48             "Code": "ICPA1",
49             "AmountInsured": 25000.0,
50             "Currency": "RON",
51             "Risks": [
52                 {
53                     "Name": "Personal Accident",
54                     "ImplicitReserve": 2000.00,
55                     "ValueLimit": 25000.00,
56                     "Currency": "RON"
57                 }
58             ]
59         }
60     ]
61 },
62 {
63     "Code": "PDA",
64     "BeginDate": "2021-10-10",
65     "EndDate": "2022-10-09",
66     "AmountInsured": 35000.0,
67     "PremiumAmount": 806.4,
68     "Currency": "RON",
69     "IndemnityLimit": 35000.0,
70     "WaitingPeriod": null,
71     "WaitingPeriodType": null,
72     "Modules": [
73         {
74             "Code": "PDA1",

```

```

75         "AmountInsured": 35000.0,
76         "Currency": "RON",
77         "Risks": [
78             {
79                 "Name": "Personal Accident",
80                 "ImplicitReserve": 2000.00,
81                 "ValueLimit": 35000.00,
82                 "Currency": "RON"
83             }
84         ]
85     }
86 ]
87 },
88 {
89     "Code": "MEACC",
90     "BeginDate": "2021-10-10",
91     "EndDate": "2022-10-09",
92     "AmountInsured": 20000.0,
93     "PremiumAmount": 460.8,
94     "Currency": "RON",
95     "IndemnityLimit": 20000.0,
96     "WaitingPeriod": 0.0,
97     "WaitingPeriodType": null,
98     "Modules": [
99         {
100             "Code": "MEACC1",
101             "AmountInsured": 20000.0,
102             "Currency": "RON",
103             "Risks": [
104                 {
105                     "Name": "Personal Accident",
106                     "ImplicitReserve": 2000.00,
107                     "ValueLimit": 20000.00,
108                     "Currency": "RON"
109                 }
110             ]
111         }
112     ]
113 },
114 ],
115 "BusinessStatus": "Version Closed"
116 }
117 }

```

Response description:

- **isSuccess:** Marks if the request was successful or not;
- **Error code:** Error code;
- **Error message:** Error message;
- **Result:** Null or an object returned as response for the API call.

Error messages:

Code	Text	Description
ERR01.01	ERR01.01 - Invalid/missing policy number!	There is no policy in the system with the requested number!
ERR01.02	ERR01.02 - Invalid date format!	The date should have the following format: 'xxxx-xx-xx' or 'xxxx/xx/xx'!
ERR01.03	ERR01.03. - Invalid date!	The policy was not valid at the requested date!
ERR02.01	ERR02.01 - No coverage!	The policy had no coverages at the requested date!
ERR03.01	ERR03.01 - No modules for coverage!	This coverage had no modules associated at the requested date!
ERR04.01	ERR04.01 - No covered risks!	This module had no covered risks at the requested date!

Server Automation Script Library

The following server automation script library is available.

policyDataAPIs

This library contains a main global function called `PolicyData()` which uses different functions to return a policy's details from a certain date based on the policy number and a date. The `PolicyData()` uses

the following functions:

PolicyData(); - This main function returns an object with the following methods:

- **getMainPolicy** - This method calls the **getMainPolicy()** function.
- **getCoverageModules** - This method calls the **getCoverageModules()** function.
- **getCoveredRisks** - This method calls the **getCoveredRisks()** function.
- **getPolicyData** - This method calls the **getPolicyData()** function.

invariantDateToDate(date); - This function transform an invariant date into a date without the time. Will return only the date.

Input parameters:

- **date**- (invariantDate) - The invariant date.

Output parameters:

- **invariantDate** - (Date) - The date in the new format.

getMainPolicy(policyNo, validityDate); - This function returns an object with different policy data based on the policy number and a date.

Example of calling the function: `PolicyData.getMainPolicy('80000753', '2021-10-24');`

Input parameters:

- policyNo - (string) - The policy number.
- validityDate - (string) - The request date in the format: "2021-11-01".

Output parameters:

- If the policy exists and it was valid at the requested date, the object contains the following params:
 - isSuccess: true
 - errorMessage: null
 - errorCode: null
 - result: an object with the following policy data:
 - PolicyId - (string) - The policy ID
 - PolicyIssuedDate - (invariant date) - The policy issued date
 - PolicyBeginDate - (invariant date) - The policy begin date
 - PolicyEndDate - (invariant date) - The policy end date
 - PolicyAttributeVersionDate - (invariant date) - The policy attribute version date
 - PolicyBusinessStatus - (string) - The policy business status.
- If the policy does not exist or it wasn't valid on the requested date, the object contains the following:
 - isSuccess: false
 - errorMessage: The error message

- `errorCode`: The error code
- `result`: null

`getPolicyCoverages(policyId)`; - This function returns an object with the policy coverages based on the policy ID.

Example of calling the function:

`PolicyData.getPolicyCoverages(policyId)`

Input parameters:

- `policyId` - (string) - The policy ID.

Output parameters:

- If the policy has coverages, the object contains the following params:
 - `isSuccess`: true
 - `errorMessage`: null
 - `errorCode`: null
 - `result`: an array with one or many objects with the following data:
 - `CoverageId` - (string) - The coverage ID.
 - `CoverageCode` - (string) - The coverage code.
 - `CoverageBeginDate` - (invariant date) - The coverage begin date.
 - `CoverageEndDate` - (invariant date) - The coverage end date.
 - `CoverageAmountInsured` - (numeric) - The coverage insured amount.

- CoveragePremiumAmount - (numeric) - The coverage premium amount.
- CoverageCurrency - (string) - The coverage currency code.
- CoverageIndemnityLimit - (numeric) - The coverage indemnity limit amount.
- CoverageWaitingPeriod - (numeric) - The coverage waiting period.
- CoverageWaitingPeriodType - (string/optionSet) - The coverage waiting period.
- If the policy does not have any coverages, the object contains the following:
 - isSuccess: false
 - errorMessage: The error message
 - errorCode: The error code
 - result: null

`getCoverageModules(parentId);` - This function returns an object with the coverage modules based on the coverage ID.

Example of calling the function:

`PolicyData.getCoverageModules(coverageId)`

Input parameters:

- parentId - (string) - The coverage ID.

Output parameters:

- If the coverage have modules, the object contains the following params:

- isSuccess : true
- errorMessage: null
- errorCode: null
- result: an array with one or many objects with the following data:
 - ModuleId - (string) - The module ID.
 - ModuleCode - (string) - The module code.
 - ModuleAmountInsured - (numeric) - The module insured amount.
 - ModuleCurrency - (string) - The module currency code.
- If the coverage does not have any modules, the object contains the following:
 - isSuccess: false
 - errorMessage: The error message
 - errorCode: The error code
 - result: null

`getCoveredRisks(moduleId);` - This function returns an object with the covered risks based on the module ID.

Example of calling the function: `PolicyData.getCoveredRisks(moduleId)`

Input parameters:

- moduleId - (string) - The module ID.

Output parameters:

- If the module have covered risks, the object contains the following params:
 - isSuccess : true
 - errorMessage: null
 - errorCode: null
 - result: an array with one or many objects with the following data:
 - RiskName - (string) - The risk name.
 - RiskImplicitReserve - (numeric) - The risk implicit reserve.
 - RiskValueLimit - (numeric) - The risk value limit.
 - RiskCurrency - (string) - The risk currency code.
- If the module does not have any covered risks, the object contains the following:
 - isSuccess: false
 - errorMessage: The error message
 - errorCode: The error code
 - result: null

getPolicyData(token); - This function returns the queried policy with the help of the functions presented above.

Example of calling the function: `PolicyData.getPolicyData(token)`

Input parameters:

- token - (object) - An object which contains the following:
 - policyNo - (string) - The policy number.
 - validityDate - (string) - The date in the following format: "2021-11-01".

Output parameters:

- If the policy exists and it was valid at the requested date, the object will contain:
 - isSuccess : true
 - errorMessage: null
 - errorCode: null
 - result: an object with the following data:
 - PolicyBeginDate - (invariant date) - The policy begin date.
 - PolicyEndDate - (invariant date) - The policy end date.
 - Coverages - (array) - An array which contains one or many objects with the following data:
 - CoverageCode - (string) - The coverage code.
 - CoverageBeginDate - (invariant date) - The coverage begin date.
 - CoverageEndDate - (invariant date) - The coverage end date.
 - CoverageAmountInsured - (numeric) - The coverage insured amount.
 - CoveragePremiumAmount - (numeric) - The coverage premium amount.

- CoverageCurrency - (string) - The coverage currency code.
- CoverageIndemnityLimit - (numeric) - The coverage indemnity limit amount.
- CoverageWaitingPeriod - (numeric) - The coverage waiting period.
- CoverageWaitingPeriodType - (string/optionSet) - The coverage waiting period.
- Modules - (array) - An array which contains one or many objects with the following data:
 - ModuleCode - (string) - The module code.
 - ModuleAmountInsured - (numeric) - The module insured amount.
 - ModuleCurrency - (string) - The module currency code.
 - CoveredRisks - An array containing one or many objects with the following data:
 - RiskName - (string) - The risk name.
 - RiskImplicitReserve - (numeric) - The risk implicit reserve.
 - RiskValueLimit - (numeric) - The risk value limit.
 - RiskCurrency - (string) - The risk currency code.

- **BusinessStatus-** (string) - The policy business status.
- If the policy does not exist or it was not valid at the requested date, the object will contain the following:
 - **isSuccess:** false
 - **errorMessage:** The error message
 - **errorCode:** The error code
 - **result:** null

PolicyStatusChange API

This API is used in order to automatize the transition of a policy in the **Issued** and **In Force** statuses.

The PolicyStatusChangeAPI endpoint uses a library that contains functions used in the process.

PolicyStatusChange Library

This library contains 2 main functions (General and Notifications), and each of these 2 functions contain different functions that help on policy status change process.

The first main function (General) contains the functions presented below and returns them as methods from an object. The object construction is given below:

```
1 | {  
2 |   findPolicy: findPolicy,  
3 |   validateInput: validateInput,  
4 |   updatePolicy: updatePolicy
```

5 | }

(General) functions:

`findPolicy(inputValues)`: function that executes a query to find a specific policy.

Input parameters:

- inputValues - parameters

Output parameters:

- query - found policy

`validateInput(inputValues)`: function that validates the input.

Input parameters:

- inputValues - parameters

Output parameters:

- rez

`updatePolicy(inputValues)`: function that make the updates for the status and other data .

Input parameters:

- inputValues - parameters

Output parameters:

- response

The second main function (Notifications) is used to send notifications on the policy status change (transitions) and contains the functions presented below and returns them as methods from an object. The object construction is given below:

```

1  {
2      getPolicyStatusName: getPolicyStatusName,
3      getAccountDataForNotification:
4  getAccountDataForNotification,
5      checkEmailTemplateExists: checkEmailTemplateExists,
6      generateNotifcation: generateNotifcation,
7      sendNotification: sendNotification
  }

```

(Notifications) functions:

getPolicyStatusName(statusName); function that returns the business status display name of a entity.

Input parameters:

- statusName - (string) - The entity status name

Output parameters:

- query - Array with the results

getAccountDataForNotification(accountdId); function that returns different attributes from the “Account” entity, based on the account ID.

Input parameters:

- accountdId - (string) - The account id;

Output parameters:

- query- Array that contains an object with the following results:
 - FirstName
 - LastName
 - Email
 - Phone

`checkEmailTemplateExists(templateName);`: function that checks if an email template exists in the emailTemplate entity.

Input parameters:

- `templateName` - (string) - The template name;

Output parameters:

- `tokens` - An object with the following results:
 - `TemplateName`

`generateNotifcation(values);`: function that creates the object that will be used to send a notification.

Input parameters:

- `values` - (object) - The context object

Output parameters:

- `tokens` - An object with the following results
 - `FirstName`
 - `LastName`
 - `Name`
 - `PolicyNumber`
 - `IssuedDate`
 - `Email`
 - `BusinessStatus`
 - `PolicyEndDate`

`sendNotification(policyData, templateName);` function that sends a notification with the help of the functions presented above and only if the `PolicyAdminUsed` parameter is set to 1. This function will be triggered only on the following `FTOS_INSPA_Policy` transitions status change:

- Decline by screening
- Closed by Claim
- Withdraw on client's request
- Cancelled
- Lapsed

Input parameters:

- `policyData`- (object) - The context object
- `templateName`- (string) - The template name

Output parameters:

- N/A

Call:

```

1  var p = {
2      "policyIdentifiers": {
3          "policyId": null,
4          "policyNo": '8000043'
5      },
6      "updateType": "Issued",
7      "policyUpdates": {
8          "policyNo": null,
9          "policyBeginDate": null,
10         "policyValidityType": null,
11         "policyValidity": '6'
12     }
13 }
14
15 ebs.callActionByNameAsync("PolicyStatusChangeAPI", p)
16     .then(

```

```

17 |         function (e) {
18 |             console.log(e.UIResult.Data)
19 |         }
20 |     });

```

Validations:

- "ERR06.01 - PolicyId or PolicyNo could not be null!": could not find a policy if at least one of the parameters is not specified;
- "ERR06.03 - The Policy is not in Proposal status, therefore the transition to Issued is not possible!": when transitioning the policy to the **Issued** status, the policy need to be in the **Proposal** status;
- ERR06.04 - The Policy is not in Issued status, therefore the transition to In Force is not possible!";: when transitioning the policy to the **In Force** status, the policy need to be in the **Issued** status.

Response:

```

▼ {result: {...}, isSuccess: true} ⓘ
  isSuccess: true
  result:
    ▼ policyData: Array(1)
      ▼ 0:
        newPolicyStatus: "Issued"
        policyBeginDate: {invariantDate: '2021-11-05'}
        policyEndDate: "2022-05-04"
        policyId: "6d979535-385b-4ab0-8338-b9749245b13f"
        policyNo: "80000891"
        validity: "6"
        validityType: "Months"
        ▶ [[Prototype]]: Object
        length: 1
        ▶ [[Prototype]]: Array(0)
        ▶ [[Prototype]]: Object
        ▶ [[Prototype]]: Object

```

Policy Versioning

There are cases, like policy renewal when you need to create a new version of a specific policy. To accommodate the versioning functionality, from an insurance business perspective, the **Core Policy Admin** solution uses a combination of the **FintechOS standard versioning process mechanism** and custom development. Consequently, you use the **FTOS_VersioningHelper** client side library and follow the standard procedure when configuring version settings, version settings items and

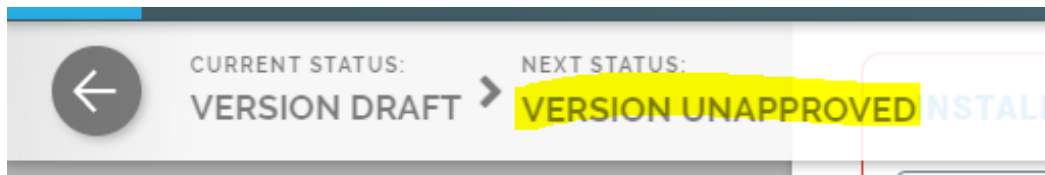
entity settings. However, for policy versioning, you use the **FTOS_VersioningHelper_Edit** client side library in order to keep some attributes **Read Only**, even when the policy is in **Version Draft** business status, with **isEditable** option enabled.

Policy Versioning Process Description

The versioning functionality is available only for issued, enforced or approved policies, for users that have the rights to use the functionality.

After clicking the **New Version** button on the initial policy (P1), a policy clone (P2) is registered, in **Version Draft** status.

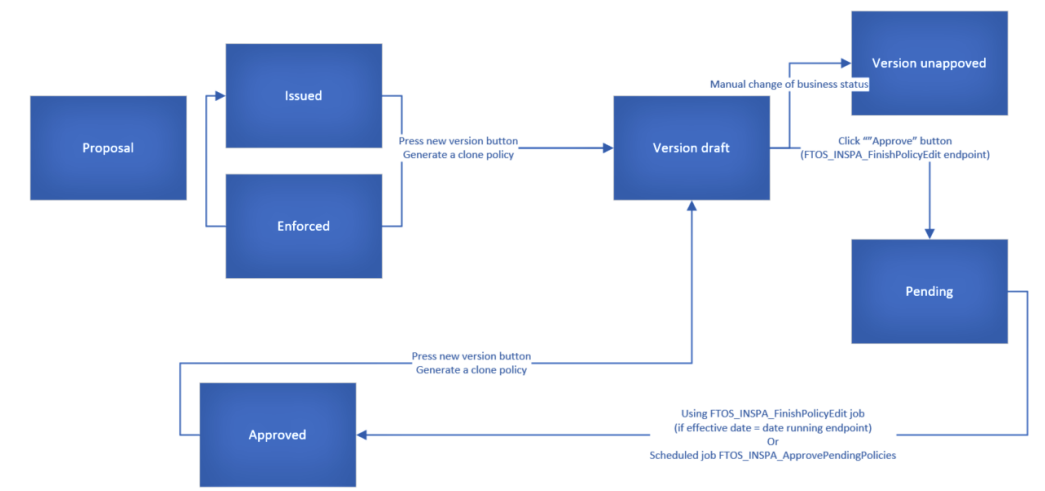
The status of the new policy P2 can be changed to **Version Unapproved**, following the standard process (manually change the status of the record).



The standard approval process (changing the status from Version **Draft** to **Approved**) was replaced by an automated process, especially implemented for the policy.

The user needs to fill in the effective date for the change and press **Approve** button. The version will remain in **Version Draft** status until the effective date is reached and it will be automatically transitioned to **Approved** status, on that day.

If the effective date is also the date when the user validated the changes by pressing “Approve button”, then the request will transition automatically from **Version Draft** to **Pending** and also from **Pending** to **Approved**.



If the Policy Begin Date is greater than the current date, then the policy status will be transitioned to the **Issued** status.

If the Policy End Date is lesser than the current date, then the policy status will be transitioned to the **Maturity** status.

If the conditions above are not respected, then the policy will be transitioned to the previous status of the policy.

The following processes will generate automatically newly approved versions of the policies:

- **Cancellation** (Ex-Surrender) - for the updates of end date and premium;
- **Lapsing** - or the updates of end date and premium.
- **Cancellation** without an update of the premium (eg. Cancellation with Claims), the schedule has to be updated with a schedule including just the paid installments in the paid status and the future unpaid installments in Canceled status (new status for PaymentScheduleDetail);
- **Lapsing** - same as above;
- **Cancellation** with premium returned the schedule has to be updated with a schedule including the paid installments, the future unpaid installments in **Canceled** status plus an additional installment which is

equal with :minus: the value of the returned amount. The due date will be equal to the approval date of the cancellation and the status of this payment schedule detail will be On Time (TBD).

After the automatic version approval process, the status for policy must be changed according to the process which triggered the versioning:

- If the version approval has been made after the **Lapsing** process - the policy status is transitioned from **Approved** to **Lapsed**.
- If the version approval has been made after the **Cancellation** process - the policy status will be transitioned from **Approved** to the specific status for **Cancellation**: Closed by claim, Decline by screening etc.

Business workflow configuration actions

Pending - Approved - When you click **Approve**, the script checks the effective approval date. If it is less than or equal to the current date, it will move the policy version record into the **Approved** status. If not, the record remains in **Pending** status until the **FTOS_INSPA_ApprovePendingPolicies** job moves it to **Approved**, according to the effective chosen date.

Version Draft - Pending - If the necessary validations are met, clicking **Approve** changes the status of the cloned policy from **Version Draft** to **Pending**.

A new version of the policy can be made from the **Issued** and **InForce** statuses.

The **Approved** status is not visible in the interface, and the transition is automatically made to the business status of the policy.

For advanced versioning configurations, endpoints, and server side scripts, check the [Advanced Versioning Configurations](#) page.

Master Policy Versioning Process Description

1. Access an existing Master Policy and click the **Insert (+)** button, called **New Version**, available on the top right corner.

2. An edit form of that Master Policy opens, with a new version in the **Version Draft** status, so you will be able to change and update the information related to that Master Policy.
 - When the Edit form opens, the **Master Policy Summary** tab is automatically selected.
 - Both tabs, **Master Policy Summary** and **Policies List**, are displayed.
3. The status of the newly added version of the Master Policy is **Version Draft** and the next status is also displayed, **Version Unapproved**.

Once the edit form opens, the user is able to change the following:

- The values from any field of the **Master Policy** section, except the **Master Policy No** and the **Currency** fields. The **Master Policy No** remains the same like the one before versioning;
- the text from the **Mentions** section;
- the values from the **Premium Payment Schedule** section.
 - The user is able to only update the installments in the **On Time** status;
 - Once the values here change and a new version of the Master Policy is added, a new version is added for the **Premium Payment Schedule** section.

NOTE In the manual versioning phase, the changes made to the Master Policy are not subject to validation. Also no changes are triggered for the associated policies. This is done in a later phase and through an MTA process at Master Policy level for example.

After the desired changes are made in the **Version Draft** status, the user is able to:

- Unapprove this version, by manually switching the status of the version from the top left corner;
- Approve the new version by filling in the **Effective Date** of the version and clicking the **Approve** button.

- The effective date is available at the end of the Master Policy form in the **Master Policy Summary** section, and it is mandatory if the user wants to approve a version;
- The **Effective Date** cannot be sooner than the beginning date of the Master Policy;
- If the user will press “Save and close” or “Save and reload” (top right corner buttons) before pressing

Core Policy Admin Formulas

The formulas are defined using Business Formulas in Innovation Studio. The current page reveals the logic behind formulas used in the Life and Health module. For more information on how the Business Formulas works, check out the [Business Formulas](#) chapter in the [Innovation Studio](#) product documentation.

The following formulas and calculations are used in the Core Policy Admin module.

Register/Cancel an MTA Request

- ***The additional coverage premium amount*** = *(new annual premium - initial annual premium)/12* No. of uninsured months*
- ***Updated coverage premium amount*** = *Initial premium amount * (12- No of uninsured months)/12 + The additional coverage premium amount*
- ***Additional policy premium amount*** = *Sum (The additional coverage premium amounts)*
- ***Updated policy premium amount*** = *Sum (Updated coverage premium amount)*

MTA Refactoring According to Pro Rata Type

Daily prorata = 1/365 from the premium amount of a contract.

Monthly prorata = 1/12 from the premium amount of a contract.

If Prorata type = daily

- ***AdditionalCoveragePremiumAmount*** = $(\text{coverageNewPremium} - \text{initialPremiumAmount}) / 365 * \text{uninsuredPeriod}$
- ***UpdatedCoveragePremiumAmount*** = $\text{initalPremiumAmount} * (365 - \text{uninsuredPeriod}) / 365$
- ***FreqAdditionalCoveragePremiumAmount*** = $(\text{newPremiumAmount} - \text{initialPremiumAmount}) / 365 * \text{uninsuredPeriod}$
- ***FreqUpdatedCoveragePremiumAmount*** = $(\text{initialPremiumAmount} * (365 - \text{uninsuredPeriod}) / 365) + (\text{newPremiumAmount} / 365 * \text{uninsuredPeriod})$

If Prorata type = monthly

- ***AdditionalCoveragePremiumAmount*** = $(\text{coverageNewPremium} - \text{initialPremiumAmount}) / 12 * \text{uninsuredPeriod}$
- ***UpdatedCoveragePremiumAmount*** = $\text{initalPremiumAmount} * (12 - \text{uninsuredPeriod}) / 12$
- ***FreqAdditionalCoveragePremiumAmount*** = $(\text{newPremiumAmount} - \text{initialPremiumAmount}) / 12 * \text{uninsuredPeriod}$
- ***FreqUpdatedCoveragePremiumAmount*** = $(\text{initialPremiumAmount} * (12 - \text{uninsuredPeriod}) / 12) + (\text{newPremiumAmount} / 12 * \text{uninsuredPeriod})$

Master Policies

In the Master Policy entity, the `noOfValidityMonths` attribute represents the number of calendar months between the begin date and the end date of the Master Policy. It is calculated as per below:

No of validity months = *Master Policy Begin Date - Master Policy End Date*

The number of installments of the Master Policy (noOfInstallments attribute, numeric type) is calculated based on the attributes: number of validity months (calculated above) and payment frequency of the Master Policy.

Payment frequency	No of installments when $v > 12$ months	$6 \leq v < 12$ months	$v < 6$ months
annually	$\text{roundup}(v/12;0)$	N/A	N/A
semi-annually	$\text{roundup}(v/6;0)$	$\text{roundup}(v/6;0)$	N/A
quarterly	$\text{roundup}(v/3;0)$	$\text{roundup}(v/3;0)$	$\text{roundup}(v/3;0)$
monthly	v	v	v

Where:

- v = The number of validity months of the Master Policy (noOfValidityMonths);
- N/A = The system does not allow this combination.

Installation

Follow the guidelines below to import and configure the **Core Policy Admin** module.

NOTE In order to help you navigate easily between the installation steps, the images supporting this guide were allowed to include some parts of the screen background. However, note that your screen might have a different background color, since this is a customizable feature of the **FintechOS Platform**. Still, the menu items are the same. And they are what you would be looking for and clicking on when you perform the installation.

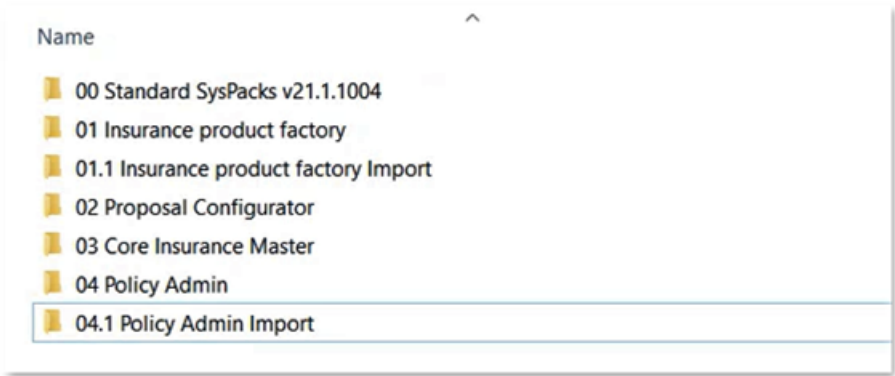
Installing Core Policy Admin 3.2.0

1 Prerequisites

Platform version compatibility: **HPFI 21.2.2.2** and above.

Dependencies: **SySDigitalSolutionPackages v21.2.2000.zip** or higher.

Insurance Apps Dependencies: **Core Insurance Master V2.2.0**.



If you already have a **Policy Admin Import** asset installed on your environment, you need to apply the appropriate upgrade asset so start by checking if there is any upgrade listed for your solution before applying this package.

2 Import Packages

Package description:

- **Core Policy Admin v2.0** - Data package for managing policies.
- **Policy Admin Import v2.0** - Data import configuration package for roles and insurance parameters.

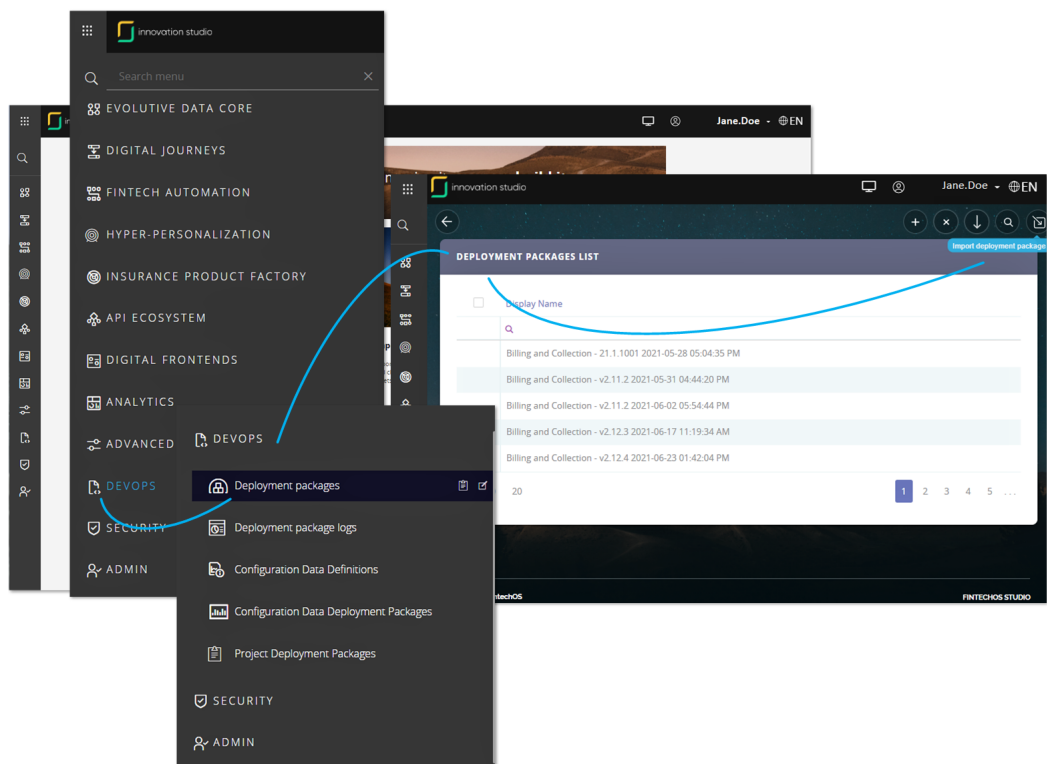
IMPORTANT! Make sure that the **Policy Admin Import v2.0** is imported **after** the **Core Policy Admin v2.0** package, otherwise the policy automatic transitions won't work as expected.

3 Installation

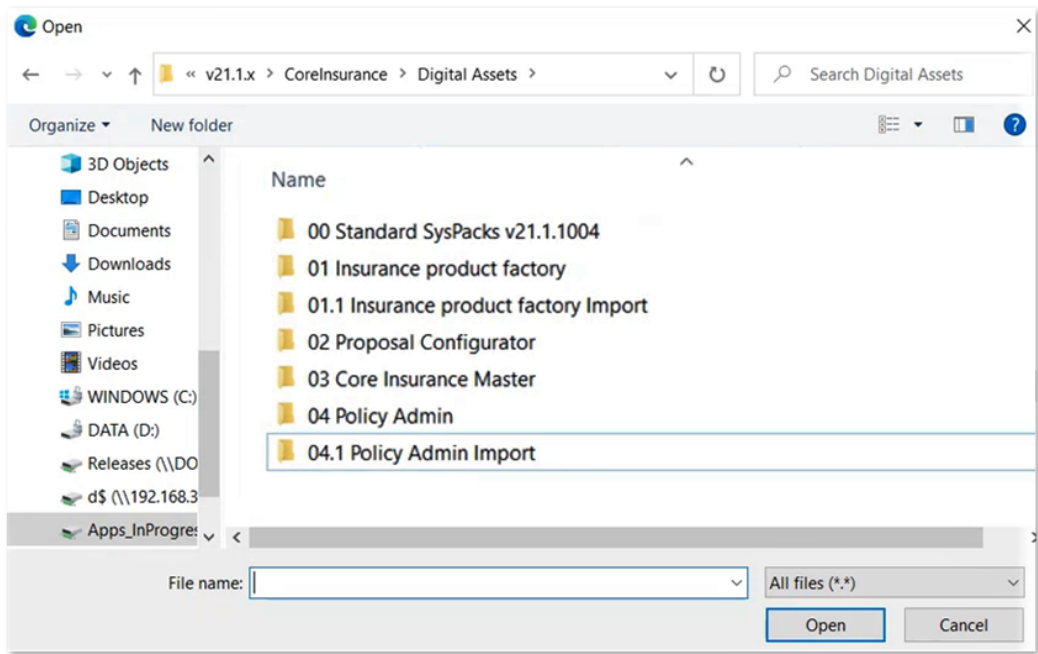
The following is the standard procedure for how to import a digital asset:

Go to **Innovation Studio** and log in with **Developer** credentials.

Navigate to the **DevOps** menu entry. From the dropdown, click on **Deployment Package** to open the **Deployment Package List** page. On the top right-hand side corner, click the **Import deployment package** icon.



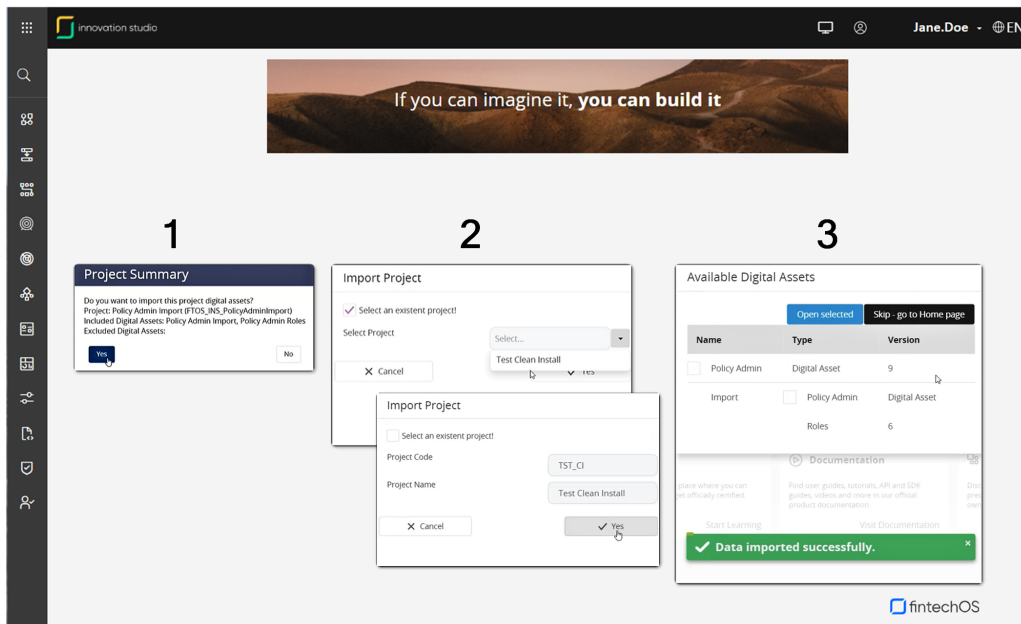
The local **File Explorer** window opens. Select from the local drive the file to be imported. Pay attention to the order of importing the chosen package, as stated in the **Import Packages** section and the **Prerequisites** section.



After selection, a new form is launched showing the contents of the package and asking if you want to import those. Check the list and click **Continue Import** if you are satisfied.

PACKAGE INTEGRITY VALIDATIONS						
Digital Asset	Version	Version On Envt...	Minimum Platf...	Environment PL...	Validation Message	Validation Status
Policy Admin	15		21.1.5.0	21.1.5.0	Existing platform version is ok	Passed
Policy Admin Menu	15		21.1.5.0	21.1.5.0	Existing platform version is ok	Passed
Policy Admin	15		21.1.5.0	21.1.5.0	New	Passed
Policy Admin Menu	15		21.1.5.0	21.1.5.0	New	Passed

A pop-up asking you to confirm that you want to import appears. Click **Yes** to continue. After this confirmation, another window opens where you indicate the **Project** name. Either give a new name for your import or select the name of a previous import project, from the helper dropdown. You might want to select the same name for the project that you use to import all the packages of the Insurance suite, as enumerated in the [Prerequisites](#) section. Confirm the **Project Name** selection and click **Import** to start the job.



After the import is finished, you can see the success message and the **Deployment Package Log** window which allows you to see into the details of the import project.

4 After installment configurations

4.1 Enable System Parameter

Proceed to enable the **PolicyAdminUsed** system parameter as described below:

Go to **Studio** and log in with **Developer** or **Release Manager** credentials.

Navigate to the **DevOps** menu entry. From the dropdown, click on **Deployment Package** to open the **Deployment Package List** page. From the list, select the **Core Insurance Master** Digital Asset and double click to open the Digital Asset **Form**. Inside the form, go to the **General** tab and click **Set as context digital asset**. This way, you allow the **Core Insurance Master** to manage the policy flows stored by the **Core Policy Admin** module.

1 General 2 Configuration Items 3 Related Projects 4 Data Import Files 5 Dependencies 6 Configuration Items Migration

Lock Digital Asset

Code: 1234 Name: LoanOrigination_v2

Description: This is a demo asset.

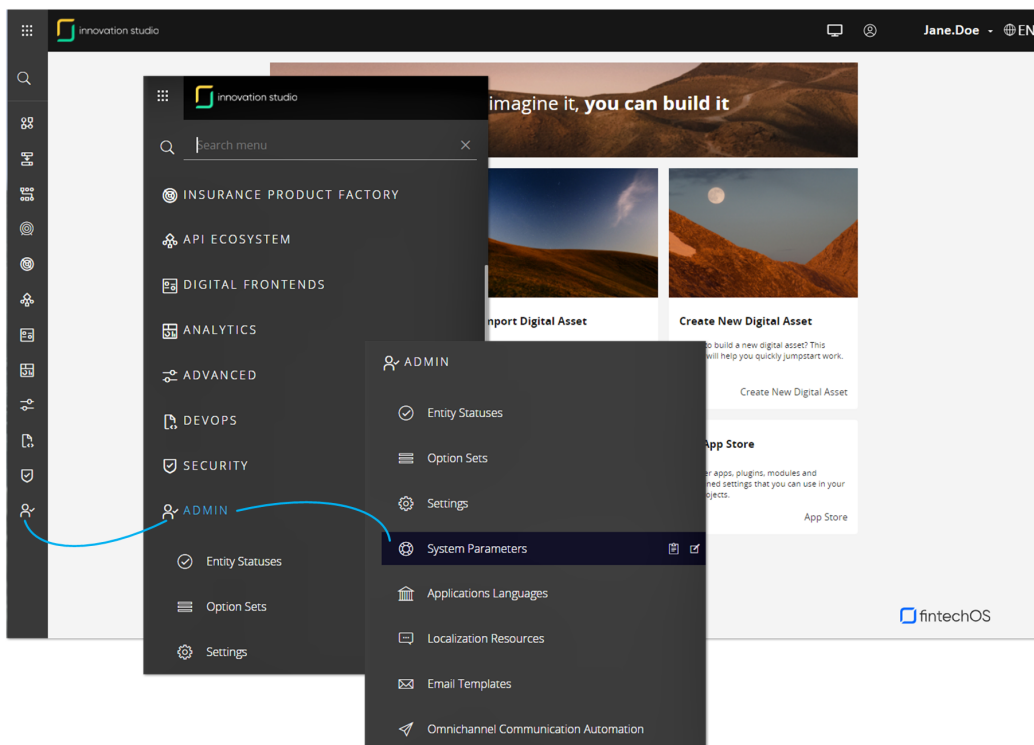
Type: Digital Asset

Version: 1 Status: Unlocked

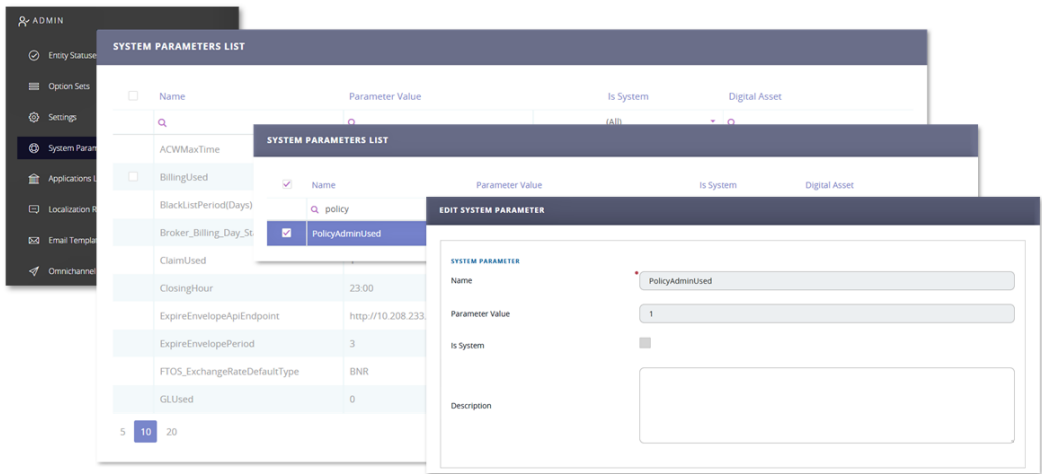
Obsolete: ☐ Creation Mode: Primary

[Set as context digital asset](#)

After this step, go back to the main menu and navigate to the **Admin** menu entry. From the dropdown, click on **System Parameters** to open the **System Parameters List**.



Depending on the number of views set on the list, it is possible that you don't see the **PolicyAdminUsed** parameter. Inside the list, use the **Search by Name** option to search the **PolicyAdminUsed** parameter and double click to open it for edit. Inside the **Edit System Parameter** form, set the value to 1, then click **Save and close**, at the top right corner of your screen.



IMPORTANT! The **Core Policy Admin** automatic transitions won't be enabled if you don't enable the **PolicyAdminUsed** parameter.

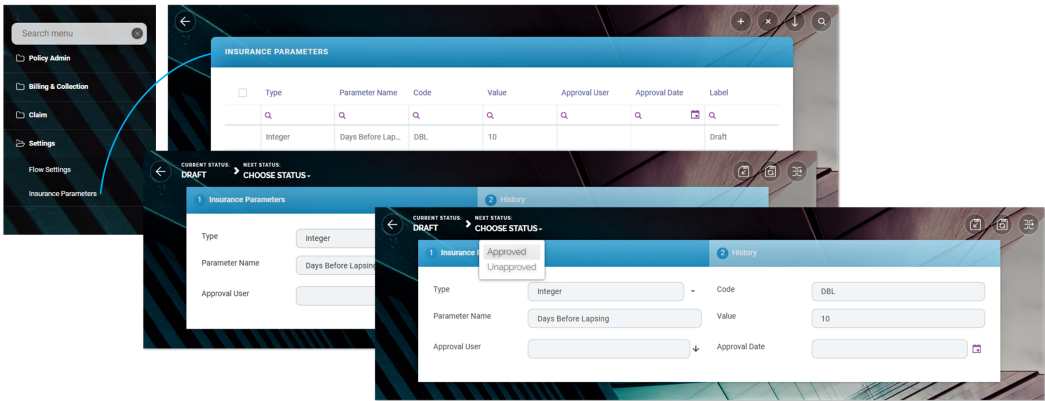
4.2 Approve Insurance Parameters

Go to **Portal** and log in with **Developer** or **Release Manager** credentials.

On the left side of the screen, open the main menu. From the dropdown, choose **Settings** and click to open it. Got to **Insurance Parameters** and click to open.

From the **Insurance Parameters List** , select a parameter and double click to open it for editing.

After the **Parameter** window opens, from top left corner of your screen, change parameter status from **Draft** to **Approved**.



Repeat this procedure for all the **Insurance Parameters** in **Draft** status.

IMPORTANT! You must approve all **Insurance Parameters** otherwise they won't be considered in automatic **Core Policy Admin** transitions.

4.2 Notifications

In order to send the notifications, the SMTP credentials are needed to be added in the job server and portal's webconfig file. Please check if there are SMTP keys inserted (please find below some credentials only for example).

```
<add key="SMTP:Port" value="****" />
<add key="SMTP:Host" value="****" />
<add key="SMTP:EnableSSL" value="0" />
<add key="SMTP:User" value="****" />
<add key="SMTP:Password" value="****" />
<add key="DefaultFromEmail" value="****" />
```

*** = your SMTP information

Installing Core Policy Admin Import 3.2.0

1 Prerequisites

Internal version: **v3.2.0.20**

Platform version compatibility: **HPFI 21.2.2.2**

Dependencies: **SySDigitalSolutionPackages v21.2.2000.zip** or higher

Insurance Apps Dependencies: **Core Policy AdminV3.2.0**

Upgrade instructions: install **Core Policy Admin Import Upgrade V3.2.0** instead

Install instructions: Follow the standard instruction on how to import a digital asset [here](#).

2 After installment configurations

If you already have a Core Policy Admin Import asset installed on your environment you will need to apply the appropriate upgrade asset so start by check the Upgrade from section before applying this package.

Installing Core Policy Admin Import Upgrade 3.2.0

1 Prerequisites




Platform version compatibility: **HPFI 21.2.2.2**.

Dependencies: **SySDigitalSolutionPackages v21.2.2000.zip** or higher

Insurance Apps Dependencies: **Core Policy Admin 3.2.0**

Upgrade from: **Core Policy Admin Import 1.14.9**

Upgrade instructions: before importing the digital asset, run the SQL scripts available in the **Upgrade** folder from the digital asset location.

-  BEFORE 10 FTOS_PA_PolicyCancellation.sql
-  BEFORE 20 Move Generate Policy to PolicyAdmin.sql
-  BEFORE 30 Insert Policy Admin Security Roles.sql

Install instructions: Follow the standard instruction on how to import a digital asset [here](#). After installing the digital solution, run the SQL script from **Policy Admin Import Upgrade** folder.

2 After installment configurations

If you already have a Core Policy AdminImport asset installed on your environment you will need to apply the appropriate upgrade asset so start by check the Upgrade from section before applying this package.

Glossary

D

DAUDD

Days after unpaid due date

DIDE

Direct debit

M

MTA

Mid Term Adjustments

P

PWDAY

Policy automatically withdraw day

U

UPR Reports

Unearned Premium Reports