



Guidewire InsuranceNowTM

Portal Development

Release 2021.1

© 2021 Guidewire Software, Inc.

For information about Guidewire trademarks, visit <http://guidewire.com/legal-notices>.

Guidewire Proprietary & Confidential — DO NOT DISTRIBUTE

Product Name: Guidewire InsuranceNow

Product Release: 2021.1

Document Name: Portal Development

Document Revision: 07-June-2021

Contents

| | |
|---|-----------|
| About documentation | 5 |
| Support | 5 |
| 1 Using the InsuranceNow V5 API | 7 |
| API Authentication | 9 |
| API Authorization | 10 |
| API Security Group | 11 |
| "Try out" the API V5 | 11 |
| Handling sensitive or internal data | 14 |
| Generate your own SDK | 15 |
| 2 Develop a Consumer Sales Portal | 17 |
| Consumer sales portal setup requirements | 17 |
| Consumer Sales Portal authentication | 18 |
| Configure Consumer Sales Portal authentication | 18 |
| Submit a quote to an agent. | 20 |
| Issue a policy using the Sales Portal | 21 |
| Manage access to quote | 24 |
| Refer quote to agent | 25 |
| Configure Consumer Sales Portal to refer quotes to agents | 25 |
| Email quote details and reminders | 26 |
| Configure Consumer Sales Portal to send quote emails | 26 |
| Quote email content | 27 |
| Consumer Sales Portal quote reminders | 28 |
| Configure the email reminder task | 28 |
| 3 Develop a Consumer Service Portal | 31 |
| Consumer Service Portal Authentication | 31 |
| Submit a claim | 32 |
| List claims and claim details | 34 |
| Add attachments in a service portal. | 34 |
| List policy and policy details | 36 |
| Change policy coverage | 37 |
| Make payments | 41 |
| 4 Develop an Agent Portal | 45 |
| Agent Portal authentication | 45 |
| Submit a quote for approval | 46 |
| View and delete tasks | 49 |
| Create a task on a policy or application | 50 |
| Create a note on a policy or application | 52 |
| View and delete notes | 55 |
| Work on tasks | 55 |
| Transfer a task | 57 |
| Endorse a policy | 58 |
| Cancel a policy | 60 |
| Renew a policy | 63 |
| Add attachments in an agent portal | 64 |

About documentation

Support

For assistance, visit the Guidewire Community.

Guidewire customers

<https://community.guidewire.com>

Guidewire partners

<https://partner.guidewire.com>

Using the InsuranceNow V5 API

The InsuranceNow API V5 enables the development of InsuranceNow portals. The InsuranceNow API V5 provides an interface to create quotes and applications, process payments, route quotes to agents, manage policies, and submit claims.

These REST API endpoints are available within the system through an interactive API interface. The interactive API uses the Swagger OpenAPI specification to provide a way to view the API endpoints and execute API requests.

InsuranceNow accepts API endpoints with the following format:

```
https://host:port/coreapi/version/endpoint
```

Here is an example API Request:

```
curl -X GET "https://xx.xx.xxx.xxx:xxxxx/coreapi/v5/customers?postalCode=95119" -H "accept: application/json" -H "authorization: Basic YWRtaW46OTk5OQ=="
```

Before InsuranceNow executes API requests, it validates the callers authentication and authorization to execute the request.

REST API Concepts

The API uses the Representational State Transfer (REST) architecture to issue requests and receive responses from the server.

Resources

Resources and sub-resources identify the objects that can be created, retrieved, and updated by the RESTful API. When using the API, data objects such as a quote, application, or customer are resources. Data objects such as a coverage or driver on a specific policy are considered sub-resources. `/applications/{systemId}` is an example of a resource and `/applications/{systemId}/drivers/{driverNumber}` is an example of a sub-resource. Using sub-resources is useful when you need access to certain details such as the list of coverages on a specific risk.

Methods

The API uses GET, PATCH, PUT, POST, and DELETE methods to create, read, update, and delete resources. Some methods are safe and some are idempotent. Safe methods do not modify resources. An idempotent method can be called multiple times and the result on the resource is the same. Methods that are safe are also idempotent.

GET

The GET method retrieves a specified resource or sub-resource for the endpoint it acts on. This method is safe and idempotent. The GET method does not change the resource that it retrieves and duplicate GET requests have no impact on the resource either.

PATCH

The PATCH method updates one or more field in a resource and updates the `_revision`. The body of the request only needs to include the fields to update. When you submit a PATCH request, the server compares the `_revision` of the resource you are updating with the `_revision` on the server prior to accepting the request. This method is not safe but it is idempotent.

Note: Some fields associated with a resource, such as `id`, cannot be changed by the PATCH method.

PUT

The PUT method replaces the specified resource or sub-resource and updates the `_revision`. The `_revision` prevents concurrent updates to the same revision. When you submit a PUT request, the server compares the `_revision` of the resource you are updating with the `_revision` on the server prior to accepting the request. This method is not safe but it is idempotent.

Note: Some fields associated with a resource, such as `id`, cannot be changed by the PUT method.

POST

The POST method creates a new resource. This method is not safe and it is not idempotent as a new resource is created each time you submit a POST method. The POST method returns the URL to the new resource in the Location header and the `_revision` in the body of the response. The URL of the resource includes the `systemId` which identifies the new resource.

DELETE

The DELETE method deletes a specified resources or sub-resource. This method is idempotent. If you submit the same request more than once, no action is taken as the resource would no longer be available. However, you may receive an error message as the result of trying to delete a resource more than once.

Full endpoints

For certain resources, the API provides an endpoint that interacts with the entire payload of that resource and an endpoint that interacts with the resource without including all of its sub-resources. API endpoints that end with `/full` interact with the entire resource which includes the sub-resources. For example, `PUT /applications/{systemId}` requires less payload than `PUT /applications/{systemId}/full`. The payload for `PUT /applications/{systemId}/full` needs to include content about sub-resources such as locations and lines.

Links

The body of the API response can include one or more `_links` sections. These links provide direct access to other related items you might want to access. For example, when you get the full application response there are many sets of links available in the response. The coverages section of the full application response includes links to the coverage, its parent, and the associated list of coverage items:

```
...
  "coverages": [
    {
      ...
      "_links": [
        {
          "rel": "self",
          "href": "/coreapi/v5/applications/883/lines/Homeowners/risks/1/coverages/CovA"
        },
        {
          "rel": "parent",
          "href": "/coreapi/v5/applications/883/lines/Homeowners/risks/1"
        },
        {
          "rel": "coverageItems",
          "href": "/coreapi/v5/applications/883/lines/Homeowners/risks/1/coverages/CovA/coverageItems"
        }
      ]
    }
  ]
```


API Authentication

In general, all API callers must authenticate with the server and pass authorization requirements before the API request is executed by the server.

Note: The `/clients/{clientId}/sessions` API does not require authentication or authorization as it is used to initiate a session from an anonymous user.

Authenticate from Interactive API

When you use the interactive API to submit API requests to the server, the authentication details for each API request are populated by the interactive tool.

When using the interactive API, authentication works in the following way:

1. A user with the **REST API Documentation View** and the **Administrator** role accesses the interactive API.
2. The user enters authorization details in the **Available Authorizations** dialog.

Users have the option to authorize with the following options:

Basic authentication

A valid InsuranceNow username and password provide authentication.

JWT

An Okta-provided JWT provides authentication.

OAuth2

An internally configured `swagger-api` client is used to obtain a JWT. In this case, InsuranceNow is the identity provider.

Note: From the interactive API, the OAuth2 option must only be used for testing purposes.

The authentication details apply to all API calls submitted from the interactive API for the current session unless the user selects a different authentication method.

3. The user selects an API to try, enters the required parameters, and executes the API request. Before the request is executed by the server, the server validates the authorization details.

Authenticate from a Consumer Sales Portal

The Consumer Sales Portal uses a JWT from Okta to authenticate with InsuranceNow. When a consumer sales portal submits API requests to the server, the following steps occur:

When a consumer service portal user has not logged in:

1. Portal submits the `POST /clients/{clientId}/sessions` API endpoint.
2. InsuranceNow requests an Okta-issued JWT through the Guidewire Hub.
3. As requests from the Consumer Sales Portal are anonymous, Okta generates a JWT for the `directportal` user.
4. InsuranceNow includes the Okta-issued JWT in the API response.
5. The portal includes the Okta-issued JWT in HTTP authorization header of each API request for that session.
6. InsuranceNow validates the Okta-issued JWT prior to executing each API request. For more information, see “Manage access to quote” on page 24.

When a consumer service portal user has logged in:

1. Okta generates a JWT upon user login.
2. The portal included the Okta-issued JWT in HTTP authorization header of each API request for that session.
3. InsuranceNow validates the Okta-issued JWT prior to executing each API request.

For the steps to configure Consumer Sales Portal authentication, see “Consumer Sales Portal authentication” on page 18

Authenticate from an Agent Portal

API requests from an Agent Portal authenticate with the server using OAuth2. When the Agent Portal submits API requests to the server, the following steps occur:

1. The Agent Portal uses OAuth2 to authenticate with the server as the InsuranceNow agent user.
2. The identity provider generates a JWT for the InsuranceNow agent user using OAuth2.
3. The Agent Portal includes the JWT in the HTTP authorization header of each API request.

For the steps to configure Agent Portal authentication, see “Agent Portal authentication ” on page 45.

Authenticate from a Consumer Service Portal

The Consumer Service Portal uses a JWT from Okta to authenticate with InsuranceNow. JWT client authentication requires that InsuranceNow is configured with the required settings and credentials to request and decode a JWT from Okta.

There are 2 modes when a consumer service portal submits API requests to the server:

1. When a consumer service portal user has not logged in:
 - a. Portal submits the POST /clients/{clientId}/sessions API endpoint.
 - b. Portal submits the POST /clients/{clientId}/sessions API endpoint.
 - c. InsuranceNow requests an Okta-issued JWT through the Guidewire Hub.
 - d. Okta generates a JWT for the portal user.
 - e. InsuranceNow includes the Okta-issued JWT in the API response.
 - f. The portal includes the Okta-issued JWT in HTTP authorization header of each API request for that session.
 - g. InsuranceNow validates the Okta-issued JWT prior to executing each API request.
2. When a consumer service portal user has logged in:
 - a. Okta generates a JWT upon user login.
 - b. The portal included the Okta-issued JWT in HTTP authorization header of each API request for that session.
 - c. InsuranceNow validates the Okta-issued JWT prior to executing each API request.

For the steps to configure Consumer Service Portal authentication, see the *Consumer Service Portal Configuration Guide*.

API Authorization

Authorization to execute APIs can be based on the authentication method and user authority. The server allows clients to submit API calls to the APIs associated with the API group or list of API groups defined for the user's APISecurityGroups authority.

Authorization from InsuranceNow Interactive API

When the InsuranceNow interactive API authenticates with basic authentication, the user can submit API calls to the APIs associated with the API groups defined by user's APISecurityGroups authority. In this case, the user is based on the credentials provided to the Basic Authentication option.

When the InsuranceNow interactive API authenticates with JWT, the server allows the client to submit API calls to the APIs associated with the API groups defined by user's APISecurityGroups authority. By default, the JWT is created for the **DirectPortal** user which has access to API provided to the DirectSalesPortalGroup security group.

When the InsuranceNow interactive API authenticates with OAuth2, the server verifies the user's APISecurityGroups authority setting before executing the API request. In this case, the user is the user that logged into InsuranceNow.

Authorization from Consumer Sales Portals

Consumer Sales Portals use JWT client authentication to submit API requests. By default, Consumer Sales Portal accesses the API as the **DirectPortal** user which has access to APIs provided to the DirectSalesPortalGroup security group.

Authorization from Consumer Service Portals

Consumer Service Portals use JWT client authentication to submit API requests. By default, the Consumer Service Portal accesses the API as the Service Portal web portal user which has access to APIs provided to the ServicePortalGroup.

Authorization from Agent Portals

Agent Portal uses OAuth 2 authentication to submit API requests. The Agent Portal accesses the APIs as the InsuranceNow agent user that logged into the Agent Portal. By default, InsuranceNow agent users that access the Agent Portal have access to API provided to the AgentPortalGroup security group.

API Security Group

InsuranceNow uses API security groups to determine which API endpoints a user has authorization to execute. The following API security groups are available:

DirectSalesPortalGroup

Provides access to APIs required by the consumer sales portal.

AgentPortalGroup

Provides access to APIs required by the agent portal.

DataServiceGroup

Provides access to APIs required by data services.

ServicePortalGroup

Provides access to APIs required by the consumer service portal.

"Try out" the API V5

The interactive API includes an option to "try out" the API. The **Try it out** option executes the API requests against the environment that you access. To perform test operations, use the **Try it out** option on the development or quality assurance environments.

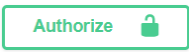
Before you begin

- Only users with the **REST API Documentation View** authority and the **Administrator** role can view the interactive API.
- Determine which authentication method to use in order to test the user authorization. See "API Authentication" on page 9.

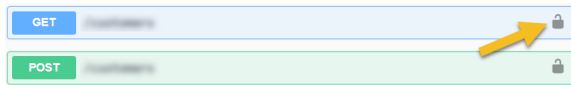
About this task

To "try out" an API request, you enter authentication information, enter request parameters, execute the request, and then view the response.

Procedure

1. Select **Admin**→**System Tools**→**REST API Documentation**
2. On the **REST API Documentation** page, select **core_v5** in the **Select API and version** field. Then, click **Open**.
3. Above the APIs, click  to provide authorization to submit requests.

Note: As an alternative, you can select the icon that looks like a padlock which appears for each endpoint:



4. For JWT authentication, perform the following steps in the **JWT (apiKey)** section:

JWT (apiKey)

JWT Authorization header using the Bearer scheme.
 Note, the header value must begin with **Bearer**, followed by a space, followed by the JWT. For example: **Bearer ey30...cQ6w**
 Name: Authorization
 In: header
 Value:

Authorize

Close

- a. In the **Value** field, enter Bearer followed by the JWT. For example, Bearer ey30...cQ6w.
 - b. Click **Authorize**.
5. For Basic authentication, perform the following steps in **Basic authorization** section of the **Available authorizations** dialog.

Basic authorization

Username:

Password:

Authorize

Close

- a. Enter the **Username** and **Password** that you want to use to authenticate with InsuranceNow.
 - b. Click **Authorize**.
6. For OAuth2, perform the following steps in the **oauth2(OAuth, accessCode)** section of the **Available authorizations** dialog.

oauth2 (OAuth2, accessCode)

Authorization URL: /oauth/authorize

Token URL: /oauth/token

Flow: accessCode

client_id:

client_secret:

Authorize

Close

- a. Retain the default setting for **client_id**.
- b. Leave the **client_secret** empty.
- c. Click **Authorize**.

Note: Ignore the information about scopes as it does not apply when accessing the API. The details provided about the Authorization URL, Token URL, and Flow are used by the interactive API and are listed for informational purposes only.

7. Click **Close** to close the **Available authorizations** dialog.
8. Click the endpoint that you want to try.
9. In the **Parameters** section, click the **Try it out** button.

Customers Create, retrieve and update customers and their information

GET /customers

List Customers. Requires at least one of the following parameters { 'indexName', 'taxid', 'contactNumber', 'emailAddress', 'applicationRef', 'createdSinceDate', 'recentlyViewed=true' }

Parameters

Try it out

10. In the **Parameters** section, any required parameters and then click **Execute**.

Customers Create, retrieve and update customers and their information

GET /customers

List Customers. Requires at least one of the following parameters { 'indexName', 'taxid', 'contactNumber', 'emailAddress', 'applicationRef', 'createdSinceDate', 'recentlyViewed=true' }

Parameters

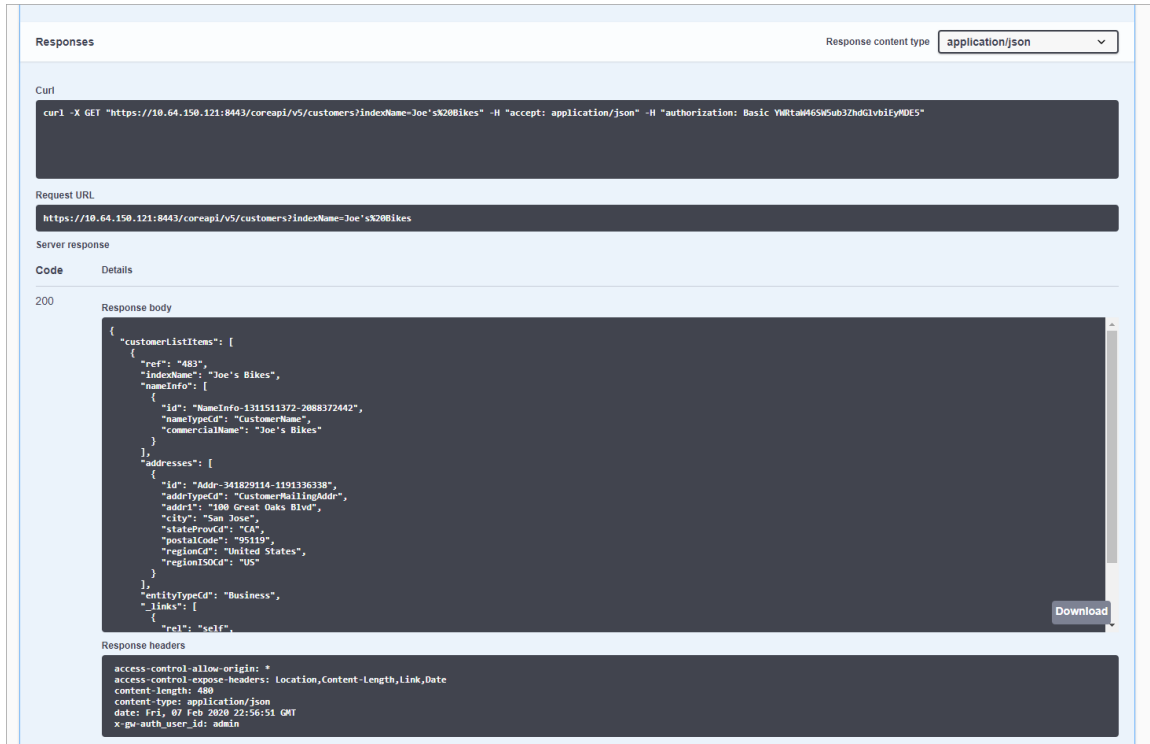
Cancel

| Name | Description |
|-------------------------------------|---|
| indexName string (query) | Select customers where IndexName or DBAIndexName is LIKE 'indexName%' Joe's Bikes |
| taxid string (query) | Select customers where Taxid, TaxidTrust, TaxidEstate, SSN or SSNJoint EQUALS 'taxid' taxid - Select customers where Taxid, TaxidT |
| address string (query) | Select customers where LookupAddress is LIKE 'address%' address - Select customers where LookupAd |
| city string (query) | Select customers where MailCity, BillCity or LookupCity is LIKE 'city%' city - Select customers where MailCity, BillCity |
| optionalFields string (query) | Comma-delimited list of optional fields to be included. Currently supports policyCount and openTaskCount. optionalFields - Comma-delimited list of optio |

Execute Clear

Result

The API response displays in the **Responses** section.



Responses Response content type: application/json

Curl

```
curl -X GET "https://10.64.150.121:8443/coreapi/v5/customers?indexName=Joe's%20Bikes" -H "accept: application/json" -H "authorization: Basic YWRtaW46S200Bikes"
```

Request URL

```
https://10.64.150.121:8443/coreapi/v5/customers?indexName=Joe's%20Bikes
```

Server response

| Code | Details |
|------|--|
| 200 | <p>Response body</p> <pre>{ "customerListItems": [{ "ref": "483", "indexName": "Joe's Bikes", "nameInfo": { "id": "NameInfo-1311511372-2088372442", "nameTypeCd": "CustomerName", "commercialName": "Joe's Bikes" } }], "addresses": [{ "id": "Addr-381829114-1191336338", "addressTypeCd": "CustomerMailingAddr", "addr1": "100 Great Oaks Blvd", "city": "San Jose", "stateProvCd": "CA", "postalCode": "95119", "regionCd": "United States", "regionISOcd": "US" }], "entityTypeCd": "Business", "_links": { "rel": "self" } }</pre> <p>Response headers</p> <pre>access-control-allow-origin: * access-control-expose-headers: Location,Content-Length,Link,Date content-length: 400 content-type: application/json date: Fri, 07 Feb 2020 22:56:51 GMT x-gw-auth_user_id: admin</pre> |

Server response codes 200 and 201 indicate a successful API call.

Next steps

If you want to try an API using a different authentication method, open the **Available Authorizations** dialog again and select **Logout**. Then, you can enter details for a different authentication method.

Handling sensitive or internal data

The API handles sensitive data and internal coderefs values in the following ways:

Sensitive data in API responses

In API responses, the system masks the following types of data with asterisks:

- Tax ID
- Bank account number
- Customer login credentials
- Insurance score
- Debit card number
- Credit card security code
- Credit card expiry year
- Credit card expiry month
- Credit card number

Sensitive data in API requests

API requests submit un-masked sensitive data when there is requirement to add or update the data. When updating a resource with a PUT or PATCH request, the API request can leave masked values as-is unless the request includes an update to the masked the value.

coderefs in API response

API requests that begin with GET /coderef only return coderefs that are needed to issue and manage policies. For example, API responses can include coderefs that define lists of valid values for a field. However, the API

will not return coderefs related to third-party integrations or data related to integrations between InsuranceNow modules.

Note: The `coderef-exclusion.xml` file includes the list of coderefs that the API excludes from API responses. You can add additional coderefs to the exclusion list by overriding the `coderef-exclusion.xml` (`coreAPI/src/main/resources/com/guidewire/insurancenow/api/impl/core_v5/model/coderef/coderef-exclusion.xml`) file in your build-out and registering the override file with the APIV5 namespace and the `coderef-exclusion` repository.

Generate your own SDK

You can use tools like SwaggerHub or Swagger Codegen to generate your own client SDK for the InsuranceNow API in Javascript, Java, and other programming languages. You can use the wrapper classes in the resulting client SDK to call the InsuranceNow API from another application and not have to deal with HTTP requests and responses.

But if you generated your own client SDK based on the InsuranceNow Swagger API specification, you may need to regenerate your SDK when the Swagger specification is updated due to name or structure changes.

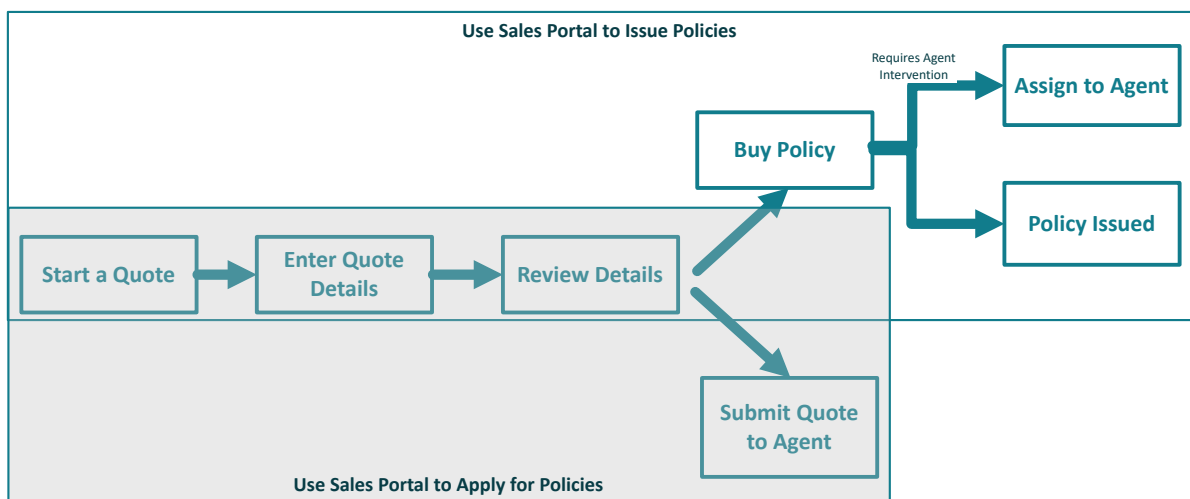
Develop a Consumer Sales Portal

Developers can use the v5 API to develop and configure a Consumer Sales Portal that issues a policy or submits a quote to an agent. In the build-out, attribute metadata is defined for the data objects associated with each product. When developing portal pages, you can refer to the product's attribute metadata to build each page with various fields and then use the API to process the data entered in those fields. For more information on attribute metadata, see *Product Attribute Reference*.

This section includes information about the Consumer Sales Portal API flows and related functionality. However, the interactive API includes all the parameters available for each API. For information on how to view and test the API and its parameters, see ““Try out” the API V5” on page 11.

Consumer Sales Portal API flows

While the business flow for each Consumer Sales Portal (Sales Portal) varies based on business needs, typically developers configure the Sales Portal to either issue policies or to submit policy applications.



Note: Many variations to the business flow are possible. For example, based on the requirements of the Sales Portal, assigning the quote or application to an agent can happen at any time.

Consumer sales portal setup requirements

The following setup is required before consumers can access the Consumer Sales Portal:

Register the Consumer Sales Portal with Guidewire Identity Federation Hub

Contact the Guidewire Cloud team to register the portal with the Guidewire Identity Federation Hub (Guidewire Hub). The Guidewire Hub enables the Consumer Sales Portal to be issued a JWT from Okta.

Configure the Consumer Sales Portal to authenticate with InsuranceNow

InsuranceNow must be configured to request and decode JWTs from Okta. See the *Configuration Guide* for the steps to configure Consumer Sales Portal Authentication.

Configure the Consumer Sales Portal to use Google Places for address completion

Specify the Google API key in the `GOOGLE_API` parameter of the `webpack.prod.js` and `webpack.dev.js` files. It is recommended that the Sales Portal does not use the same Google API key as InsuranceNow core.

Configure the `directportal` user with the authority attributes required to issue a policy for the products available in the Consumer Sales Portal

If the `directportal` user does not have the authority to perform the steps required to issue a policy, policy issuance will not succeed in the Consumer Sales Portal.

Consumer Sales Portal authentication

The Consumer Sales Portal uses a JWT from Okta to authenticate with InsuranceNow. JWT client authentication requires that the Guidewire Cloud team registers the application with the Guidewire Identity Federation Hub and that InsuranceNow is configured with the required settings and credentials to request and receive a JWT from Okta.

The following steps describe how consumer service portals authenticate with InsuranceNow:

1. Portal submits the `POST /clients/{clientId}/sessions` API endpoint.
2. InsuranceNow requests an Okta-issued JWT token
3. InsuranceNow includes the Okta-issued JWT token in the API response.
4. The portal includes the Okta-issued JWT token with each subsequent API request for that session.
5. InsuranceNow validates the Okta-issued JWT token prior to executing each API request.

Configure Consumer Sales Portal authentication

Configure the local and deployment-specific settings for Consumer Sales Portal authentication.

Before you begin

Contact Guidewire Cloud team to register the application with Guidewire Identity Federation Hub (Guidewire Hub).

About this task

The JWT client settings provides the information used by the client to access the InsuranceNow API. The Okta HTTP client settings provide InsuranceNow the details required to request and decode the Okta-issued JWT.

Procedure

Configure the local JWT client settings

1. Configure the following parameters in `web/APP-INF/mda/jwt/jwt-client-settings.xml` to provide details for the JWT token request:

| Parameter | Description |
|--------------|---|
| CustomerId | The name of the application used within InsuranceNow. For example, <code>ConsumerSalesPortal</code> . |
| ClientId | Client identifier provided by the Guidewire Cloud team. This ID is used to communicate with Guidewire Hub). |
| ClientSecret | Client secret provided by the Guidewire Cloud team. This secret provides authorization for the client associated with the <code>ClientId</code> . |

| Parameter | Description |
|-------------|---|
| IN_UserName | The user name associated with the JWT token. In general, the DirectPortal user is used for consumer portal access. |

Configure the local Okta HTTP client settings

2. Configure the following parameters in `web/APP-INF/mda/jwt/okta-httpclient-settings.xml`:

| Parameter | Description |
|---------------|--|
| ClientId | Client identifier provided by the Guidewire Cloud team. This information is provided in the application registration response. |
| Grant_Type | Enter <code>CLIENT_CREDENTIALS</code> . |
| Scope | The scope associated with the application. For example, <code>inow-consumerportal-api.consumer</code> . |
| AuthServerUri | The authorization server to request and decode JWT tokens. This information is provided in the application registration response. |
| Content-Type | Enter <code>application/x-www-form-urlencoded</code> to identify the API content type. |
| JWKUrl | Endpoint URL to fetch the public key. The public key is used to decode the JWT token. The format is <code><AuthServerUri>/v1/keys</code> . |
| Issuer | URL that identifies the issuer of the token. |

Configure deployment-specific settings

3. For each deployment environment, configure the following parameters in the **JWT client** section of `config.properties`:

| Parameter | Description |
|-------------------|---|
| JWT_CUSTOMER_ID | The name of the application used within InsuranceNow. For example, <code>ConsumerSalesPortal</code> . |
| JWT_CLIENT_ID | Client identifier provided by the Guidewire Cloud team. This ID is used to communicate with Guidewire Hub. |
| JWT_CLIENT_SECRET | Client secret provided by the Guidewire Cloud team. This secret provides authorization for the client associated with the ClientId. |
| JWT_IN_USERNAME | The user name associated with the JWT token. In general, the DirectPortal user is used for consumer portal access. |

4. For each deployment environment, configure the following parameters in the **OKTA client settings** section of `config.properties`:

| Parameter | Description |
|----------------------|--|
| OKTA_CLIENT_ID | Client identifier provided by the Guidewire Cloud team. This information is provided in the application registration response. |
| OKTA_GRANT_TYPE | Enter <code>CLIENT_CREDENTIALS</code> . |
| OKTA_SCOPE | The scope associated with the application. For example, <code>inow-consumerportal-api.consumer</code> . |
| OKTA_AUTH_SERVER_URI | The authorization server to request and decode JWT tokens. This information is provided in the application registration response. |
| OKTA_CONTENT_TYPE | Enter <code>application/x-www-form-urlencoded</code> to identify the API content type. |
| OKTA_JWK_URL | Endpoint URL to fetch the public key. The public key is used to decode the JWT token. The format is <code><AuthServerUri>/v1/keys</code> . |
| OKTA_ISSUER | URL that identifies the issuer of the token. |

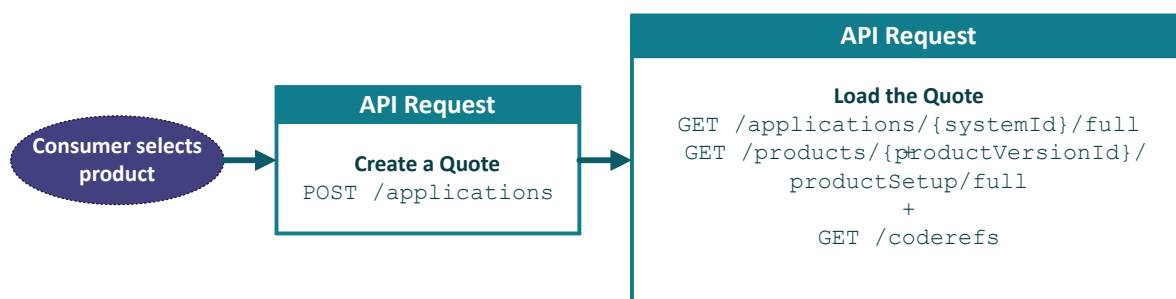
Submit a quote to an agent

While the business flow for each Sales Portal varies based on business needs, the following API flow applies to most implementations:

1. Start a quote.
2. Enter quote details.
3. Submit quote to an agent

Start a quote

The process required to start a quote includes the consumer selecting a product and then sending API requests to create a quote, access the quote, and load the details required to start the quote.



Note: The complete URL of the application is available in the Location header of the POST /applications API response. The URL of the application includes the `systemId`.

Enter quote details

The process required to enter quote details is an iterative process where the consumer enters quote details on various tabs or pages, resolves any errors, and then has the opportunity to apply for the policy. The product setup defines the quote details that the consumer must enter to complete a quote.

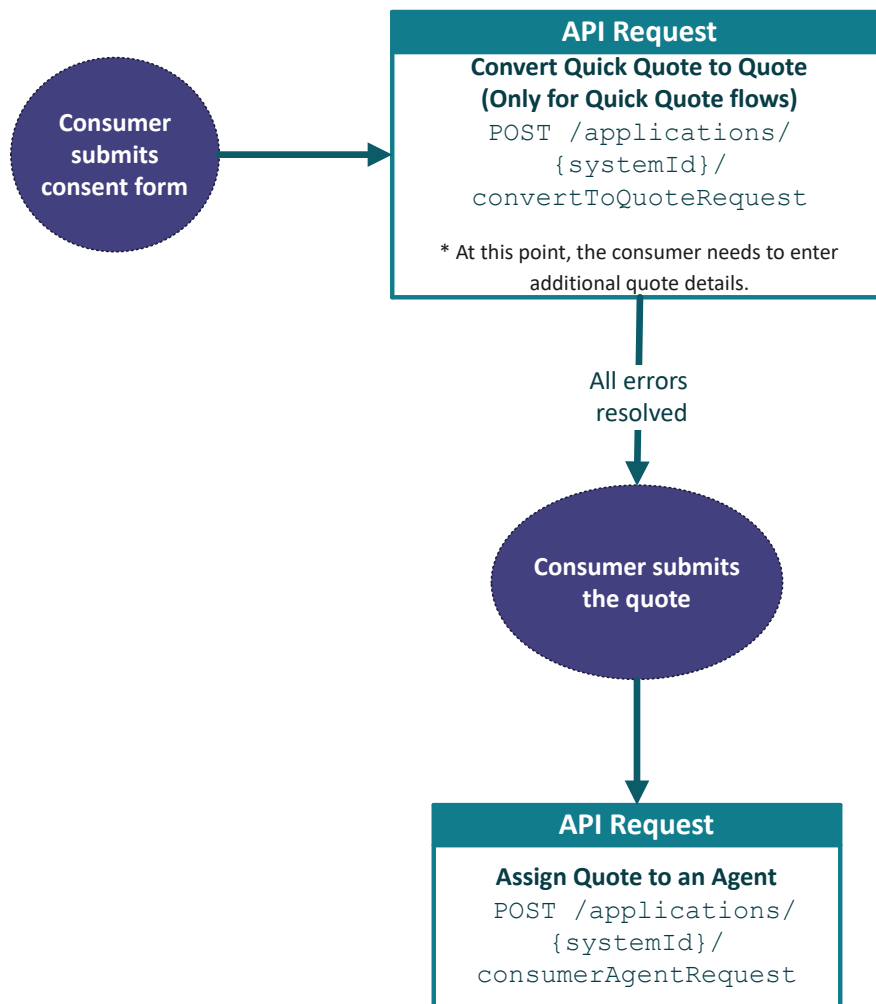
Each time the portal loads or saves the quote as part of a step or similar action, the portal submits a PUT request with the latest `_revision` of this quote.

Note: To save the quote, the revision number is required.



Submit quote to an agent

The process required to submit a quote to an agent requires that the consumer accepts the conditions on the consent form. Then for quick quotes, an API request is needed to convert the quick quote to a quote. Finally, the consumer submits the quote for further processing and the portal sends an API request to assign the quote to an agent.

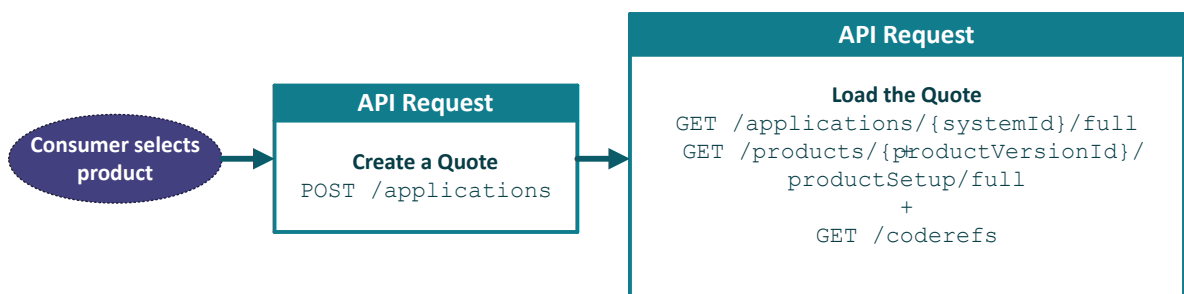


Issue a policy using the Sales Portal

While the business flow for each Sales Portal varies based on business needs, the following API flow applies to most implementations:

Start a quote

The process required to start a quote includes the consumer selecting a product and then sending API requests to create a quote, access the quote, and load the details required to start the quote.



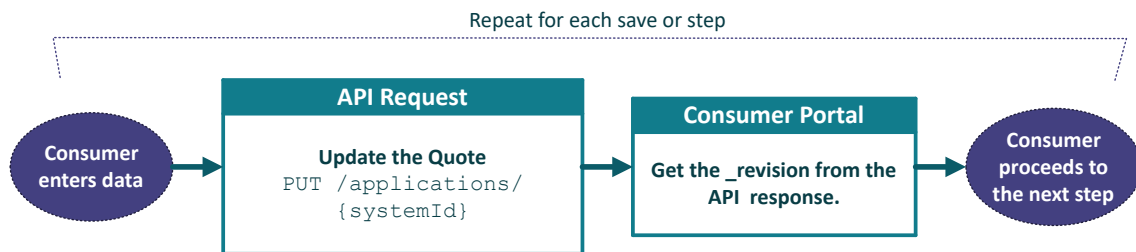
Note: The complete URL of the application is available in the Location header of the POST / applications API response. The URL of the application includes the `systemId`.

Enter quote details

The process required to enter quote details is an iterative process where the consumer enters quote details on various tabs or pages, resolves any errors, and then has the opportunity to apply for the policy. The product setup defines the quote details that the consumer must enter to complete a quote.

Each time the portal loads or saves the quote as part of a step or similar action, the portal submits a PUT request with the latest `_revision` of this quote.

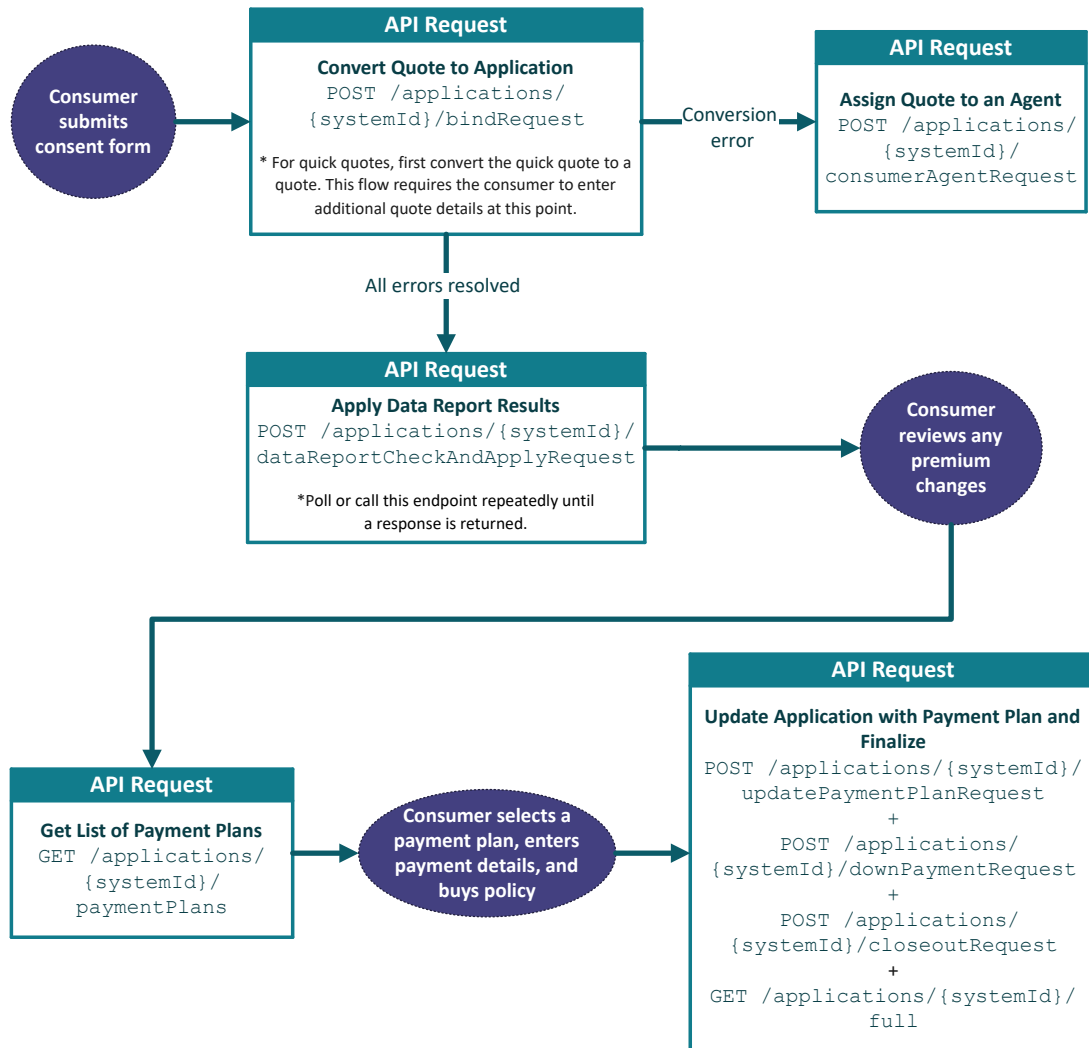
Note: To save the quote, the revision number is required.



Buy the policy

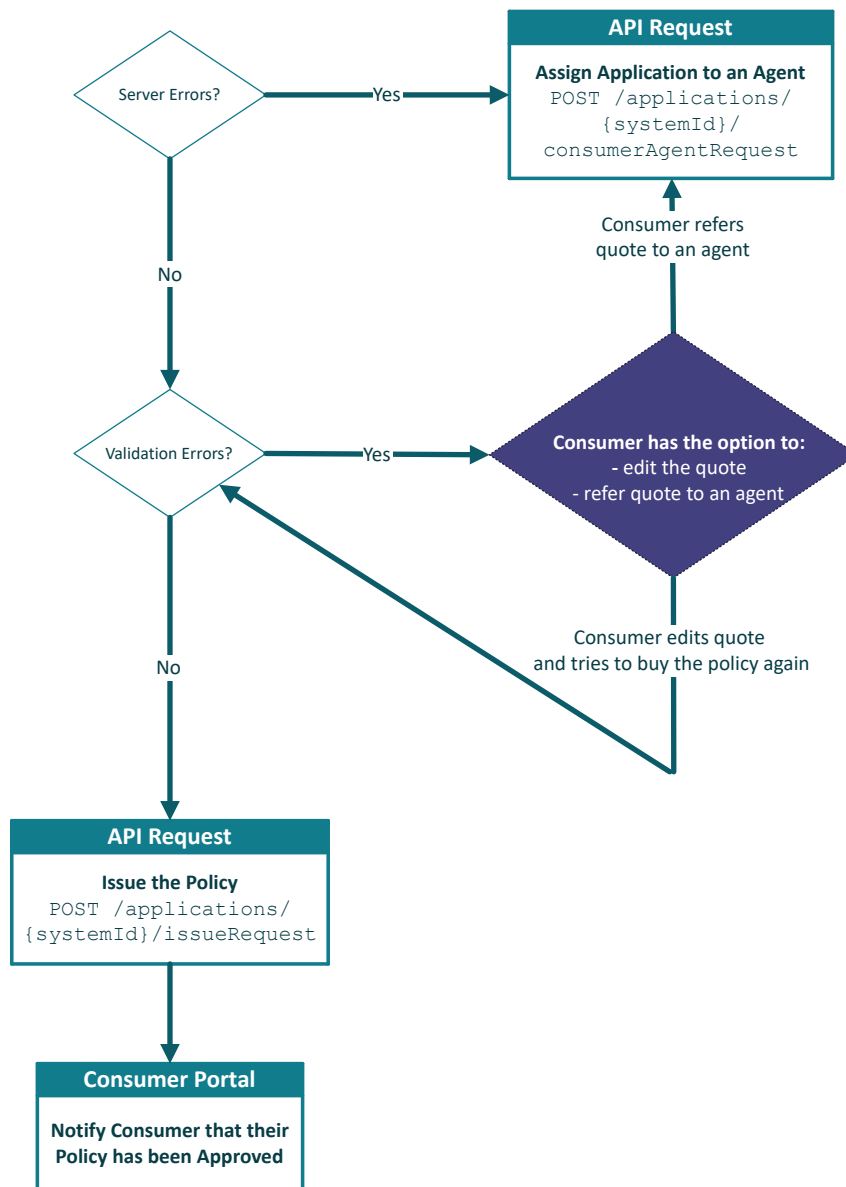
The process required to buy a policy includes API requests to convert the quote to an application, apply data report results, list the payment plans, and update the application with payment detail.

Note: The results of the data reports may trigger a recalculation of premium.



Issue policy or refer quote to agent

The process to issue a policy includes either an API request to issue the policy or assign the application to an agent.



Manage access to quote

The Consumer Sales Portal uses JWT to manage accesses to quotes:

- Whenever a consumer starts a quote, a JWT is generated and associated with the quote and the consumer portal session. The portal calls the POST /clients/{clientId}/sessions endpoint to create a session and a JWT for that session is returned. When the new quote is created in InsuranceNow, a decoded identifier in JWT is added to the BasicPolicy.SessionLink field of the quote.
- When a consumer attempts to retrieve previously created quote, a JWT is generated for the new session and then the portal requests the **Quote Number** and the **Zip Code** associated with the quote. The portal calls POST /clients/{clientId}/sessions to create a new session and then it calls PUT /applications/updateSessionLinkRequest to validate the request. After InsuranceNow validates the request, it updates the BasicPolicy.SessionLink field of the quote with the identifier of the latest JWT.

Note: The JWT has a timeout that is set by Guidewire Cloud Services. When the timeout associated with the JWT has expired, the portal returns the consumer to the landing page where the consumer can either start a new quote or retrieve an existing quote.

Refer quote to agent

When a Consumer Sales Portal calls the `/applications/{resourceId}/consumerAgentRequest` API, the system runs the `PostBind` rule to create a task. The Consumer Sales Portal refers a quote to an agent when the consumer requests it or when the policy process requires agent assistance.

Note: Referring a quote to an agent only succeeds when the quote includes the email address or the phone number of the consumer.

By default, the `PostBind` rule creates a `Sales Portal Application Followup (ApplicationTask2017)` task. This task appears in the Inbox of each agent in the `UWClerk` group. The system deletes the task after an agent starts working on the quote.

Consumers cannot view or edit quotes that have been referred to an agent. For more information about the Sales Portal behavior, see the *Consumer Sales Portal Features*.

Configure Consumer Sales Portal to refer quotes to agents

To enable the Consumer Sales Portal to refer quotes to agents, verify that your product and Consumer Sales Portal files include the required configuration.

Procedure

1. For each product, add or verify that the `PostBind` rule is defined in the `cw` (country-wide) or state-specific product rule.xml file.

```
...
<ProductRule id="id" RuleTypeCd="PostBind" Processor="java"
  RuleSource="com.iscs.start.uw.common.product.model.template.iic.personalauto.cw.v01_00_00.rule.PostBind" />
...
```

2. For each product, add or verify that the `PostBind` action is defined as part of the `UWExternalQuoteBind` sequence in the `cw` (country-wide) or state-specific product services.xml file.

```
...
<sequence name='UWExternalQuoteBind'>
  <action type='service' name='UWExternalQuoteBindToApp' />
  <action type='link' name='UWUnderwrite' />
  <action type='service' name='UWRuleHandler' param='PostBind' /> <!-- Create fail-safe followup task. This
Task does not survive Policy Issuance -->
  <action type='service' name='UWRuleHandler' param='Approval' />
  <action type='service' name='UWRuleHandler' param='PreApprovalApply' />
  <actionparam Name='RunIfPriorRuleErrors' Value='Yes' />
</action>
  <action type='service' name='UWApplicationCalculateWritten' />
  <action type='service' name='UWReinsuranceProcess' />
  <action type='service' name='UWRuleTasks' />
</sequence>
...
```

3. If you do not have an `application task.xml` file in your build-out, create a copy of the `uw/app/model/template/task.xml` and register the copy with the `UWApplication` package namespace and the `task-template` repository.
4. Open the `uw/app/model/template/task.xml` in your build-out, and verify that the task template includes `ApplicationTask2017`.

```
...
<TaskTemplate
  id='ApplicationTask2017'
  Name='Sales Portal Application Followup'
  Priority='4'
  PriorityOverrideInd='Yes'
```

```

Description='Followup on UnIssued Sales Portal Application for $!InsuredName.gets("CommercialName")'
DescriptionOverrideInd='Yes'
DefaultOwner='UWClerk'
DefaultOwnerCd='Group'
DefaultOwnerOverrideInd='Yes'
WorkDtDays='2'
WorkDtOverrideInd='Yes'
Text='Followup on UnIssued Sales Portal Application for $!InsuredName.gets("CommercialName")'
TextOverrideInd='Yes'
CriticalDtDays=''
CriticalDtBusinessInd=''
CriticalDtOverrideInd='Yes'
CriticalDtInd='No'
ReminderDtDays=''
ReminderDtBusinessInd=''
ReminderDtOverrideInd='Yes'
ReportTo=''
ReportToCd=''
ReportToOverrideInd='Yes'
RepeatingInd=''
RepeatingOverrideInd='Yes'
WorkRequiredInd='Yes'
DeactivationDt='29991231'
CompleteRequiredInd='Yes'
TaskTypeCd='Automatic'
<TaskRule id="application-task-process-delete" RuleTypeCd="Service" Processor="velocity"
RuleSource="UWRule::rule::none::application-task-process-delete.vtl" />
</TaskTemplate>
...

```

5. For each product, verify that the quote requires either an email address or a phone number from the consumer.

Email quote details and reminders

When a Consumer Sales Portal calls the `/applications/{resourceId}/consumerEmailRequest` API, the system performs the following steps:

1. It generates a request to send an email with quote details to the email address on the specified quote.
2. It uses the Email Consumer Portal Quote template to generate the first email that is sent to the consumer.
3. The system uses the Consumer Portal Quote Reminder task (ApplicationTask2035), a reserved task, to trigger the email reminder. The task template includes configuration for the timing of the reminder and how many reminders the system sends.
4. The system uses the Email Consumer Portal Quote Reminder template to generate the email that reminds the consumer of their open or incomplete quote.

Configure Consumer Sales Portal to send quote emails

If you want to send emails to consumers about the quotes that they requested or to remind them about an unfinished quote, verify that your product and Consumer Sales Portal files include the required configuration.

Procedure

1. For each product, add or verify the consumer-portal coderef is defined in the cw (country-wide) or state-specific product list.xml file.

```

...
<coderef name="consumer-portal">
  <options key="email-template">
    <option value="ConsumerEmailQuote" name="Quote" />
    <option value="ConsumerEmailQuoteReminder" name="Reminder" />
  </options>
</coderef>

```

2. For each product, add or verify that the APIConsumerEmail-Reminder rule is defined in the cw (country-wide) or state-specific product rule.xml file.

```

...
<!-- Rules related to Consumer Portal -->

```

```
<ProductRule id="id" RuleTypeCd="APIConsumerEmail-Reminder" Processor="java"
RuleSource="com.iscs.uw.api.rule.ConsumerPortalQuoteReminderTask" />
</ProductRules>
```

3. For each product, add or verify that the APIConsumerEmail sequence is defined in the cw (country-wide) product services.xml file.

```
...
<sequence name="APIConsumerEmail">
  <action type="service" name="UWRuleHandler" param="APIConsumerEmail-Reminder" />
</sequence>
</sequences>
```

4. Configure the Consumer Sales Portal URL in the following files:

- a. In web/APP-INF/mda/template/api-settings.xml, configure the local ConsumerPortalURL.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  ConsumerPortalURL = URL to consumer portal
-->
<APISettings ConsumerPortalURL="portalurl" >
</APISettings>
```

The ConsumerPortalURL setting is used by the email templates to provide the consumer a link to the Consumer Sales Portal.

- b. In the scripts/config.properties for each deployment environment, configure the CONSUMER_PORTAL_URL.

```
...
#API Settings
CONSUMER_PORTAL_URL="portalurl"
```

Next steps

While default email templates are available in the base configuration, you can configure the content of the email.

Quote email content

The cw product list.xml file includes the default list of consumer email templates. However, you can configure the email templates to be product-specific.

List of email templates for a product

The list.xml defines the templates that will be used for the product version

```
...
<coderef name="consumer-portal">
  <options key="email-template">
    <option value="ConsumerEmailQuote" name="Quote" />
    <option value="ConsumerEmailQuoteReminder" name="Reminder" />
  </options>
</coderef>
```

Files that generate email content

A combination of the following files generate the email content.

email templates (.vtl)

The body text of the quote email is defined in the consumer-email-quote.vtl. The body text of the quote reminder email is defined in the consumer-email-quote-reminder.vtl.

Note: You can configure these VTL files to include the following objects: Application, portalURL, and retrieveQuoteURL.

email.xml

The Subject, SenderName, and SenderAddress can be configured in email.xml file. The settings in this file can be product-specific.

api-settings.xml and config.properties

The Consumer Sales Portal URL which can be included in the email is defined in `api-settings.xml`. During deployment, the Consumer Sales Portal URL setting in `config.properties` overrides the setting in `api-settings.xml`.

Consumer Sales Portal quote reminders

After the `/applications/{resourceId}/consumerEmailRequest` API triggers an email to the consumer, the Consumer Portal Quote Reminder task (`ApplicationTask2035`) is created in the system. This is a system-wide task that cannot be customized per product.

By default, the Consumer Portal Quote Reminder task sends the consumer a email reminder 10 days after the task is created. After a reminder email is sent, another Consumer Portal Quote Reminder task is generated unless the consumer has already received 3 reminder emails.

You can configure the number of days between reminders and the number of reminder emails. These settings are system-wide therefore cannot be configured differently for each product or product version.

Configure the email reminder task

You can configure when a reminder email is sent and how many reminder emails are sent to the consumer that initiated the quote.

About this task

The email reminder settings are system-wide. They cannot be configured to be product-specific.

Procedure

1. If you do not already have an `application task.xml` file in your build-out, create a copy of the `\uw\app\model\template\task.xml` and register the new task template with the `UWApplication` package namespace and the task-template repository.
2. Open the `uw\app\model\template\task.xml` in your build-out, and verify that the task template includes `ApplicationTask2035`.

```
...
<TaskTemplate
  id='ApplicationTask2035'
  Name='Consumer Portal Quote Reminder'
  Priority='1'
  PriorityOverrideInd='No'
  Description='Send a Consumer Portal quote reminder email'
  DescriptionOverrideInd='No'
  DefaultOwner='System'
  DefaultOwnerCd='User'
  DefaultOwnerOverrideInd='No'
  WorkDtDays='10'
  WorkDtOverrideInd='No'
  Text='Send a Consumer Portal quote reminder email'
  TextOverrideInd='No'
  CriticalDtDays=''
  CriticalDtBusinessInd='No'
  CriticalDtOverrideInd='No'
  CriticalDtInd='No'
  ReminderDtDays='0'
  ReminderDtBusinessInd=''
  ReminderDtOverrideInd='No'
  ReportTo=''
  ReportToCd=''
  ReportToOverrideInd='No'
  RepeatingInd=''
  RepeatingOverrideInd='No'
  WorkRequiredInd='Yes'
  DeactivationDt='29991231'
  CompleteRequiredInd='No'
  TaskTypeCd='Automatic'
  Category=''>
<TaskRule id="consumer-portal-quote-reminder-email" RuleTypeCd="System" Processor="java"
```

```
        RuleSource="API::ConsumerPortalQuoteReminderEmail::none::none">
        <Param Name="MaxReminders" Value="3" />
    </TaskRule>
    <TaskRule id="application-task-process-delete" RuleTypeCd="Service" Processor="velocity"
    RuleSource="UWRule::rule::none::application-task-process-delete.vtl" />
    </TaskTemplate>
    ...
```

3. Configure one or more of the following parameters for the task:

| | |
|-------------------|--|
| WorkDtDays | The number of days to wait before triggering the reminder task after the initial email is sent to email address on the quote. The default is 10. |
|-------------------|--|

| | |
|---------------------|--|
| MaxReminders | The number of reminders emails to send for each quote. The default is 3. |
|---------------------|--|

Develop a Consumer Service Portal

Developers can use the v5 API to develop a custom consumer service portal that performs the same tasks as the InsuranceNow Consumer Service Portal (Service Portal).

While the business flow for each consumer service portal varies based on business needs, service portals typically provide consumers the ability to report a claim, make payments, open quotes, view policy details, and make changes to coverages.

This section includes typical APIs and API flows that developers can use to develop a service portal. However, the interactive API includes all the parameters available for each API. For information on how to view and test the API and its parameters, see “Try out the API V5” on page 11.

Consumer Service Portal Authentication

This task describes the verification flow for the Service Portal registration.

Procedure

1. Register an application with the InsuranceNow Okta tenant.

```
grant type: client_credentials
```

2. Configure the `jwt-client-settings.xml`

Note: A guest user, `webportal` with Service Portal and Policy Underwriter roles must exist in InsuranceNow.

```
<JWTClient
  CustomerId="serviceportal"
  ClientId="{OKTA_CLIENT_ID}"
  ClientSecret="{OKTA_CLIENT_SECRET}"
  IN_UserName="webportal"
/>
```

3. Configure `okta-http-clientsettings.xml`

```
<OKTAHttpClient
  ClientId="{OKTA_CLIENT_ID}"
  Grant_Type="client_credentials"
  Scope="application"
  AuthServerUri="{OKTA_ISSUER_URL}/v1/token"
  Content-Type="application/x-www-form-urlencoded"
  JWKUri="{OKTA_ISSUER_URL}/v1/keys"
  Issuer="{OKTA_ISSUER_URL}"
/>
```

4. Configure the Service Portal v4 registration fields in registration-criteria.xml

Note: PolicyNumber is required, while all others are optional. Policy number and at least two additional options are required for registration verification.

```
<options key='serviceportal'>
  <option value='PolicyNumber' label='Policy Number' />
  <option value='DateOfBirth' label='Date of Birth' />
  <option value='Zip' label='Zip' />
  <!-- <option value='PolicyEffectiveDt' label='Policy Start Date' />-->
  <!-- <option value='PolicyExpirationDt' label='Policy End Date' />-->
  <!-- <option value='PhoneNumber' label='Phone Number' />-->
  <!-- <option value='Email' label='E-mail' />-->
  <!-- <option value='CustomerNumber' label='Customer Number' />-->
</options>
```

5. Add a user group using the following naming convention: insurer.env.project.role, e.g. iic.qa.serviceportal.users and generate an Okta API token

```
<OKTAHttpClient
  UserGroupId="{OKTA_USER_GROUP_ID}"
  APIToken="{OKTA_API_TOKEN}"
  OrgUrl="https://insurancenow.oktapreview.com"
/>
```

6. In okta-http-clientsettings.xml, configure the user group ID and API token.

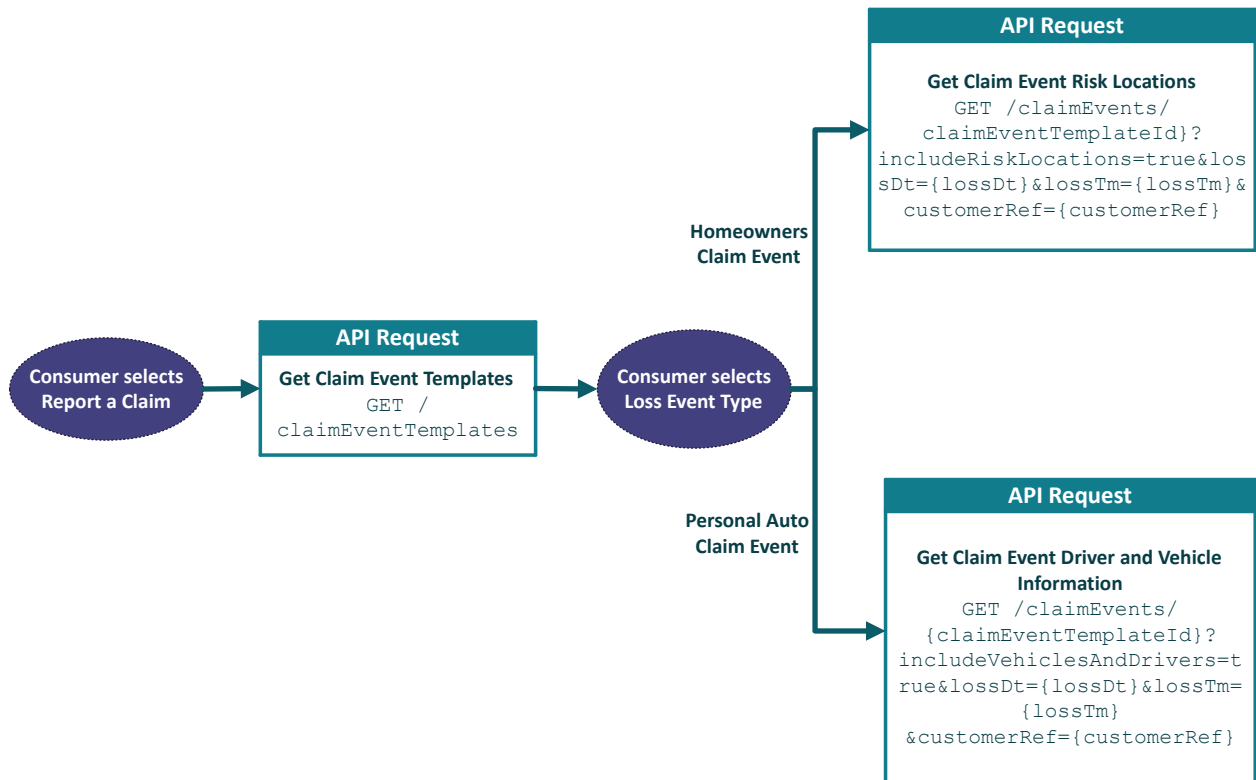
Submit a claim

Consumers create a claim in the Consumer Service Portal (Service Portal) with the following steps:

1. Report a claim.
2. Enter loss details
3. Submit a claim.

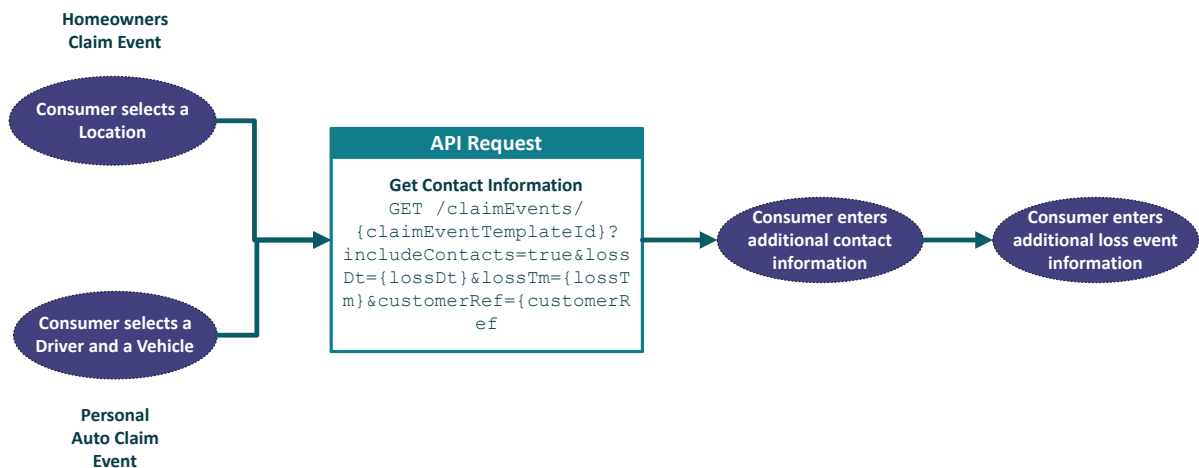
Report a Claim

Submitting a claim starts with the consumer reporting a claim and selecting a loss event type. The types of policies held by the customer and the claim product setup determine the types of loss events that are available to the consumer.



Enter Loss Details

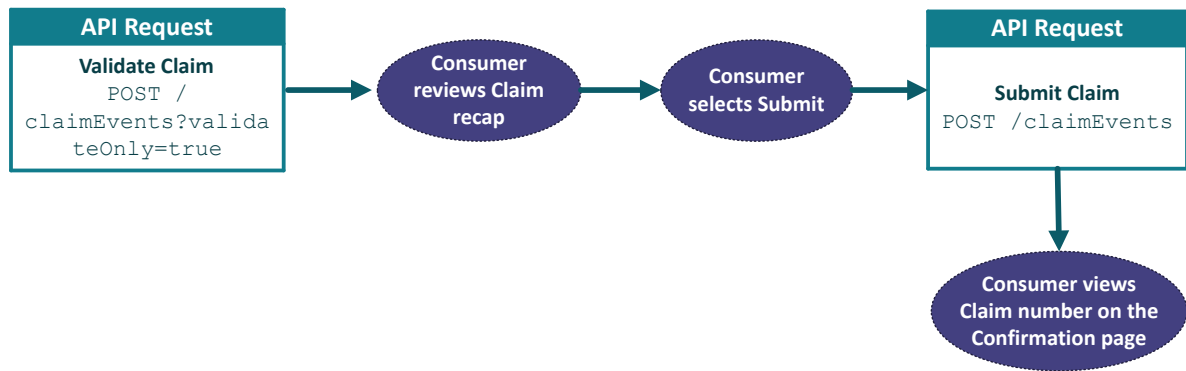
Entering the loss event details includes the consumer selecting a location, entering contact information, and providing additional details on the loss event. Some of the customer information is populated from their existing customer account.



When the consumer enters loss event information, they have the option to add pictures and PDFs. For more information, see “Add attachments in a service portal” on page 34.

Submit a Claim

Submitting a claim includes the consumer reviewing the **Claim recap** page and selecting the option to submit the claim.



Note: In this flow, the portal provides a summary of the data entered by the user on the **Claim Recap** page without the need of an API to retrieve the data. The POST /claimEvents endpoint is the first API call that writes the claim event details to the server.

List claims and claim details

A Consumer Service Portal (Service Portal) can list the claims for a given customer.

- To list the claims for a customer, the portal issues the following call:

```
GET /customers/systemId/claims
```

- To display claim details, the portal issues the following call:

```
GET /claims/systemId/consumerSummary
```

Add attachments in a service portal

In the Consumer Service Portal (Service Portal), consumers can add attachments to claims and policies. Adding an attachment is a multistage process that begins with using the `multifileupload` servlet to upload the file to the server and ends by including an attachment reference from the claim or policy to the file.

Adding an attachment to a loss event.

To add an attachment to a loss event, the portal calls the POST /multifileupload API to upload the attachment to the server and then the portal retrieves the name from the response. Finally, the portal provides the document's file name when it calls the POST /claimevents endpoint to submit the claim.

Adding an attachment to an existing claim

To add an attachment to a claim, the portal calls the POST /multifileupload API to upload the attachment to the server and then the portal retrieves the name from the response. Finally, the portal calls the POST /claims/systemId/documents endpoint to add an attachment reference to the claim.

Add an attachment to a policy

To add an attachment to a policy the portal calls the POST /multifileupload API to upload the attachment to the server and then the portal retrieves the name from the response. Finally, the portal calls the POST /policies/systemId/documents endpoint to add an attachment reference to the policy.

Example code

In the following example code, the `uploadFiles` method uploads the file to the server, and then the `attachFiles` method adds the attachment to a claim.

```
import { API, MULTI_PART_API } from '../api';
import {
  setErrors as policySetErrors,
```

```

    setTotalUploads,
    setCurrentUploads,
    setSubmittingRow as policySetSubmittingRow,
  } from '../components/features/dashboard/policies/policyReducer';

import { setErrors as claimSetErrors } from '../components/features/dashboard/claims/claimsReducer';

const attachFiles = async (filenames, containerType, id, dispatch, notify) => {
  let resource;
  let templateId;
  let memo;
  if (containerType === 'Policy') {
    resource = 'policies';
    templateId = 'XPolicyAttachment0002';
    memo = 'Policy Documents attached by the Consumer Portal';
  } else if (containerType === 'Claim') {
    resource = 'claims';
    templateId = 'XLossNoticeAttachment0002';
    memo = 'Claim Documents attached by the Consumer Portal';
  } else {
    // return if no container type provided
    return;
  }
  // const promises = [];
  dispatch(setTotalUploads(filenames.length));
  dispatch(setCurrentUploads(1));
  for (let i = 0; i < filenames.length; i++) {
    await API.post(`${resource}/${id}/documents`, {
      fileName: filenames[i],
      templateId,
      description: filenames[i],
      memo,
    })
    .then(response => {
      console.log(response);
      notify(`Success, unable to upload ${filenames[i]}`, {
        variant: 'success',
        anchorOrigin: {
          horizontal: 'center',
          vertical: 'top',
        },
      });
      dispatch(setCurrentUploads(i + 1));
    })
    .catch(err => {
      console.log(err);
      notify(`error, unable to upload ${filenames[i]}`, {
        variant: 'error',
        anchorOrigin: {
          horizontal: 'center',
          vertical: 'top',
        },
      });
      dispatch(setCurrentUploads(i + 1));
    })
    .finally(() => {});
  }
  dispatch(policySetSubmittingRow(null));
  dispatch(setCurrentUploads(null));
  dispatch(setTotalUploads(null));
};

export const uploadFiles = (files, id, containerType, notify) => {
  let fd = new FormData();
  files.map(file => fd.append(file.name, file));

  return async dispatch => {
    return MULTI_PART_API.post('multifileupload', fd)
      .then(response => {
        console.log(response);
        const filenames = [];
        // TODO we may need to read the response from response.data.files later
        response.data.map(data => filenames.push(data.name));
        attachFiles(filenames, containerType, id, dispatch, notify);
      })
      .catch(err => {
        console.log(err);
        if (containerType === 'Policy') {
          dispatch(policySetErrors(err.message));
        }
      });
  };
};

```

```

    } else if (containerType === 'Claim') {
      dispatch(claimSetErrors(err.message));
    }
  });
};
};

```

Note: This example might not work for your particular scenario and is only provided as an example.

List policy and policy details

A Consumer Service Portal (Service Portal) can list policies and policy details for a given customer.

Display list of policies with policy name

To display a list of policies with the policy name, call the following APIs:

- To get a list of policies created after a certain date, call the following API:

```
GET /policies?customerId=customerId&createdSinceDt=createdSinceDate
```

The results include information about the latest term of each policy. When the latest term has a future effective date, the results include information about the future policy and a policyref (systemId) value for the current policy.

- To display policy information about the current policy, call the following API:

```
GET /policies/systemId
```

- To determine the product name to display for each policy, call the following API:

```
GET /products?productVersionRefId=policy.policyMini.basicPolicy.productVersionIdRef
```

For example, you can include the product name Homeowners or Premier Personal Auto as part of the policy description.

Display details of a policy

To display the details of a policy, call the following API:

```
GET /policies/systemId/insuredPolicyDetails
```

List policy documents

To list policy documents, call the following API:

```
GET /policies/systemId/insuredDocuments
```

Provide a download link to a policy document

To provide a link that allows the consumer to download a policy document, use the following API:

```
GET /policies/systemId/documents/documentId/content
```

Note: You can determine the documentId values from the API response of the GET `/policies/systemId/insuredDocuments`.

For example, the documentID is `OutputItem-1417739612-1987242998` the API response:

```
{
  "documentListItems": [
    {
      "ref": "OutputItem-1417739612-1987242998",
      "type": "Output",
      "name": "Insured Installment Invoice",
      "description": "Insured Installment Invoice",
      ...
      "_links": [
        {
          "rel": "content",
          "href": "/coreapi/v5/policies/1/documents/OutputItem-1417739612-1987242998/content"
        },
        {
          "rel": "parent",
          "href": "/coreapi/v5/policies/1"
        }
      ]
    }
  ]
}
```

Change policy coverage

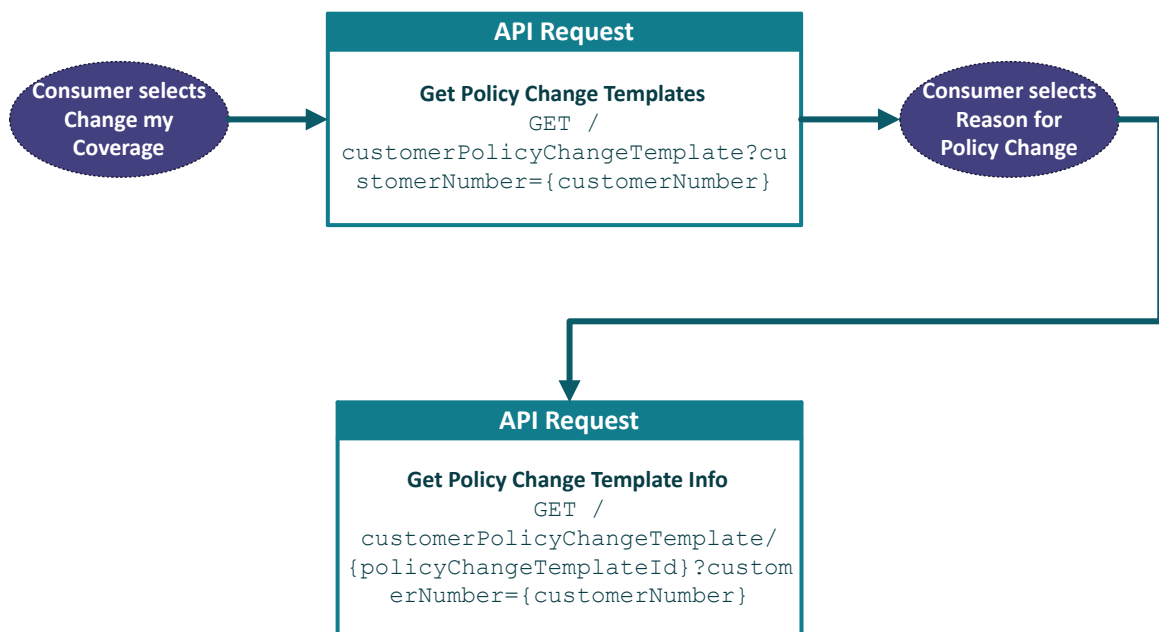
Consumers can change their policy coverage in the Consumer Service Portal (Service Portal) using the following steps:

1. Initiate a coverage change.
2. Enter coverage change details.
3. Submit a coverage change.

The consumer steps and the API calls required to complete each change request vary based on the type of policy coverage change.

Initiate Coverage Change

Initiating a coverage change starts with a consumer deciding to change their coverage and selecting the change type. The types of policies that consumer has determines the types policy change options that consumer can select from. Once the **Reason for Policy Change** is selected, the API retrieves the associated policy change template which includes the information that the portal needs to update the policy.



Enter Coverage Change Details

Updating a coverage on a policy requires the consume to enter detail about the change. API calls then load fields specific to the change request.

Note: The actual field names and steps can vary based on portal configuration. For example, some portals might decide not to preview the impact that a coverage change has on the premium.

| Coverage change request | Process to enter coverage details |
|-------------------------|---|
| Add Vehicle | <ol style="list-style-type: none"> The consumer selects a Policy to Change and enters a Requested Effective Date. The consumer enters the vehicle's make, model, and year. <ul style="list-style-type: none"> If the consumer knows the VIN, they enter the VIN number and then the portal calls the following API to list the models: <pre>GET /vehicleModels</pre> Then, the consumer selects the vehicle from the listed options. If the consumer does not know the VIN, they enter a Year and then the portal calls the following API to list the makes: <pre>GET /vehicleManufacturers</pre> After the consumer selects a Make, the portal calls the following API to list models: <pre>GET /vehicleModels</pre> If the consumer cannot find their car using the methods above, the portal displays text fields for the consumer to enter the vehicle Make, Model, and Year. The consumer enters additional vehicle details and proceeds to the next step. The portal calls the following API to preview the premium change: <pre>POST /customerPolicyChangePreSubmit</pre> The portal calls the following API to display the existing premium: <pre>GET /policies/systemId/full</pre> |

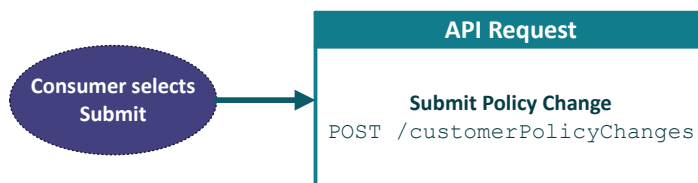
| Coverage change request | Process to enter coverage details |
|--------------------------------|--|
| Remove Vehicle | <ol style="list-style-type: none"> 1. Consumer selects Policy to Change. 2. The portal calls the following API to load the list of existing vehicles on the policy: <pre>GET /policies/systemId/lines/PersonalAuto/risks</pre> 3. The consumer enters the Requested Effective Date and Reason for Removal. They also select the Vehicle to be removed. 4. The portal calls the following API to preview the premium change: <pre>POST /customerPolicyChangePreSubmit</pre> 5. The portal calls the following API to display the existing premium: <pre>GET /policies/systemId/full</pre> |
| Add Driver | <ol style="list-style-type: none"> 1. The portal displays all the driver fields. 2. The consumer enters the information about the new driver. 3. The portal calls the following API to preview the premium change: <pre>POST /customerPolicyChangePreSubmit</pre> 4. The portal calls the following API to display the existing premium: <pre>GET /policies/systemId/full</pre> |
| Remove a Driver | <ol style="list-style-type: none"> 1. The portal calls the following API to load a list of existing drivers on the policy: <pre>GET /policies/systemId/drivers</pre> 2. The consumer enters the Requested Effective Date and Reason for Removal. They also select the Driver to be removed. 3. The portal calls the following API to preview the premium change: <pre>POST /customerPolicyChangePreSubmit</pre> 4. The portal calls the following API to display the existing premium: <pre>GET /policies/systemId/full</pre> |
| Change Billing/Mailing Address | <ol style="list-style-type: none"> 1. The portal calls the following API to get a list of policies for the customer: <pre>GET /policies?customerID=customerId</pre> 2. The portal extracts the mailing address from the insured:partyinfo:addresses section of the response of GET /policies?customerID=customerId. 3. If the response of GET /policies?customerID=customerId does not include a statementAccountRef, the portal extracts the insured's billing address from the insured:partyinfo:addresses section of the response. 4. If the response of GET /policies?customerID=customerId includes a statementAccountRef, the portal completes the following steps to get the billing address for the statement account: <ol style="list-style-type: none"> a. It calls the following API to get a list of billing accounts: <pre>GET /billingAccounts?customerId=customerId</pre> b. It notes the system ID of the billing account that matches the statementAccountRef. c. It calls the following API with the system ID of the billing account that matches the statementAccountRef: <pre>GET /billingAccounts/systemId/full</pre> |

| Coverage change request | Process to enter coverage details |
|--------------------------------|---|
| | <ol style="list-style-type: none"> The consumer enters the Requested Effective Date, and Reason for Change. They also select the Address to be changed. The consumer enters the new address. |
| Change Vehicle Finance Company | <ol style="list-style-type: none"> The consumer selects a Policy to Change. The portal calls the following API to load the interest types: <pre>GET /coderefs/UW/ai/interest-type/vehicle-finance-company</pre> The portal calls the following API to load the list of finance companies available on the policy: <pre>GET /policies/systemId/additionalInterests</pre> The consumer enters the Requested Effective Date and selects the Vehicle Finance Company to be Changed. They also enter the Reason for change. The consumer provides details on the new finance company. |
| Remove Vehicle Finance Company | <ol style="list-style-type: none"> The consumer selects a Policy to Change. The portal calls the following API to load the list of finance companies available on the policy: <pre>GET /policies/systemId/additionalInterests</pre> The consumer enters a Requested Effective Date and Reason for Removal. They also select a Vehicle Finance Company to be Removed. |
| Add Vehicle Finance Company | <ol style="list-style-type: none"> The consumer selects a Policy to Change. The portal calls the following API to load the interest types: <pre>GET /coderefs/UW/ai/interest-type/vehicle-finance-company</pre> The portal calls the following API to load the list of vehicles available on the policy: <pre>GET /policies/{systemId}/lines/PersonalAuto/risks</pre> The consumer picks a vehicle from Select Finance Vehicle. The consumer enters the fields in the New Vehicle Finance Company Information panel. |
| Change Property Deductible | <ol style="list-style-type: none"> The consumer selects a Policy to Change. The portal calls the following API to load the policy details: <pre>GET /policies/systemId</pre> The consumer selects a New Deductible and Requested Effective Date. They also can enter a Reason. The portal calls the following API to preview the premium change: <pre>POST /customerPolicyChangePreSubmit</pre> The portal calls the following API to display the existing premium: <pre>GET /policies/systemId/full</pre> |
| Change the Property Mortgage | <ol style="list-style-type: none"> The consumer selects a Policy to Change. The portal calls the following API to load additional interests on the policy: <pre>GET /policies/systemId/additionalInterests</pre> The consumer selects a Requested Effective Date, a Mortgagee to be changed, and then clicks Next. The portal calls the following API to load the mortgagee interest types: <pre>GET /coderefs/UW/ai/interest-type/mortgagee</pre> |

| Coverage change request | Process to enter coverage details |
|-------------------------------|--|
| | <ol style="list-style-type: none"> The consumer selects a Reason for Change and enters additional fields in the Current Mortgagee Information panel. |
| Add Property Mortgagee | <ol style="list-style-type: none"> The consumer selects a Policy to Change. The portal calls the following API to load the building risks associated with the policy: <pre>/policies/systemId/lines/Homeowners/risks</pre> The consumer enters the Requested Effective Date, selects one of the Properties, and clicks Next. The portal calls the following API to load the mortgagee interest types: <pre>GET /coderefs/UW/ai/interest-type/mortgagee</pre> The consumer enters the Reason for Change and the fields in the New Mortgagee Information panel. |
| Remove Property Mortgagee | <ol style="list-style-type: none"> The consumer selects a Policy to Change. The portal calls the following API to load the additional interests on the policy: <pre>GET /policies/systemId/additionalInterest</pre> The consumer enters the Requested Effective Date and a Description. They also select a Mortgagee to be Removed. |
| Request New Property Coverage | <ol style="list-style-type: none"> The consumer enters the Requested Effective Date and a Description of the items they want to cover and their items values. If the consumer wants to add pictures of the items, see “Add attachments in a service portal” on page 34. |

Submit Coverage Change

Submitting a coverage change requires a consumer to submit their request and then the portal calls an API to update the policy.

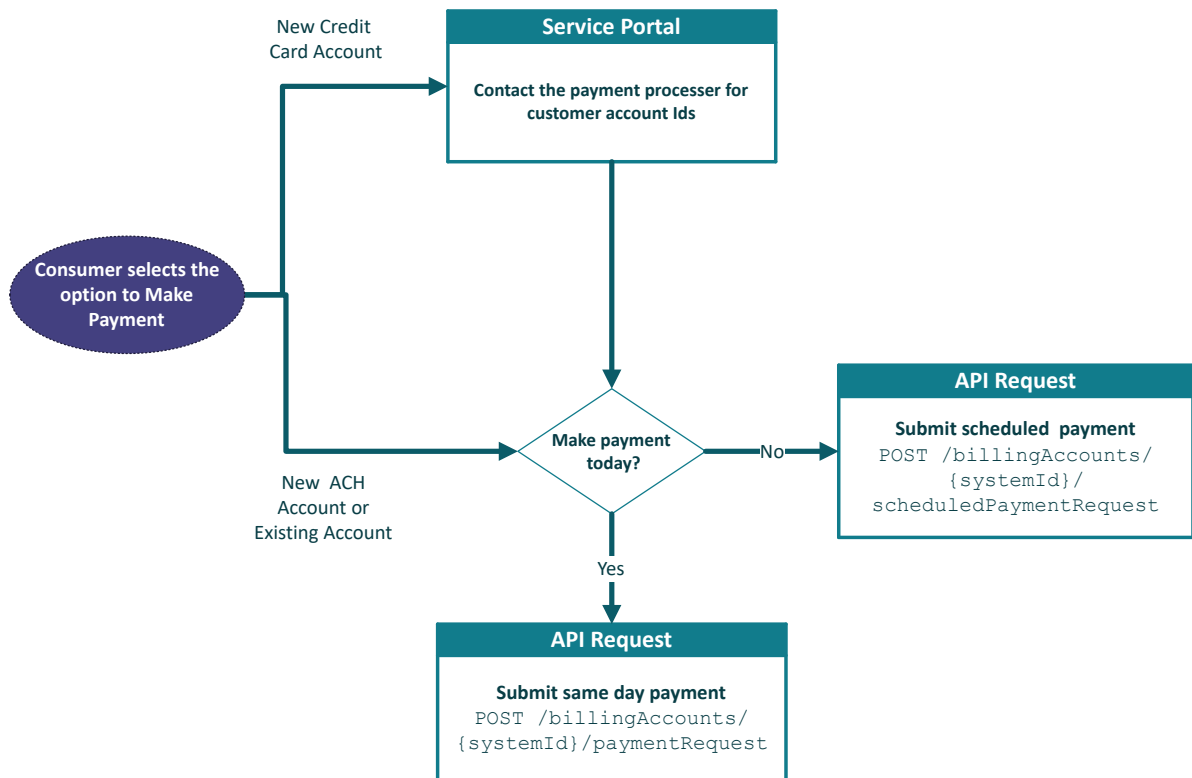


Make payments

Consumers can make payments on policies in the Consumer Service Portal (Service Portal) using the following steps:

1. Select **Make a Payment**.
2. Enter payment details.
3. Select a payment method. The Service Portal lists existing payment methods on file and the option to add a new payment method.
4. Review payment details and click **Make Payment**.

The API calls required to complete the payment request vary based on the payment account and the payment date.



Process to contact payment processor for new credit card account

Based on the payment processor, new credit card payments require some additional steps to obtain account details before the Service Portal can submit a payment request.

Using Payment Service

Before submitting a payment to a new credit card that uses the Payment Service, the Service Portal completes the following steps:

1. The Service Portal uses the GuidewirePayments SDK to display an iframe that gathers the credit card information from the consumer.
2. The GuidewirePayments SDK returns an ID to the Service Portal. The Service Portal uses this ID as the customerPaymentProfileId.
3. The Service portal generates the customerProfileId.

Using Authorize.net (Service Portal version 3 only)

Before submitting a payment to a new credit card that uses Authorize.net, the Service Portal completes the following steps:

1. The Service Portal displays the Authorize.net iframe so that the consumer can provide the credit card information to Authorize.net.
2. Authorize.net returns the customerProfileId, and customerPaymentProfileId.

Example API request for making a payment

Based on the payment date, the Service Portal calls one of following APIs to make a payment:

- The following API to processes the payment on the current date:

```
POST /billingAccounts/systemId/paymentRequest
```

- The following API processes payment on a scheduled date that is no more than 14 days in the future:

POST /billingAccounts/*systemId*/scheduledPaymentRequest

The body of the API request varies based on the payment method and payment details. The following examples provide guidance regarding the fields to include in the body of the request. However, the requirements and content might differ for your implementation.

Credit Card example

Credit card payment requests generally include the following fields:

```
{
  "scheduledDate": "YYYY-MM-DD",
  "paymentMethod": "Credit Card",
  "payments": [
    {
      "sourceRef": "BillingAccountSystemId",
      "sourceCd": "SourceCd",
      "receiptAmt": "PaymentAmount",
      "checkAmt": "PaymentAmount",
      "electronicPaymentSource": {
        "creditCardNumber": "CreditCardNumber",
        "customerProfileId": "CustomerProfileId",
        "customerPaymentProfileId": "CustomerPaymentProfileId",
        "methodCd": "Credit Card",
        "sourceName": "Mobile",
        "paymentServiceAccountId": "AccountID"
      }
    }
  ]
}
```

ACH payment example

ACH payment requests generally include the following fields:

```
{
  "scheduledDate": "YYYY-MM-DD",
  "paymentMethod": "ACH",
  "payments": [
    {
      "sourceRef": "BillingAccountSystemId",
      "sourceCd": "SourceCd",
      "receiptAmt": "PaymentAmount",
      "checkAmt": "PaymentAmount",
      "electronicPaymentSource": {
        "achName": "CardHolderName",
        "achBankAccountTypeCd": "Checking",
        "achBankAccountNumber": "AccountNumber",
        "achRoutingNumber": "RoutingNumber",
        "achStandardEntryClassCd": "WEB"
      }
    }
  ]
}
```

Payment on file example

Credit card and ACH payment request that use saved account information generally include the following fields:

```
{
  "scheduledDate": "YYYY-MM-DD",
  "paymentMethod": "PaymentMethod",
  "payments": [
    {
      "sourceCd": "SourceCd",
      "receiptAmt": "PaymentAmount",
      "checkAmt": "PaymentAmount",
      "paymentOnFileId": "PaymentOnFileId"
    }
  ]
}
```


Develop an Agent Portal

Developer can use the v5API to develop an agent portal that provides an interface for agents to submits a quote for approval and manage their tasks.

This section includes typical APIs and API flows that developers can use to develop an agent portal. However, the interactive API includes all the parameters available for each API. For information on how to view and test the API and its parameters, see ““Try out” the API V5” on page 11.

Agent Portal authentication

Oauth is an authorization framework that enables the Agent portal to obtain limited access to an HTTP service.

Before you begin

The system must be configured to use multi-factor authentication (MFA).

Procedure

1. You need to configure the public and private keys paths in the `jwt-settings.xml` for local environments and `config.properties` for each deployment environment.

Example:

```
<JWTSettings>
  <Param Name="TokenIssuer" Value= "Guidewire Software Inc." />
  <Param Name= "PublicKeyPath" Value= "${SPI.getPrefsDir()}mda/jwt/public.pem" />
  <Param Name= "PrivateKeyPath" Value= "${SPI.getPrefsDir()}mda/jwt/private_pkcs8.der" />
  <Param Name= "UserNameClaimMapping" Value= "sub" />
</JWTSettings>
```

2. Configure clients in `web/APP-INF/mda/oauth/clients.xml`. A client must be configured for each client application using OAuth.

| | |
|----------------------------------|--|
| ClientId | This must be set to some value which uniquely identifies the client. Note: Only public clients are supported. |
| GrantTypes | Comma separated list of grant types allowed for the client, including: <ul style="list-style-type: none">• Authorization Code• Implicit |
| AccessTokenExpirationTime | Expiration time of the access token in minutes. When the token expires, the user will be forced to log in again to get a new token. |

Note: Refresh tokens are not supported.

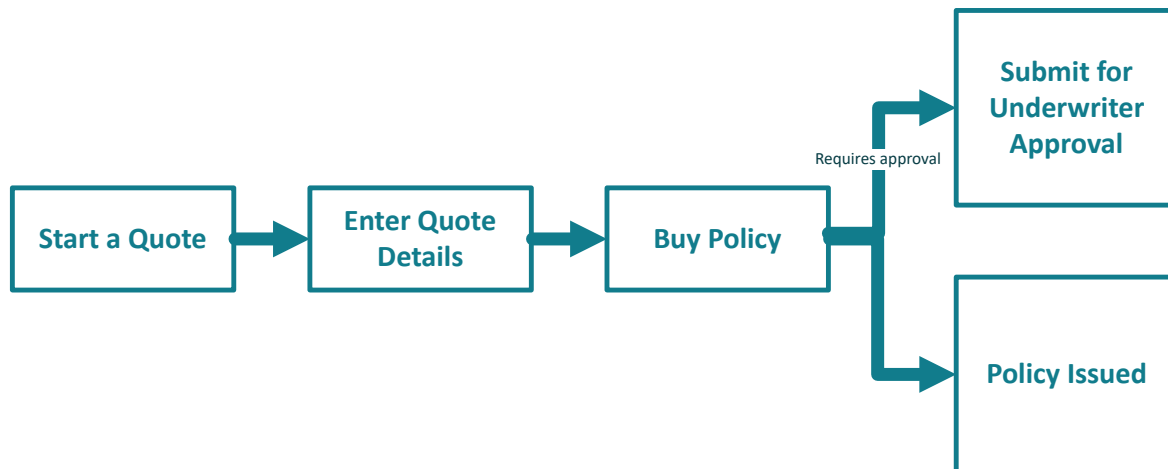
RedirectUri

Comma separated list of allowed redirection URIs for the client.

- The redirection endpoint URI must be an absolute URI as defined by [\[RFC3986\] Section 4.3](#).
- The endpoint URI may include an "application/x-www-form-urlencoded" formatted query component, which must be retained when adding additional query parameters.
 - See [Appendix B](#) and [\[RFC3986\] Section 3.4](#) for more details.
- The endpoint URI must not include a fragment component.

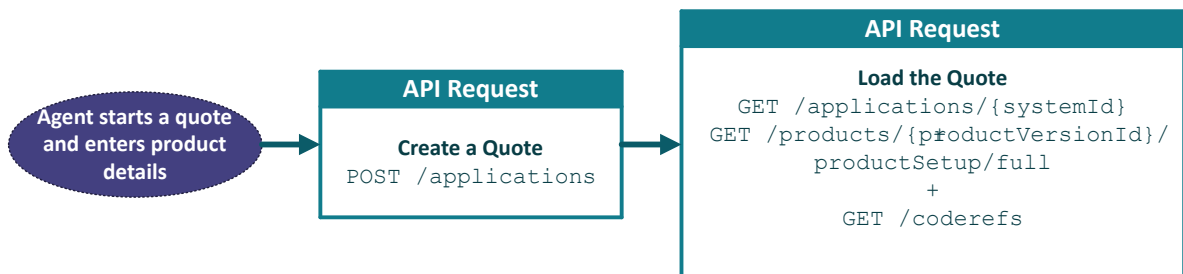
Submit a quote for approval

In an agent portal, following API flow to submit a quote for approval applies to most implementations:



Start a Quote

The process required to start a quote includes the agent initiating a new quote and selecting the product, and then the API is called to create a quote, access the quote, and load the details required to start the quote.

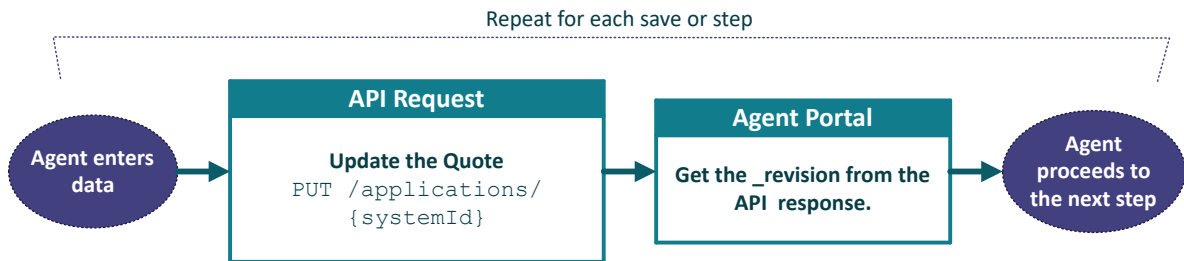


Note:

The complete URL of the application is available in the Location header of the POST /applications API response. The URL of the application includes the `systemId`.

Enter Quote Details

The process required to enter quote details is an iterative process where the user enters quote details on various tabs or pages, resolves any errors, and then has the opportunity to apply for the policy.



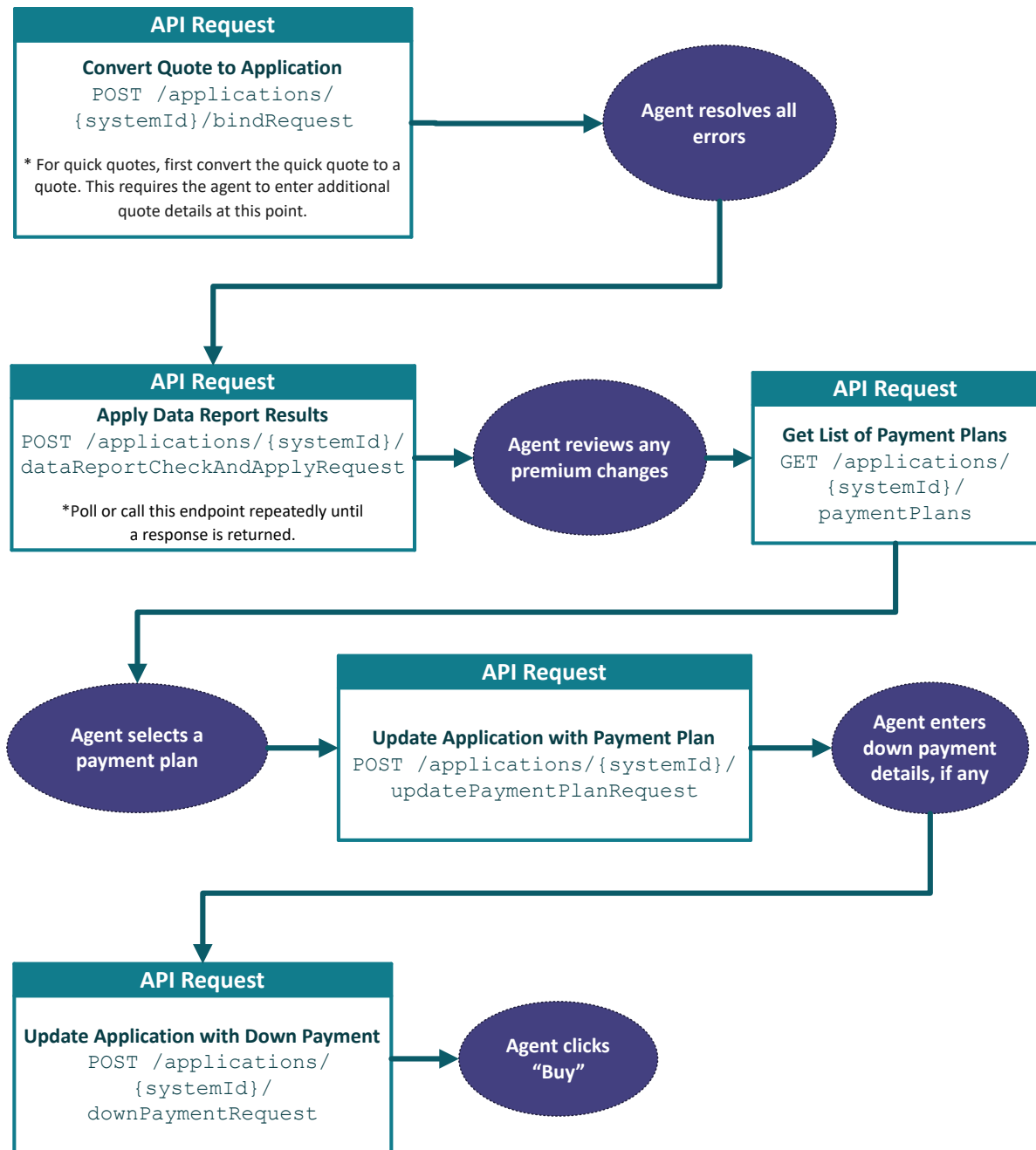
Each time the portal loads or saves the quote as part of a step or similar action, the portal submits a PUT request with the latest revision number.

Note: To save the quote, the revision number is required.

Buy the Policy

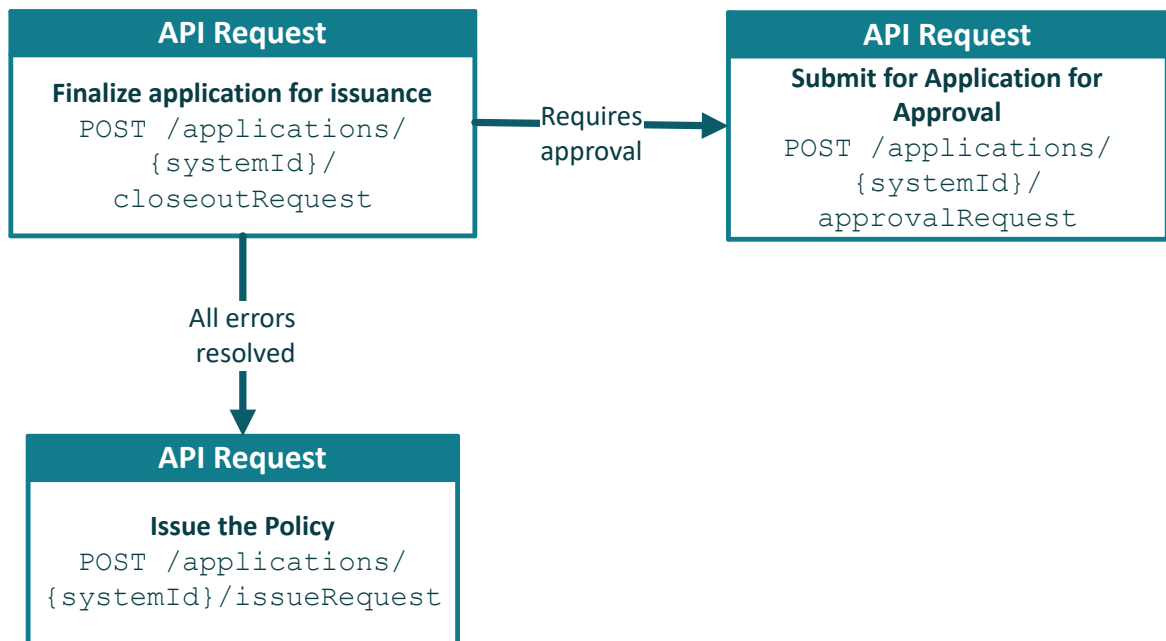
The process required to buy a policy includes API requests to convert the quote to an application, apply data report results, list the payment plans, and update the application with payment detail.

Note: The results of the data reports may trigger a recalculation of premium.



Issue Policy or Submit for Approval

The process to issue a policy includes either an API request to issue the policy or submit it for underwriting approval, if there are unresolved approval issues.



View and delete tasks

In an agent portal, you can call APIs to list tasks that are open for an agent and provide a way to view the task details.

View a list of open tasks for the current user

To display a list of open tasks for the current user, call the following API:

```
GET /tasks?status=open&ownedByUser=true
```

View the task details

To view the details of a specific task, call the following API:

```
GET /tasks/systemID
```

View the resource associated with a task

A task resource is the container that the task is associated with. For example, when a task is associated with a customer, policy, or application, the API considers the customer, policy, or application to be a resource.

To view the resource that is associated with a task, call the following API:

1. To view the details of a specific task, call the following API:

```
GET /tasks/systemID
```

2. From the API response, note the `idRef` and `modelName` values of the resource in the `taskLinks` section.

```
{
  "taskListItems": [
    {
      ...
      "taskLinks": [
        {
          "id": "TaskLink-*****_*****",
          "idRef": "SystemID",
          "modelName": "Customer",
          ...
        }
      ]
    }
  ]
}
```

```
    ],  
    ...
```

In this example, the `idRef` field specifies the `systemID` and the `modelName` is the resource or container type that the task is associated with.

3. To view the resource, call the appropriate API. For example, to view a customer resource, call `GET /customers/{systemID}`

Delete a task

To change the status of a task to **deleted**, call the following API:

```
GET /tasks/systemID/deleteRequest
```

Create a task on a policy or application

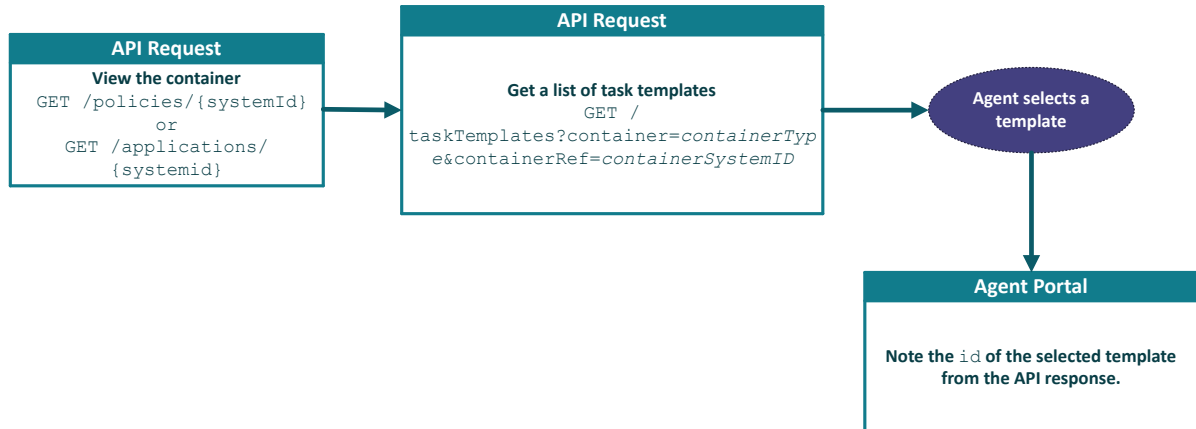
In an agent portal, you can create a task on a policy or an application. When you create a task on a policy or an application, the task becomes a workflow item for the policy or application and it can be sent to a user's Inbox.

The steps to creating a task on a policy or an application are similar. Creating a task includes the following steps:

1. Identify the task template.
2. Create the task.

Identify the applicable task templates

To identify the applicable task templates, the agent decides which policy or application requires the task. Then, the agent portal calls an API to determine the list of applicable templates. Once the agent selects one of the templates, the agent portal must note the template ID as it is required to create the task.



For example, the following steps may occur:

1. View the application or policy details:

Viewing a policy

- a. The portal submits the following API call:

```
GET /policies/systemid
```

- b. The portal displays the policy details provided in the response.

Viewing an application

- a. The portal submits the following API call:

```
GET /applications/systemid
```

- b. The portal displays the policy details provided in the response.

2. View the list of available tasks:

- a. The portal submits the following API call:

```
GET /taskTemplates?container=containerType&containerRef=containerSystemID
```

For example: GET /taskTemplates?container=Policy&containerRef=121

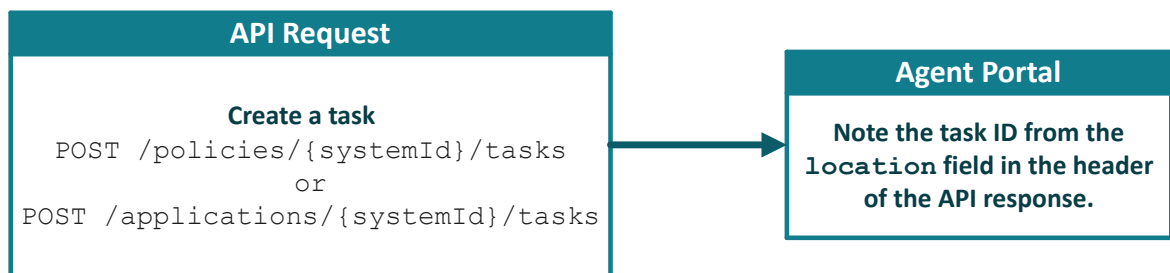
- b. The portal provides the agent a list of available task templates based on the API response and the agent selects one.
- c. From the API response, the portal notes the id of the task the agent want to create.

For example:

```
{
  "taskTemplateListItems": [
    {
      "id": "PolicyTask0001",
      "name": "General Reminder",
      "taskTypeCd": "Manual",
      "_links": [
        {
          "rel": "self",
          "href": "https://hostname/coreapi/v5/taskTemplates/PolicyTask0001?
container=Policy&containerRef=121"
        }
      ]
    },
    {
      "id": "PolicyTask0002",
      "name": "General Task",
      "taskTypeCd": "Manual",
      "_links": [
        {
          "rel": "self",
          "href": "https://hostname/coreapi/v5/taskTemplates/PolicyTask0002?
container=Policy&containerRef=121"
        }
      ]
    }
  ],
  ...
}
```

Create a task

The API to create a task differs based on the container or resource type.



For example, the following steps may occur:

1. Create the task:

Creating a task on a quote or application

To create a task on a quote or application, execute the following API call:

| Endpoint | POST /applications/{systemId}/tasks |
|----------|-------------------------------------|
| | |

Example
Request Body

```
{
  "templateId": "ApplicationTask0001",
  "priority": 0,
  "description": "Contact applicant to get more details",
  "workDt": "2021-04-20"
}
```

Note: The task template configuration determines which fields are required and the values differ based on agent selections.

Creating a task on a policy

To create a task on a policy, execute the following API call:

| Endpoint | POST /policies/{systemId}/tasks |
|----------|---------------------------------|
|----------|---------------------------------|

Example
Request Body

```
{
  "templateId": "PolicyTask0002",
  "priority": 0,
  "description": "Task to check the status",
  "reportTo": "bobagent",
  "workDt": "2021-04-19",
  "dueDt": "2021-04-19"
}
```

Note: The task template configuration determines which fields are required and the values differ based on agent selections.

2. Locate the systemId of the new task.

After you create a policy task or an application task, the `location` field in the header of the API response includes the URL of the new task and the `systemID` of the new task:

```
...
date: Mon19 Apr 2021 14:22:37 GMT
feature-policy: fullscreen 'none'; microphone 'none'
haproxy: haproxy-ingress-9bfc7d8df-r9shh
location: http://hostname/coreapi/v5/tasks/systemID
referrer-policy: no-referrer
...
```

Create a note on a policy or application

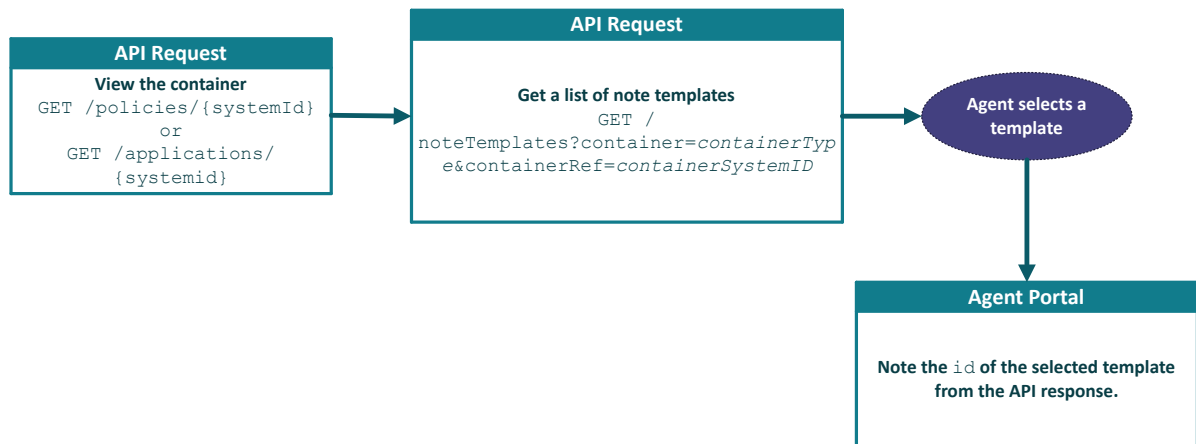
In an agent portal, you can create a note on a policy or an application.

Creating a note on a policy or application includes the following steps:

1. Identify the note template.
2. Create a note on the policy or application.

Identify the note template

To identify the applicable note templates, the agent decides which policy or application requires the note. Then, the agent portal calls an API to determine the list of applicable templates. Once the agent selects one of the templates, the agent portal must note the template ID as it is required to create the note.



For example, the following steps may occur:

1. View the application or policy details:

Viewing a policy

- a. The portal submits the following API call:

```
GET /policies/systemid
```

- b. The portal displays the policy details provided in the response.

Viewing an application

- a. The portal submits the following API call:

```
GET /applications/systemid
```

- b. The portal displays the policy details provided in the response.

2. To view the list of available note templates, the portal submits the following API call:

```
GET /noteTemplates?container=containerType&containerRef=containerSystemId
```

- Application example: `GET /noteTemplates?container=Application&containerRef=6314`
- Policy example: `GET /noteTemplates?container=Policy&containerRef=484`

3. The portal provides the agent a list of available note templates based on the API response and the agent selects one.

4. From the API response, the portal notes the id of the note the agent wants to create.

- Application example:

```
{
  "noteTemplateListItems": [
    {
      "id": "ApplicationNote0001",
      "name": "Agent Note",
      "_links": [
        {
          "rel": "self",
          "href": "/coreapi/v5/noteTemplates/ApplicationNote0001?container=Application&containerRef=6314"
        }
      ]
    },
    {
      "id": "ApplicationNote0002",
      "name": "Company Note",
      "_links": [
        {
          "rel": "self",
          "href": "/coreapi/v5/noteTemplates/ApplicationNote0002?container=Application&containerRef=6314"
        }
      ]
    }
  ]
}
```

```
    },
    ...

```

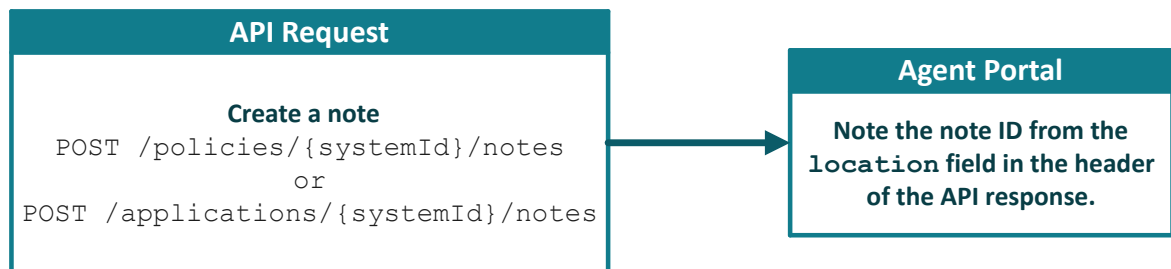
- Policy example:

```
{
  "noteTemplateListItems": [
    {
      "id": "PolicyNote0001",
      "name": "Agent Note",
      "_links": [
        {
          "rel": "self",
          "href": "/coreapi/v5/noteTemplates/PolicyNote0001?container=Policy&containerRef=484"
        }
      ]
    },
    {
      "id": "PolicyNote0002",
      "name": "Company Note",
      "_links": [
        {
          "rel": "self",
          "href": "/coreapi/v5/noteTemplates/PolicyNote0002?container=Policy&containerRef=484"
        }
      ]
    }
  ]
}
...

```

Create a note

The API to create a note differs based on the container or resource type.



For example, the following steps may occur:

1. Create the note.

Creating a note on an application

To create a note on an application, execute the following API:

| | |
|----------------------|--|
| Endpoint | POST /applications/ <i>systemId</i> /notes |
| Example Request Body | <pre>{ "templateId": "ApplicationNote0001", "description": "Inspector needs to check out the roof", "priorityCd": "1", "memo": "The roof might be hazardous" }</pre> |

Note: The values differ based on agent selections.

Creating a note on a policy

To create a note on a policy, execute the following API:

| | |
|---|---|
| Endpoint | POST /policies/ <i>systemId</i> /notes |
| Example Request Body | <pre>{ "templateId": "PolicyNote0001", "description": "Inspector needs to check out the roof", "priorityCd": "1", "memo": "The roof might be hazardous" }</pre> |
| Note: The values differ based on agent selections. | |

2. Locate the *systemId* of the new note.

After you create a policy note or an application note, the *location* field in the header of the API response includes the URL of the new task and the *noteId* of the new task:

```
...
date: Tue20 Apr 2021 18:01:57 GMT
feature-policy: fullscreen 'none'; microphone 'none'
haproxy: haproxy-ingress-9bfc7d8df-mbpwt
location: https://hostname/coreapi/v5/containerType/containerSystemId/notes/noteId/content
referrer-policy: no-referrer
...
```

View and delete notes

In an agent portal, you can call APIs to list and delete notes.

Viewing all notes on a policy

To view a list of notes for a specific policy, call the following API:

```
GET /policies/systemId/notes
```

Viewing a specific policy note

To view a specific policy note, call the following API:

```
GET /policies/systemId/notes/noteId
```

Deleting a policy notes

To delete a policy note, call the following API:

```
DELETE /policies/systemId/notes/noteId
```

Viewing all notes on an application

To view a list of notes for a specific application, call the following API:

```
GET /applications/systemId/notes
```

Viewing a specific application note

To view a specific application note, call the following API:

```
GET /applications/systemId/notes/noteId
```

Deleting an application note

To delete an application note, call the following API:

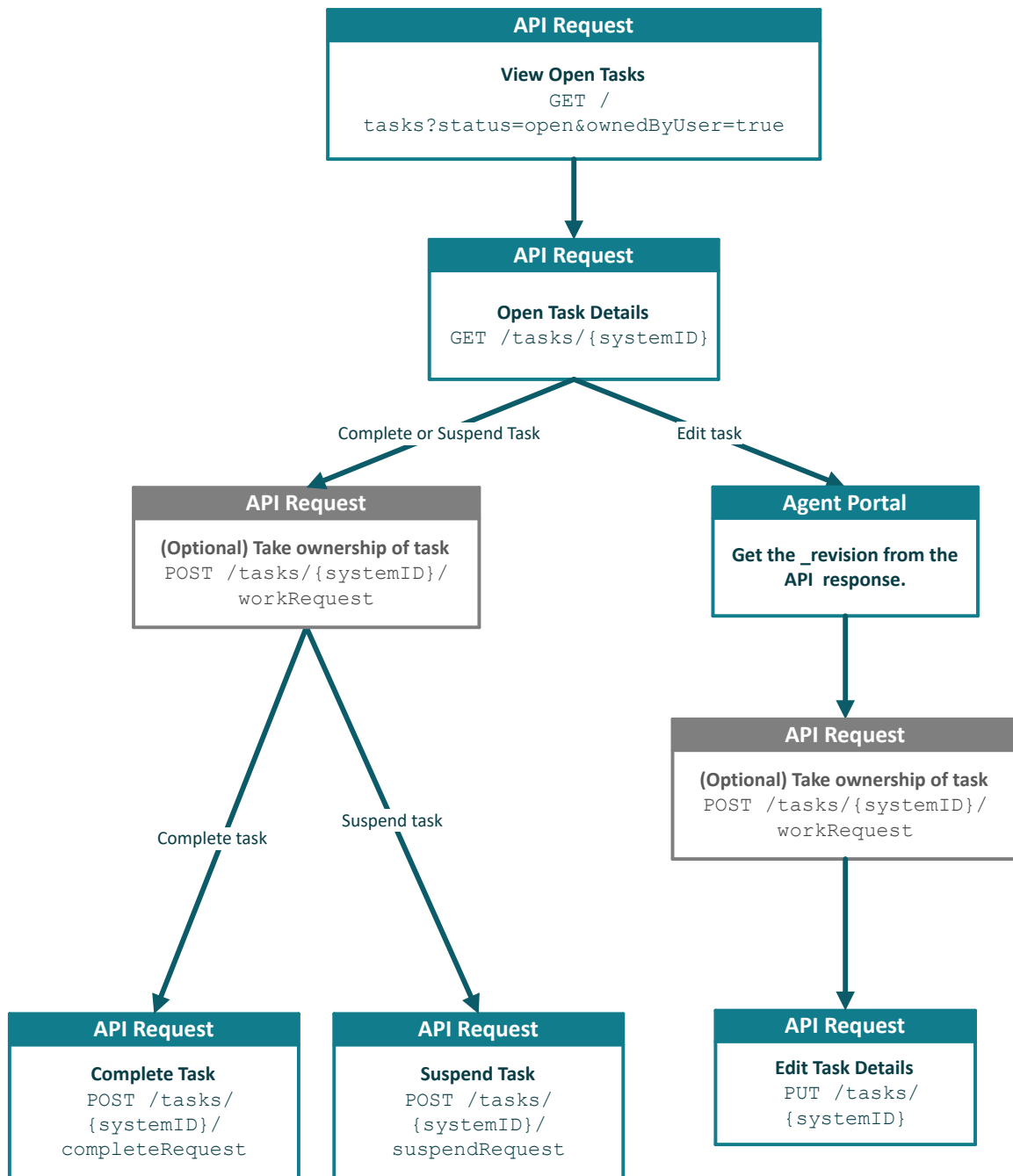
```
DELETE /applications/systemId/notes/noteId
```

Work on tasks

In general, agent portals include the ability to view open tasks and then edit the task with more details or a change in status.

The process to work on a task can include the following steps:

1. The portal displays a list of open tasks that the agent has the authority to own. If the agent is part of a group, they see tasks assigned to groups they are associated with as well.
2. The agent selects a task to view.
3. The portal displays the task details.
4. Optionally, the agent takes ownership of the task. It is not required for an agent to take ownership of a task before making changes to a task.
5. The agent determines how to address the task. This may include suspending the task, complete the task, or editing the task.



Transfer a task

In an agent portal, you can call APIs to transfer a task to a user or group.

Transfer task to a user

To transfer a task to a user, complete the following API calls:

1. To get a list of users, call the following API:

```
GET /users
```

Optionally, include parameter to filter the results. The response includes the loginId of each user that API call returns.

2. To specify the new task owner, execute the following API call:

| | |
|----------------------|---|
| Endpoint | POST /tasks/systemId/transferRequest |
| Example Request Body | <pre>{ "currentOwner": "bobagent", "comments": "Assigning to Bob Agent " }</pre> <p>Provide the loginId of the new task owner in the currentOwner field . Optionally, include comments. Note: The values will differ based on agent selections.</p> |

Transfer task to a group

1. To get a list of available groups, execute the following API call:

```
GET /tasks/systemId/availableUserTaskGroups
```

2. From the response, note the value of the group to which you want transfer the task to.

```
{
  "id": "APIGroupList-*****_*****",
  "option": [
    ...
    {
      "name": "Underwriting Clerk",
      "value": "UWClerk"
    },
    {
      "name": "Producer",
      "value": "ABC"
    },
    {
      "name": "Underwriting",
      "value": "UW"
    },
    ...
  ]
}
```

3. To specify the new group owner, execute the following API call:

| | |
|----------------------|---|
| Endpoint | POST /tasks/systemId/transferRequest |
| Example Request Body | <pre>{ "currentGroup": "UWClerk", "comments": "Assigning to UWClerk group " }</pre> <p>Indicate the group value in the currentGroup field. Optionally, include comments Note: The values will differ based on agent selections.</p> |

Endorse a policy

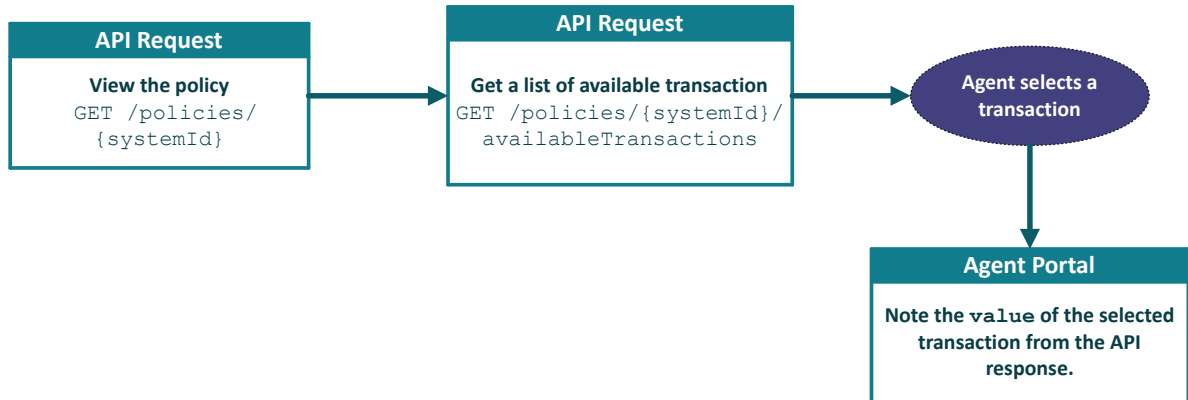
In an agent portal, agents can create an endorsement transaction in order to submit changes to an existing policy.

Endorsing a policy includes the following steps:

1. Identify the policy transaction.
2. Submit and process the endorsement.

Identify the policy transaction

Initiating an endorsement starts with an agent deciding to create a new transaction on a policy. Then, agent portal calls an API to determine the types of transactions that are valid for a policy.



Once an agent selects an endorsement transaction, the agent portal must note the code value of the transaction as it is required to request the transaction in a later step.

For example, the following steps may occur:

1. View the policy details:
 - a. The portal submits the following API call:
2. View the list of available transactions:
 - a. The portal submits the following API call:
 - b. The portal provides the user a list of available transactions based on the API response.
 - c. The agent selects the Endorsement transaction.
 - d. From the API response, the portal notes the value of the Endorsement option.

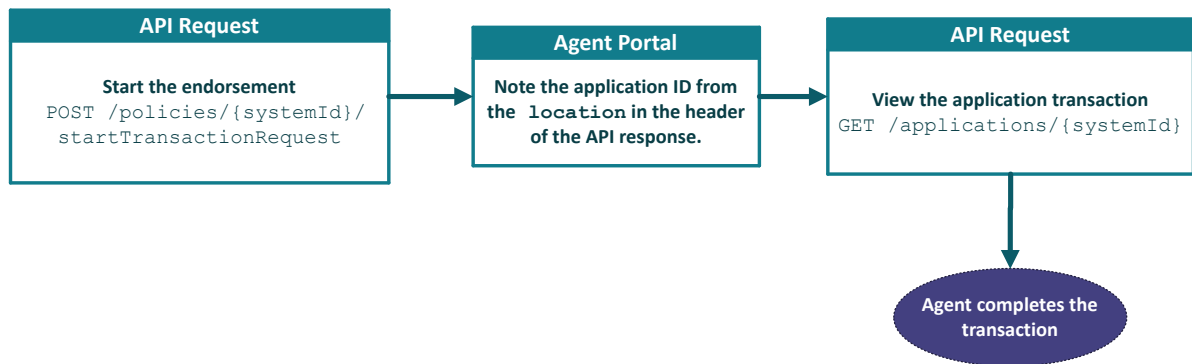
```
GET /policies/systemid
```

```
GET /policies/systemid/availableTransactions
```

```
[
  ...
  {
    "value": "Cancellation",
    "name": "Cancellation"
  },
  {
    "value": "Endorsement",
    "name": "Endorsement"
  },
  ...
]
```

Submit and process the endorsement

Submitting and processing an endorsement request starts with the agent portal calling an API to initiate an endorsement. Then, the system creates an application and the agent processes the application.



For example, the following steps may occur:

1. Start the endorsement transaction request:
 - a. The portal executes the following API call:

| | |
|--|--|
| Endpoint | POST /policies/ <i>systemId</i> /startTransactionRequest |
| Example Request Body | <pre>{ "effectiveDate": "YYYY-MM-DD", "transactionCd": "Endorsement" }</pre> |
| Provide the value of the Endorsement transaction as the transactionCd. | |

- b. From the API response header, note the *systemID* of the application in the location field of the API response header:

```
...
location: https://hostname/coreapi/v5/applications/systemID
...
```

2. View the new application
 - a. The portal executes the following API call:

```
GET /applications/systemid
```

- b. The portal displays the application details provided in the API response.
 - c. The agent completes the application.

Cancel a policy

In an agent portal, agents can cancel an existing policy.

Agents cancel policies by processing a Cancellation Notice transaction followed by a Cancellation transaction. Or, agents can cancel a policy by directly processing a Cancellation transaction.

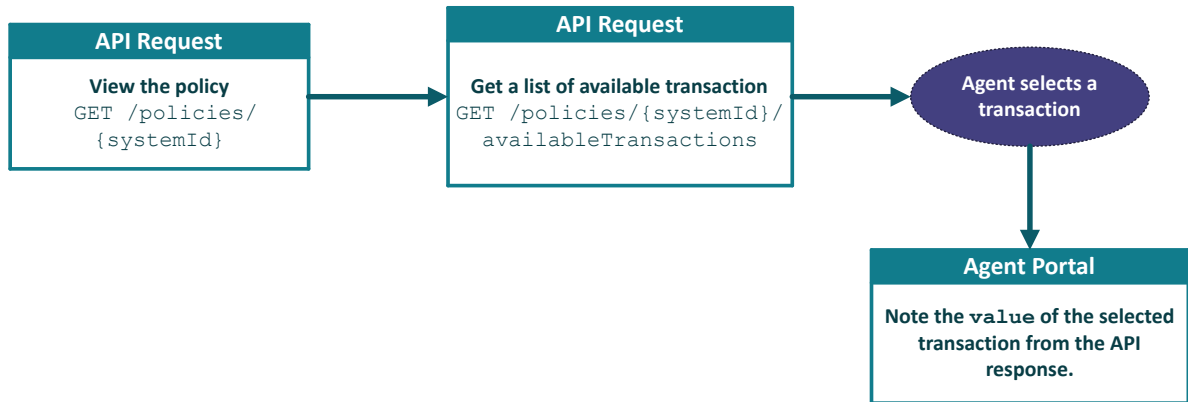
Processing a cancellation transaction includes the following steps:

1. Identify the policy transaction.
2. Gather the cancellation details.
3. Submit and process the Cancellation.

Note: The Cancellation Notice process is the same as the Cancellation process except that the transactioncd is different.

Identify the policy transaction

Initiating a cancellation starts with an agent deciding to create a new transaction on a policy. Then, agent portal calls an API to determine the types of transactions that are valid for a policy.



Once an agent selects the Cancellation transaction, the agent portal must note the code value of the transaction as it is required to request the transaction in a later step.

For example, the following steps may occur:

1. View the policy details:
 - a. The portal submits the following API call:
2. View the list of available transactions:
 - a. The portal submits the following API call:
 - b. The portal provides the user a list of available transactions based on the API response.
 - c. The agent selects the Cancellation transaction.
 - d. From the API response, the portal notes the value of the Cancellation option.

```
GET /policies/systemid
```

```
GET /policies/systemid/availableTransactions
```

```
[
  ...
  {
    "value": "Cancellation",
    "name": "Cancellation"
  },
  {
    "value": "Endorsement",
    "name": "Endorsement"
  },
  ...
]
```

Gather the cancellation details

Before an agent portal can submit a Cancellation transaction request, the agent portal needs to gather the following details:

requestedByCd

The agent portal calls the following API to determine valid requestedByCd values:

```
GET /coderefs/UWPolicy/policy/transaction/cancel-requested-by
```

Then, the agent selects the appropriate code.

reasonCd

The agent portal calls the following API to determine the cancellation reason codes that are valid for a policy:

```
GET /policies/{systemId}/transactionReasons/cancelRequestedByCd
```

Note: The cancelRequestedByCd parameter in API call is the requestedByCd code that the agent selected.

Then, the agent selects the appropriate code.

cancellationType

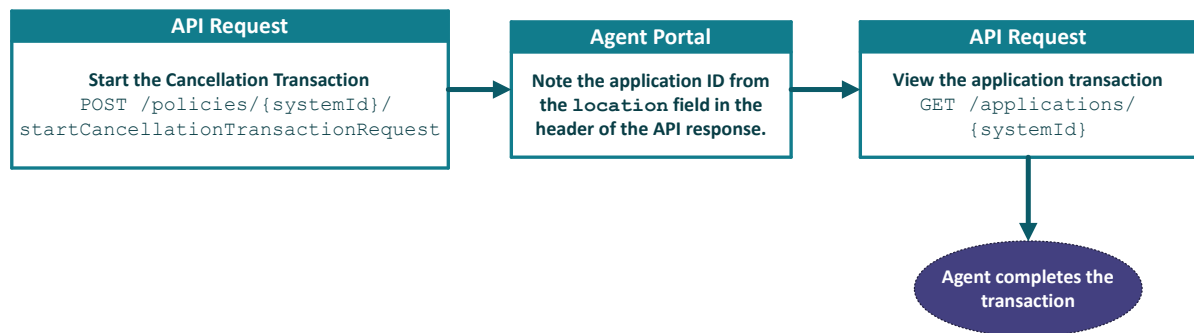
The agent portal calls the following API to determine the cancellation types that are valid for a policy:

```
GET /policies/{systemId}/cancellationType/reasonCd
```

Then, the agent selects the appropriate type.

Submit and process the Cancellation transaction request

Submitting and processing a cancellation starts with the agent portal calling an API to initiate a cancellation request. Then, the system creates an application and the agent processes the application.



For example, the following steps may occur:

1. Start the cancel transaction request:
 - a. The portal executes the following API request:

| | |
|---|---|
| Endpoint | POST /policies/{systemId}/startCancellationTransactionRequest |
| Example Request Body | <pre>{ "transactionCd": "Cancellation", "requestedByCd": "Insured", "reasonCd": "InsuredRequest", "effectiveDate": "2021-05-22", "cancellationType": "Pro-Rate" }</pre> |
| Provide the value of the Cancellation transaction as the transactionCd. | |
| Note: The other values will differ based on agent selections. | |

Note: If the Cancellation transaction follows a Cancellation Notice on the policy, the values provided in the request body of the Cancellation transaction must match those provided in the Cancellation Notice transaction. You can view the details of any existing Cancellation Notice transaction in the API response of GET /policies/{systemId}.

- b. From the API response, note the systemID of the application in the location header:

```
...
location: https://hostname/coreapi/v5/applications/{systemID}
...
```

2. View the new application
 - a. The portal executes the following API call:

```
GET /applications/systemid
```

- b. The portal displays the application details provided in the API response.
 - c. The agent completes the application.

Renew a policy

In an agent portal, agents can renew an existing policy.

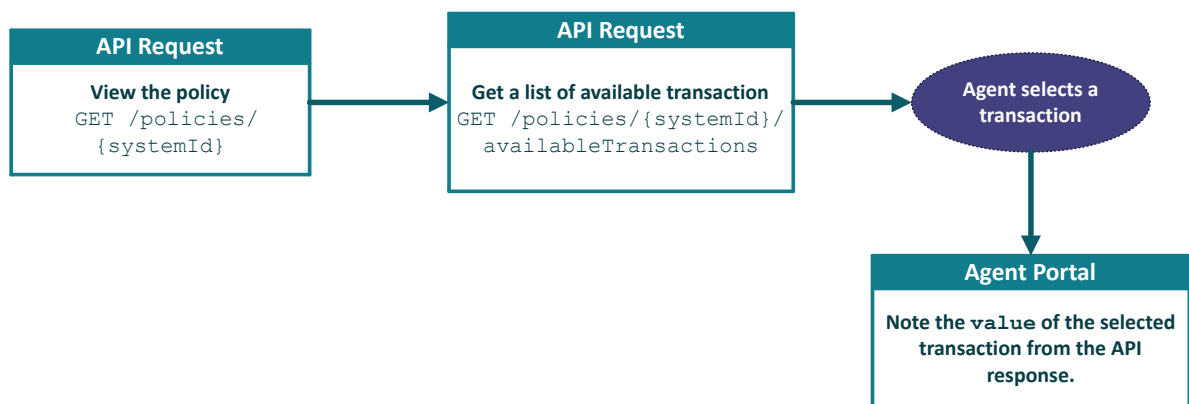
Based on product configuration, agents can directly renew a policy with a Renewal transaction. Or, they can renew a policy with a Renewal Start and Renewal Activate transaction.

Processing a renewal transaction includes the following steps:

1. Identify the policy transaction code.
2. Submit and process the renewal.

Identify the policy transaction code

Initiating a renewal starts with an agent deciding to create a Renewal transaction on a policy. Then, agent portal calls an API to determine the types of transactions that are valid for a policy.



Once an agent selects a Renewal transaction, the agent portal must note the code value of the transaction as it is required to request the transaction in a later step.

1. View the policy details:
 - a. The portal submits the following API call:

```
GET /policies/systemid
```

- b. The portal displays the policy details provided in the response.

2. View the list of available transactions:

- a. The portal submits the following API call:

```
GET /policies/systemid/availableTransactions
```

- b. The portal provides the user a list of available transactions based on the API response.
 - c. The agent selects the Renewal transaction.
 - d. From the API response, the portal notes the value of the Renewal option.

```
...
"value": "Non-Renewal",
"name": "Non-Renewal"
```

```

    },
    {
      "value": "Renewal",
      "name": "Renewal"
    },
    {
      "value": "Rewrite-New",
      "name": "Rewrite-New"
    },
    ...

```

Submit and process the renewal

Submitting and processing the renewal starts with the agent portal calling an API to initiate the renewal. Then, the system creates an application and the agent processes the application.

1. Start the Renewal transaction request:
 - a. The portal gathers renewal details from the agent.
 - b. The portal executes the following API call:

| | |
|----------------------|--|
| Endpoint | POST /policies/ <i>systemId</i> /startTransactionRequest |
| Example Request Body | <pre> { "effectiveDate": "YYYY-MM-DD", "transactionCd": "Renewal", "description": "Renewing for another year" } </pre> |

Note: The values will differ based on agent input.

- c. From the API response header, note the *systemID* of the application in the location field:

```

...
location: https://hostname/coreapi/v5/applications/systemID
...

```

2. View the new application
 - a. The portal executes the following API call:

```
GET /applications/systemid
```

- b. The portal displays the application details provided in the API response.
 - c. The agent completes the application.

Add attachments in an agent portal

In an agent portal, consumers can add attachments to applications and policies.

Adding an attachment to a container, such as an application or policy, is a multistage process:

1. Upload the file.
2. Add an attachment reference from the container to the uploaded file.

Upload the file

Creating an attachment begins with uploading the file to the server using the `multipartfileupload/v2` servlet and then identifying the file name of the document.

For example, the following steps may occur:

1. The agent portal executes the following API call:

| | |
|-----|------------------------------|
| API | POST /multipartfileupload/v2 |
|-----|------------------------------|

Request Body image: *filename*

For example:

| KEY | VALUE | DESCRIPTION |
|---|----------------|-------------|
| <input checked="" type="checkbox"/> image | joystick.png X | |
| Key | Value | Description |

2. From the API response, note the file name:

For example:

```
{
  "files": [
    {
      "name": "joystick.png",
      "size": 4321
      "thumbnailUrl": "innovation?rq=File&Filename=/data/upload/joystick-thumb.png"
    }
  ]
}
```

Identify the attachment template

After an agent identifies a policy or application that requires an attachment, the agent portal calls an API to determine the list of applicable attachment templates. Once the agent selects one of the templates, the agent portal notes the template ID of the attachment as it is required to create the attachment.

For example, the following steps may occur:

1. The portal submits the following API call:

```
GET /attachmentTemplates?container=ContainerType&containerRef=containerSystemID
```

- Application example: GET /attachmentTemplates?container=Application&containerRef=6290
- Policy example: GET /attachmentTemplates?container=Policy&containerRef=484

2. The portal provides the agent a list of available attachment templates based on the API response and the agent selects one.
3. From the API response, the portal notes the id of the attachment that the agent wants to create.

- Application example:

```
{
  "attachmentTemplateListItems": [
    {
      "id": "XApplicationAttachment0015",
      "name": "Signed Application Form",
      "_links": [
        {
          "rel": "self",
          "href": "https://hostname/coreapi/v5/attachmentTemplates/XApplicationAttachment0015?container=Application&containerRef=6290"
        }
      ]
    },
    {
      "id": "PApplicationAttachment0003",
      "name": "Inspection",
      "_links": [
```

```

        {
          "rel": "self",
          "href": "https://hostname/coreapi/v5/attachmentTemplates/PApplicationAttachment0003?
container=Application&containerRef=6290"
        }
      ],
    },
    ...
    {
      "id": "ApplicationAttachment3003",
      "name": "Miscellaneous",
      "_links": [
        {
          "rel": "self",
          "href": "https://hostname/coreapi/v5/attachmentTemplates/ApplicationAttachment3003?
container=Application&containerRef=6290"
        }
      ]
    },
  ],
},

```

- Policy example:

```

{
  "attachmentTemplateListItems": [
    {
      "id": "HOPAPolicyAttachment0001",
      "name": "Proof of Prior Insurance",
      "_links": [
        {
          "rel": "self",
          "href": "https://hostname/coreapi/v5/attachmentTemplates/HOPAPolicyAttachment0001?
container=Policy&containerRef=484"
        }
      ]
    },
    ...
    {
      "id": "PolicyAttachment3003",
      "name": "Miscellaneous",
      "_links": [
        {
          "rel": "self",
          "href": "https://hostname/coreapi/v5/attachmentTemplates/PolicyAttachment3003?
container=Policy&containerRef=484"
        }
      ]
    },
    ...
  ],
}

```

Add an attachment reference

The steps to attach a document differs based on the container or resource type:

Adding an attachment to an application

To add an attachment to an application, execute the following API:

| | |
|----------------------|---|
| Endpoint | POST /applications/ <i>systemId</i> /documents |
| Example Request Body | <pre> { "templateId": "ApplicationAttachment3003", "description": "Picture", "filename": "joystick.png", "memo": "Accident picture" } </pre> <p>Provide the name from the <code>multipfileupload/v2</code> API response as the filename in this request.</p> <p>Note: The values will differ based on agent input.</p> |

Adding an attachment to a policy

| | |
|----------------------|---|
| Endpoint | POST /policies/ <i>systemId</i> /documents |
| Example Request Body | <pre>{ "templateId": "PolicyAttachment3003", "description": "Picture", "filename": "joystick.png", "memo": "Accident picture" }</pre> <p>Provide the name from the multipfileupload/v2 API response as the filename in this request.</p> <p>Note: The values will differ based on agent input.</p> |

