Setting Up the HTTP Service

Last Modified on 08/19/2021 9:57 am EDT

V9.8

Overview

Cora SeQuence now exposes data and services to other applications in the context of dynamic workflows. The engine for the new HTTP listener operates and is deployed as a new WebAPI application.

To design and generate HTTP services, Cora SeQuence uses an open-source tool, Swagger Editor, which supports OpenAPI 3.0. It helps you design and build the APIs, and also document, validate, and consume them. With the embedded Swagger editor, you can write the OpenAPI 3.0 specifications in ready-to-use formats with a defined set of parameters to create standardize HTTP services.

In the new HTTP listener studio you can easily design your CRUD operations (POST, GET, PUT, PATCH, and DELETE) defined in YAML that allows both humans and computers to discover and understand the capabilities of the service without access to the source code.

For more details on OpenAPI specifications and Swagger, refer https://swagger.io/docs/specification/about/.

HTTP listener studio

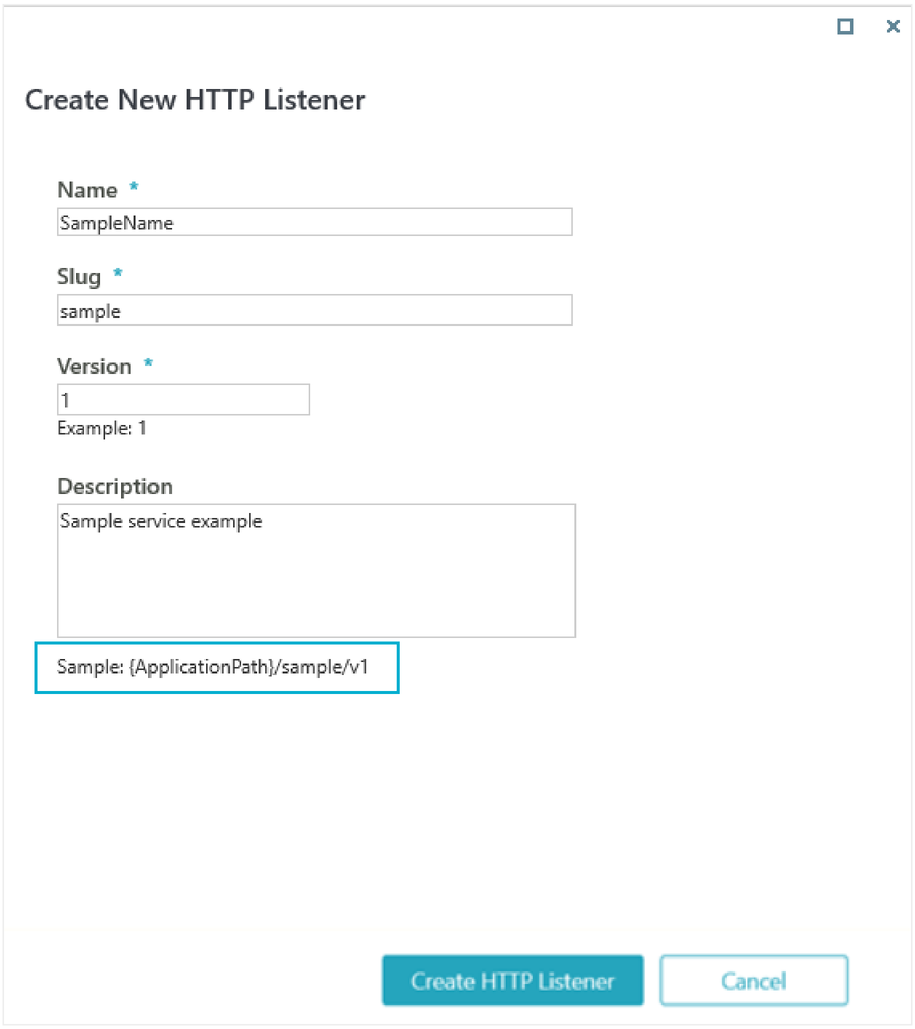
For each HTTP service, you need to create an HTTP listener with the API definition of the service. To create an HTTP listener, and set up an endpoint for HTTP services,

Endpoint properties

1. Go to **Global Settings** > **HTTP Listeners**.
2. Click **+ Add HTTP Listener**.
3. On the Create New HTTP Listener window, add the following information:
    a. Name: Name of the HTTP service. The service name must be significant, and with no special characters.
    b. Slug: exact address (URL) of the endpoint. The slug must be significant, only with lower case alphabets, and with no special characters.
    c. Version: Version of the endpoint. The version must be unique and constructed as a major version only, represented with an integer, (for example 1).

       NOTE
       The combination of slug name and version forms the unique name of each endpoint. An HTTP service may call several endpoints with the same slug name but with different versions, to allow backward compatibility for services in production.

    d. Description: Description of the endpoint.
       See 'Sample' at the bottom of the window to view how your service endpoint will be exposed.


Create New HTTP Listener

Name *
SampleName

Slug *
sample

Version *
1
Example: 1

Description
Sample service example

Sample: {ApplicationPath}/sample/v1

NOTE
The service-based URL contains Slug and Version data. The endpoint parameters are managed in the property window and do not reflect in the service definitions YAML document. Any change in the above service parameter will change the service end-point.

4. Click **Create HTTP Listener**.

HTTP service definitions

After setting and validating the endpoint properties, you can open and edit the HTTP service in the HTTP listener studio i.e. Swagger editor. The studio has two sections, the left one is the YAML code, and the right one provides the display framework of the YAML code and its validations from the left.

The HTTP services in Cora SeQuence use YAML (Yet Another Markup Language) as the data format, which is similar to XML but with less syntax. YAML is a human-readable data language that is used for data storage and transmission in applications.

NOTE
In Cora SeQuence, we have added default tags to guide you through and help you understand the markup structure. Use the structure to define your API definition according to your implementation needs.

For details, refer to https://swagger.io/docs/specification/basic-structure/.

Based on an OpenAPI, the service definitions contain the following root sections:

- Info (required): The info section contains the API information like:
  - title (required): the name of your API documentation.
  - description (optional): the additional information about your API documentation.
  - version (required): the string that specifies the version of the API documentation. *Do not confuse it with the endpoint version.*
    Add the property value in single or double quotes.
- Paths (required): The paths section defines the endpoints in your API, and the operations supported by these endpoints. An operation definition includes parameters, request body (if any), possible response status codes (such as 200 OK or 404 Not Found), and response contents.
  For details, refer to https://swagger.io/docs/specification/paths-and-operations/.

- Components (optional): Different API operations may have some common parameters and may return the same response structure. This may result in code duplication. To avoid this, you can place the common definitions in the Components section and can refer to them using $ref.
  For details, refer to https://swagger.io/docs/specification/components/.

On top of the above mentioned sections, we use the following OpenAPI custom extensions to support the associated dynamic workflows functionality.

Relate an operation to a workflow

- x-operation-handler: This extension defines the operation context of a dynamic workflow (for example, which workflow to start). The extension should be added on the root level of the 'path' section, as shown below.

  ```
  paths:
  …
    x-operation-handler:
      name: ExecuteWorkflow
  ```

  For example, to start a workflow, add the following parameters:

  ```
  x-operation-handler:
    name: ExecuteWorkflow
    workflowSpaceId: '[Space GUID of executed workflow]'
  ```

  To resume a workflow, specify the *workflowInstanceId* or the *activityInstanceId* as a parameter (query/path/body).
  In the following example the parameters were specified as query parameters:

  ```
  parameters:
          - name: workflowInstanceId
            in: query
            schema:
            type: string
  ```

  combined with:

```
x-operation-handler:
  name: ExecuteWorkflow
```

Execute a workflow dynamically

Similar to the web service listener definition, the implementor can configure the HTTP service to execute a template workflow dynamically. You need to provide the workflowSpaceId (a GUID) as part of the URL path or as a query parameter.

As part of the URL path:

```
paths:
  /cases/{workflowSpaceId}:
    post:
      operationId: AddCase
      parameters:
      - name: workflowSpaceId
  in: path
  required: true
  schema:
  type: string
  format: uuid
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/AddCaseRequest'
      responses:
        '201':
          description: 'Case Created.'
      x-operation-handler:
        name: ExecuteWorkflow
      x-untyped-response: true
```

As a query parameter:

```
paths:
  /cases:
    post:
      operationId: AddCase
      parameters:
      - name: workflowSpaceId
  in: query
  required: true
  schema:
  type: string
  format: uuid
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/AddCaseRequest'
      responses:
        '201':
          description: 'Case Created.'
      x-operation-handler:
        name: ExecuteWorkflow
      x-untyped-response: true
```

Error handling

- x-error-catalog: This extension provides error handling infrastructure. It defines a set of errors that may be returned by the service operation (referred to as part of the operation's *x-returned-errors* definition). As part of the workflow design, fault activity can be used to compensate for the defined errors in this section. The extension should be added on the root level of the API definition.

```
x-error-catalog:
  - id: Error1
    code: ErrCode1
    title: 'Error 1 title'
    status: 400
    message: 'Error 1 message: {arg1}, {arg2}.'
    messageParameters:
    - name: arg1
    - name: arg2
```

- x-returned-errors: This section relates between the operation error catalog and a specific error that should appear at the operation level. The extension should be added on the root level of the 'path' section.

```
paths:
…
  responses:
    x-returned-errors:
      - Error1
      - Error2
```

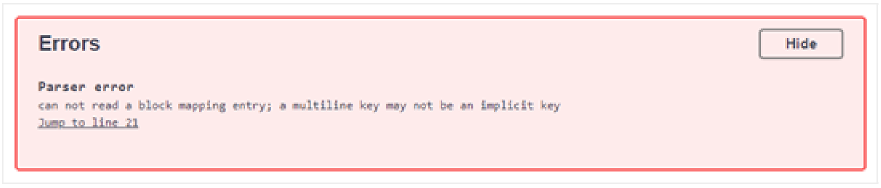
List of errors

The errors are listed alphabetically in the Swagger and the custom errors, defined in the Swagger, are listed first with the prefix PNMsoft.Sequence.WebAPI.

Error type	Error code	Parameters
InvalidArgument	400	IList
Unauthenticated	401	-
Forbidden	403	-
NotFound	404	-
MethodNotAllowed	405	methodName (string)
Aborted	409	-
AlreadyExists	409	-
PreconditionFailed	412	-
PayloadTooLarge	413	-
TooManyRequests	429	-
Unknown	500	-
Internal	500	-

HTTP service definitions validations

The Swagger editor component validates the OpenAPI definitions in real-time with reference to the problematic line in the code and an indicative error description:



```
Parser error
can not read a block mapping entry; a multiline key may not be an implicit key
Jump to line 21
```

Click **Save** to validate the extension’s validation.

Authentication support

Cora SeQuence authenticates at the application level, so you can't configure different authentication methods per service.

For more details on defining authentication in Cora SeQuence, refer to theConfigure Cora SeQuence for SAML 2.0 SSO with OAuth 2.0 for Service-to-Service article.

In-process support

The HTTP services can be consumed by another workflow from the same Cora SeQuence application process by using in-process consumer activity. Unlike WebService listener, for HTTP services there is no need to define the access mode as “In-process” to allow this functionality.

For more details on how to configure In-Process consumer activity, refer to theIn-Process Consumer Activity Overview article.

In case you need to explicitly define an operation as “Private” (that can only be consumed internally), add the following section to the service definitions:

```
x-operation-handler:
…
  AccessMode: InProcess
```

Limitations

- Currently, Cora SeQuence HTTP Listeners do not support messages with multipart content. To define requests or responses with multipart content, use the `untyped` extension, and set it to true. Note that you will need to set up a separate mechanism to handle the content.
- Due to a cross-origin resource sharing (CORS) limitation, client browsers fail to retrieve authorization tokens in environments that use the OAuth2 authorization protocol with the authorization server located on a different domain. To avoid this issue, make sure that the URLs accessed by browsers have a common domain name.

V9.2

Overview

Cora SeQuence now exposes data and services to other applications in the context of dynamic workflows. The engine for the new HTTP listener operates and is deployed as a new WebAPI application.

To design and generate HTTP services, Cora SeQuence uses an open-source tool, Swagger Editor, which supports OpenAPI 3.0. It helps you design and build the APIs, and also document, validate, and consume them. With the embedded Swagger editor, you can write the OpenAPI 3.0 specifications in ready-to-use formats with a defined set of parameters to create standardize HTTP services.

In the new HTTP listener studio you can easily design your CRUD operations (POST, GET, PUT, PATCH, and DELETE) defined in YAML that allows both humans and computers to discover and understand the capabilities of the service without access to the source code.

For more details on OpenAPI specifications and Swagger, refer https://swagger.io/docs/specification/about/.

HTTP listener studio

For each HTTP service, you need to create an HTTP listener with the API definition of the service. To create an HTTP listener, and set up an endpoint for HTTP services,

Endpoint properties

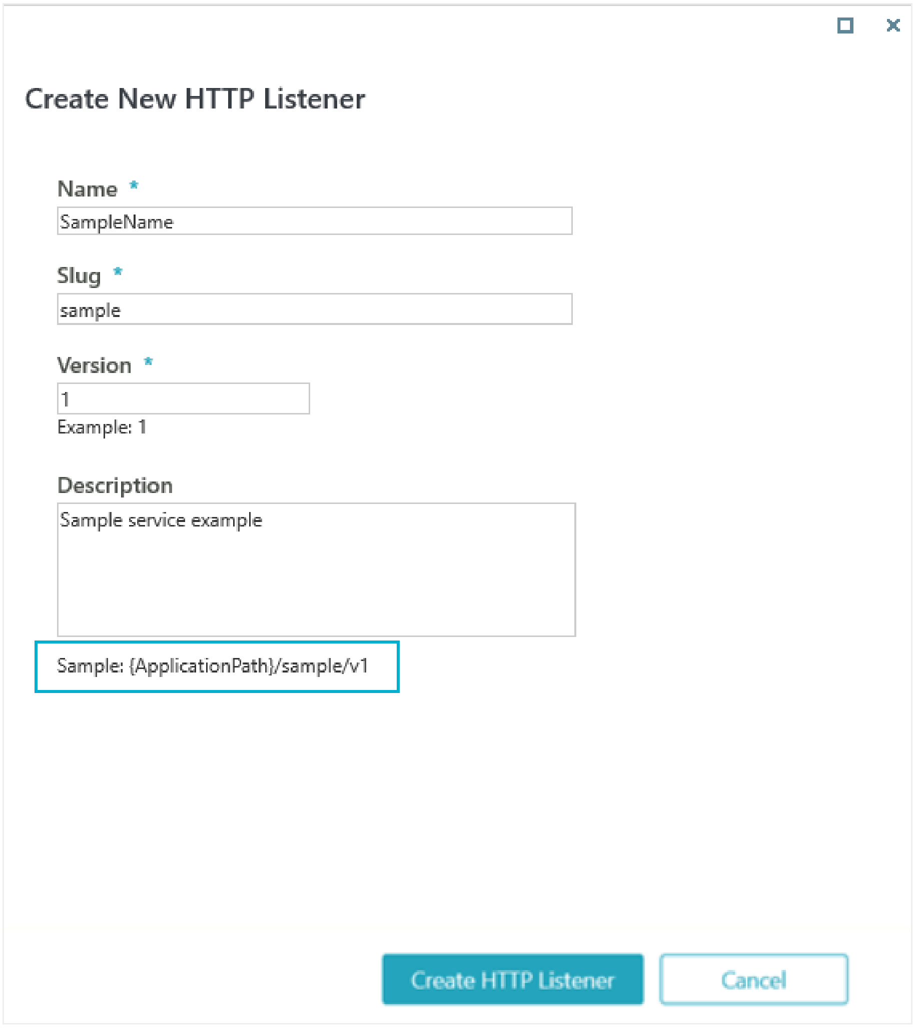
1. Go to Global Settings > HTTP Listeners.
2. Click + Add HTTP Listener.
3. On the Create New HTTP Listener window, add the following information:
    a. Name: Name of the HTTP service. The service name must be significant, and with no special characters.
    b. Slug: exact address (URL) of the endpoint. The slug must be significant, only with lower case

alphabets, and with no special characters.

c. Version: Version of the endpoint. The version must be unique and constructed as a major version only, represented with an integer, (for example 1).

> **NOTE**
> The combination of slug name and version forms the unique name of each endpoint. An HTTP service may call several endpoints with the same slug name but with different versions, to allow backward compatibility for services in production.

d. Description: Description of the endpoint.
See 'Sample' at the bottom of the window to view how your service endpoint will be exposed.


Create New HTTP Listener

Name *
SampleName

Slug *
sample

Version *
1
Example: 1

Description
Sample service example

Sample: {ApplicationPath}/sample/v1

> **NOTE**
> The service-based URL contains Slug and Version data. The endpoint parameters are managed in the property window and do not reflect in the service definitions YAML document. Any change in the above service parameter will change the service end-point.

4. Click **Create HTTP Listener**.

HTTP service definitions

After setting and validating the endpoint properties, you can open and edit the HTTP service in the HTTP listener studio i.e. Swagger editor. The studio has two sections, the left one is the YAML code, and the right one provides the display framework of the YAML code and its validations from the left.

The HTTP services in Cora SeQuence use YAML (Yet Another Markup Language) as the data format, which is similar to XML but with less syntax. YAML is a human-readable data language that is used for data storage and transmission in applications.

NOTE
In Cora SeQuence, we have added default tags to guide you through and help you understand the markup structure. Use the structure to define your API definition according to your implementation needs.
For details, refer to https://swagger.io/docs/specification/basic-structure/.

Based on an OpenAPI, the service definitions contain the following root sections:

- Info (required): The info section contains the API information like:
  - title (required): the name of your API documentation.
  - description (optional): the additional information about your API documentation.
  - version (required): the string that specifies the version of the API documentation. *Do not confuse it with the endpoint version.*
    Add the property value in single or double quotes.
- Paths (required): The paths section defines the endpoints in your API, and the operations supported by these endpoints. An operation definition includes parameters, request body (if any), possible response status codes (such as 200 OK or 404 Not Found), and response contents.
  For details, refer to https://swagger.io/docs/specification/paths-and-operations/.

- Components (optional): Different API operations may have some common parameters and may return the same response structure. This may result in code duplication. To avoid this, you can place the common definitions in the Components section and can refer to them using $ref.
  For details, refer to https://swagger.io/docs/specification/components/.

On top of the above mentioned sections, we use the following OpenAPI custom extensions to support the associated dynamic workflows functionality.

Relate an operation to a workflow

- x-operation-handler: This extension defines the operation context of a dynamic workflow (for example, which workflow to start). The extension should be added on the root level of the 'path' section, as shown below.

```
paths:
…
  x-operation-handler:
    name: ExecuteWorkflow
```

  For example, to start a workflow, add the following parameters:

```
x-operation-handler:
  name: ExecuteWorkflow
  workflowSpaceId: '[Space GUID of executed workflow]'
```

To resume a workflow, specify the *workflowInstanceId* or the *activityInstanceId* as a parameter (query/path/body).
In the following example the parameters were specified as query parameters:

```
parameters:
        - name: workflowInstanceId
          in: query
          schema:
          type: string
```

combined with:

```
x-operation-handler:
  name: ExecuteWorkflow
```

Execute a workflow dynamically

Similar to the web service listener definition, the implementor can configure the HTTP service to execute a template workflow dynamically. You need to provide the workflowSpaceId (a GUID) as part of the URL path or as a query parameter.

As part of the URL path:

```
paths:
  /cases/{workflowSpaceId}:
    post:
      operationId: AddCase
      parameters:
      - name: workflowSpaceId
  in: path
  required: true
  schema:
  type: string
  format: uuid
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/AddCaseRequest'
      responses:
        '201':
          description: 'Case Created.'
      x-operation-handler:
        name: ExecuteWorkflow
      x-untyped-response: true
```

As a query parameter:

```
paths:
  /cases:
    post:
      operationId: AddCase
      parameters:
      - name: workflowSpaceId
  in: query
  required: true
  schema:
  type: string
  format: uuid
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/AddCaseRequest'
      responses:
        '201':
          description: 'Case Created.'
      x-operation-handler:
        name: ExecuteWorkflow
      x-untyped-response: true
```

Error handling

- x-error-catalog: This extension provides error handling infrastructure. It defines a set of errors that may be returned by the service operation (referred to as part of the operation's *x-returned-errors* definition). As part of the workflow design, fault activity can be used to compensate for the defined errors in this section. The extension should be added on the root level of the API definition.

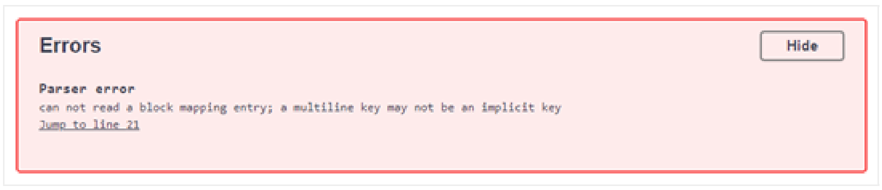
```
x-error-catalog:
  - id: Error1
    code: ErrCode1
    title: 'Error 1 title'
    status: 400
    message: 'Error 1 message: {arg1}, {arg2}.'
    messageParameters:
    - name: arg1
    - name: arg2
```

- x-returned-errors: This section relates between the operation error catalog and a specific error that should appear at the operation level. The extension should be added on the root level of the 'path' section.

```
paths:
…
  responses:
    x-returned-errors:
      - Error1
      - Error2
```

HTTP service definitions validations

The Swagger editor component validates the OpenAPI definitions in real-time with reference to the problematic line in the code and an indicative error description:



Errors

Parser error
can not read a block mapping entry; a multiline key may not be an implicit key
Jump to line 21

Click **Save** to validate the extension’s validation.

Authentication support

Cora SeQuence authenticates at the application level, so you can't configure different authentication methods per service.

For more details on defining authentication in Cora SeQuence, refer to theConfigure Cora SeQuence for SAML 2.0 SSO with OAuth 2.0 for Service-to-Service article.

In-process support

The HTTP services can be consumed by another workflow from the same Cora SeQuence application process by using in-process consumer activity. Unlike WebService listener, for HTTP services there is no need to define the access mode as “In-process” to allow this functionality.

For more details on how to configure In-Process consumer activity, refer to theIn-Process Consumer Activity Overview article.

In case you need to explicitly define an operation as “Private” (that can only be consumed internally), add the following section to the service definitions:

```
x-operation-handler:
…
  AccessMode: InProcess
```

Limitations

- Currently, Cora SeQuence HTTP Listeners do not support messages with multipart content. To define requests or responses with multipart content, use the `untyped` extension, and set it to true. Note that you will need to set up a separate mechanism to handle the content.
- Due to a cross-origin resource sharing (CORS) limitation, client browsers fail to retrieve authorization tokens in environments that use the OAuth2 authorization protocol with the authorization server located on a different domain. To avoid this issue, make sure that the URLs accessed by browsers have a common domain name.