Convenient Interaction with Data Space Integration (Beta)

Integration Package

Table of Contents

Disclaimer

No part of the SAP Licensed End-User Documentation may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

The information contained in the SAP Licensed End-User Documentation may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

SAP and other SAP products and services mentioned in the Licensed End-User Documentation as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies. Please see Trademark Information on SAP.com for additional trademark information and notices.

Introduction

This is the documentation for the Cloud Integration (CI) package “Convenient Interaction with Data Space Integration (Beta)”.

The integration package is used for an easier and more accessible way of interacting with the Data Space Integration capability of the SAP Integration Suite.

The integration package is available on SAP Business Accelerator Hub.

Note that the EDC management API that this integration package uses is still in an experimental beta stage. This is why this package is marked as beta, too.

Used APIs

Data Space Integration

Data Space Integration is a new capability of SAP Integration Suite and enables convenient, secure, and self-sovereign data exchange in data spaces, such as Catena-X.

The current integration package supports Data Space Integration version 1.15.x on the consumer side (based on FOSS Eclipse Dataspace Components 0.7.2 and Tractus-X EDC 0.7.6). The following resources can be used to get information about the Data Space Integration API:

- CX-0018 Dataspace Connectivity 3.1.0
- Data Space Integration API 1.15.x

Known Issues

- Catalog distributions are not supported in the current version of the integration package; the counter party address is always used from the initial request.
- Specifying no data plane request will lead to a simple negotiation:
    - a follow-up request to the Convenient API with an Azure storage push data plane request sub-object will succeed
    - a follow-up request to the Convenient API with an HTTP data plane request sub-object will always lead to a renegotiation, as there was no EDR returned in the first request

Restrictions

As this is a beta version of this integration package, it is not intended to use this for productive data.

Prerequisites

The following products and components are prerequisite for the integration package:

- SAP Integration Suite, with capabilities:
    - Cloud Integration

- Data Space Integration

Overview

The following integration flow is used to negotiate and access the asset:

- Negotiate And Access Asset from Asset Provider

Deployment

Due to security and separation of concerns it is recommended to define separate DSI company policies for each use case. As company policies are bound to the client ID of the use case specific DSI service key which is configured in the integration flow, we must configure the right DSI service key at the integration flow for each use case. The integration flow is then deployed on different use case specific endpoints, each having its on role, which must be configured in the service key for triggering the integration flow.

Request Payload

Following are the properties of the request payload, see also the Example Payloads.

Connectivity, Contract and Negotiation Properties

The connectivity properties are used to identify the asset provider and its data space protocol catalog endpoint. The contract and negotiation properties control how contract agreements are managed.

Property	Description	Example
counterPartyId	The ID of the asset provider.	BPNL1234567890ZZ
counterPartyAddress	The address of the asset provider's connector.	https://edc.provider.com/api/v1/dsp
contractAgreementId	The optional ID of an existing contract agreement to reuse for asset access. Bypasses the catalog request and the negotiation. Can only be set if forceNegotiation is not present or is false.	
forceNegotiation	If true, the contract negotiation is forced even if a contract agreement exists. Can only be set to true if contractAgreementId is not present.	false

Catalog Filter Properties

A list of key value pairs to select the right offer from the catalog of the asset provider. The filter properties are use case specific. The following table shows the currently used filter properties and their values for the Business Partner Data Management use case:

Key (name)	Description	Example Value (value)
dct:type[1]	The mandatory asset type according to CX-0018.	BPDMGate (see CX-0074)
dct:subject[2]	The optional asset subject.	FullAccessGateInputForSharingMember (see CX-0074)
cx-common:version[3]	The mandatory asset version according to CX-0018.	6.0 (see CX-0074)

HTTP Data Plane Request

The sub object (httpDataPlaneRequest) used by the asset consumer to push or pull data to / from the data plane of the asset provider. The following table shows example values for the Business Partner Data Management use case:

Property	Description	Example
httpMethod	The HTTP method according to RFC 2616.	POST
path	The path to the API behind the asset, not incl. query parameters.	/input/changelog/search
query	The optional query parameters to the API behind the asset.	page=0&size=100

HTTP Body in the HTTP Data Plane Request

The sub object (httpBody) which is send to the API behind the asset. The HTTP body is optional, so it can be omitted for GET or POST without body.

Property	Description	Example
characterEncoding	The character encoding of the HTTP body according to ISO/IEC 10646:2020.	UTF-8
contentType	The optional content type for the HTTP body according to RFC 2616.	application/json
bodyAsBytes	The base64-encoded content of the HTTP body. Used for non-JSON content or as alternative to body for JSON content - consumers can choose either format.	

[1] Note that the currently used JSON key by the BPDM use case is "https://purl.org/dc/terms/type"

[2] Note that the currently used JSON key by the BPDM use case is "https://purl.org/dc/terms/subject"

[3] Note that the currently used JSON key by the BPDM use case is "https://w3id.org/catenax/ontology/common/version"

Property	Description	Example
body	The structured JSON content of the HTTP body. Alternative to bodyAsBytes for JSON content - consumers can choose either format.	

Azure Storage Push Data Plane Request

The sub object (azureStoragePushDataPlaneRequest) used by the asset provider to push data to the Azure Storage data plane of the asset consumer.

Property	Description	Example
accountName	The name of the Azure account to which the asset data is pushed. Note that the name of the DSI credential alias, under which the credentials are stored to authenticate against the Azure account, must be '{accountName}-key1', see the DSI documentation.	
containerName	The name of the container in the Azure account to which the asset data (multiple files or single file) is pushed.	
folderName	The optional name of the folder (prefix of the blobs) in the container to which the asset data (multiple files or single file) is pushed. If not specified, asset data is pushed to the container root folder.	

Amazon S3 Push Data Plane Request

Note that the Amazon S3 data plane request is currently not supported by the integration flow.

Response Payload

Following are the properties of the response payload, see also the Example Payloads.

General Properties

Property	Description	Example
statusCode	The HTTP status code according to RFC 2616.	200
statusText	The HTTP status text according to RFC 2616.	OK
errorMessage	The error message in case of a status code with 4xx or 5xx.	
assetId	The ID of the asset, if a corresponding offer has been found in the catalog.	
offerId	The ID of the offer, if an offer has been found in the catalog.	
contractNegotiationId	The ID of the contract negotiation, if a contract negotiation has been initiated in the current request or had been finalized in a previous request.	

Property	Description	Example
contractAgreementId	The contract agreement id of the concluded contract.	
transferProcessId	The ID of the transfer process, if a transfer process has been initiated.	
transferProcessState	The state of the transfer process by the end of the integration flow, if a transfer process has been initiated explicitly for Azure Storage .	COMPLETED

The following HTTP status codes are returned in case of an error:

- 404 - When no offer is found, the API returns 'The offer you were looking for does not exist in the catalog.'
- 400 - When too many offers are found, the API returns 'Your request was invalid. Too many offers match the selected filter criteria.'

HTTP Body from the HTTP Data Plane Specific Response

A sub object (httpBody) returned from the API behind the asset:

Property	Description	Example
characterEncoding	The character encoding of the body according to ISO/IEC 10646:2020.	UTF-8
contentType	The content type which the consumer can expect in the body according to RFC 2616.	application/json
bodyAsBytes	The base64-encoded content of the HTTP body. Used for non-JSON content or as alternative to body for JSON content.	
body	The structured JSON content of the HTTP body. Alternative to bodyAsBytes for JSON content.	

Response Body Format Evaluation:

- **UsePlainJsonBodyInResponseUseCase# = false (default):** Always returns bodyAsBytes with Base64-encoded content
- **UsePlainJsonBodyInResponseUseCase# = true:** Returns body with structured JSON for application/json content types, bodyAsBytes for other content types
- **Configuration:** Each use case has its own property and can be configured independently

Process

Negotiate the Asset

With the negotiate and access asset request, the asset provider and its data space protocol catalog endpoint are uniquely identified (counterPartyId and counterPartyAddress).

The control plane is used to get the right offer from the catalog of the asset provider based on the filter properties (catalogFilterProperties) sent with the request. Note that the offer is additionally filtered based on the consumer / provider usage policies (in Data Space Integration) and the consumer BPNL, if an access policy for the consumer is in place at the provider. Note also that the list of filter properties is use case specific. If not exactly one offer is returned for these criteria, the integration flow will fail and return an error message.

When the contract agreement ID (contractAgreementId) is provided in the request: The integration flow skips the catalog request and offer determination described above, directly attempting to use the specified contract agreement for asset access. For HTTP data transfers, if no valid EDR (Endpoint Data Reference) exists for the contract agreement (see also below), the integration flow does perform the catalog request eventually to get the current offer and automatically triggers a new contract negotiation, returning the new contract agreement id in the response. For Azure Storage data transfers, no validation is performed (see also below).

When the contract agreement ID is not provided: After the catalog request and offer determination, the integration flow gets the latest contract agreement for the offered asset, if any exists. Note that multiple contract agreements can exist for the same asset, but only the latest one is used, even if the contract policies of the current offer do not match the contract policies of the contract agreement. So, we assume that the contract for the requested asset was concluded by both parties on the base of the then valid contract policies / offer.

When the contract negotiation is forced (forceNegotiation): A new contract negotiation is always triggered, regardless of whether a contract agreement exists or was provided. The catalog request is performed to get the current offer before the contract negotiation. Note that the integration flow currently does not check whether the contract agreement itself is still valid, as the validity of a contract agreement is currently not provided by the Data Space Protocol (DSP). The assumption here is that the provider will not start the transfer process (also for the EDR) if the contract agreement is not valid anymore. In this case a forced contract negotiation (see forceNegotiation in the request payload) must be triggered by the sender of the “Negotiate and Access Asset” request.

For HTTP data transfers and if a contract agreement exists, it is checked based on the contract agreement id and the provider BPNL, whether there is a valid endpoint data reference (EDR) with a short-lived token cached. If a cached EDR is not returned, the next step is to initiate an EDR contract negotiation of the data exchange contract based on the ID and the provider usage policy of the offer. Once the contract negotiation is finalized and the contract agreement has been returned, it is again checked, whether there is now a valid endpoint data reference (EDR) with a short-lived token cached.

For Azure Storage data transfers, we do not use the endpoint data reference (EDR) mechanism. So, if there is no contract agreement (provided or offer-based), the integration flow will initiate the simple contract negotiation of the data exchange contract based on the ID and the provider usage policy of the offer. Note that the transfer process may fail if the provider has canceled the contract agreement on its side.

Access the Asset

For HTTP data transfers, the short-lived token from the EDR is used to authenticate and communicate with the API behind the asset (data plane) using the HTTP parameters from the request (httpMethod, path, query, httpBody). Once processing on provider side has finished, the response is returned to the sender of the “Negotiate and Access Asset” request (statusCode and httpBody).

For Azure Storage data transfers, the integration flow directly initiates the transfer process using the Azure parameters from the request (accountName, containerName, folderName), so that the provider can push the data to the Azure Storage data plane of the asset consumer. If the transfer process is started successfully, the response contains the transfer process parameters (transferProcessId and transferProcessState). If the transfer process is not started successfully, the integration flow will fail and return an error message. After the transfer process is started, the integration flow will return to the sender; any state after and including STARTED may be returned, depending on the progress of the transfer process. Explicit polling for COMPLETED is not performed. The sender of the “Negotiate and Access Asset” request can then poll for the transfer process state itself, if needed, using the DSI management API or poll for the file storage location of the transferred file.

Configuration Steps

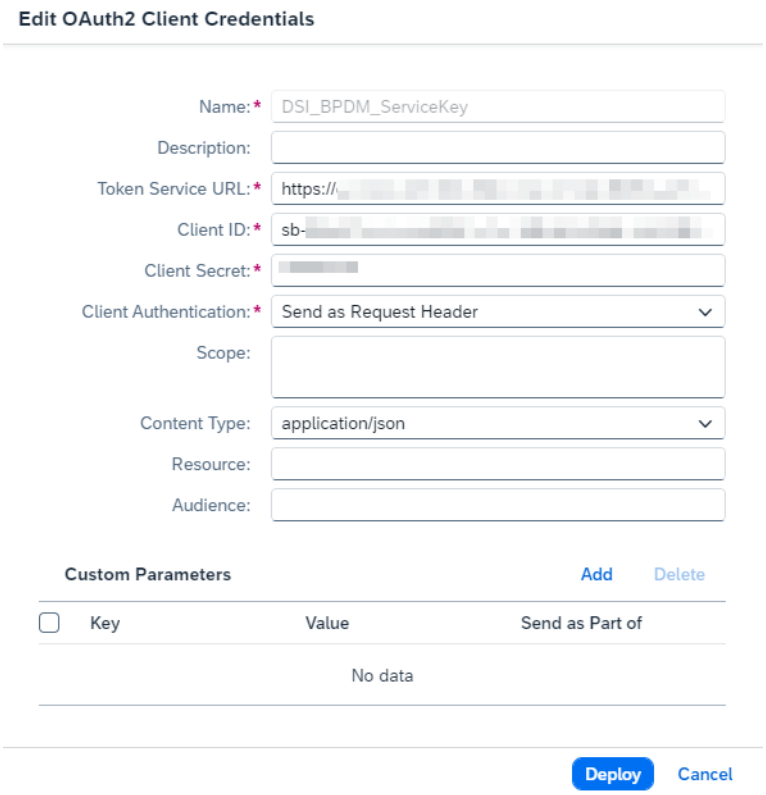
The following steps describe, how to setup the integration between an SAP application and Data Space Integration.

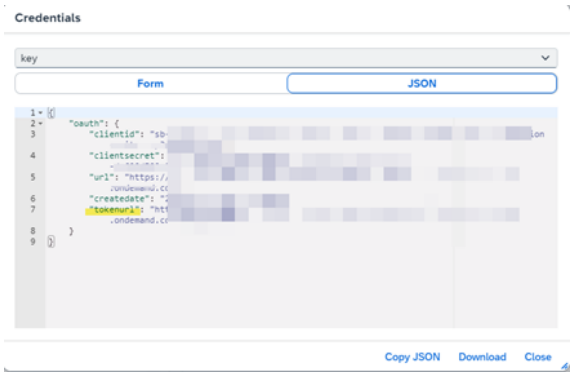
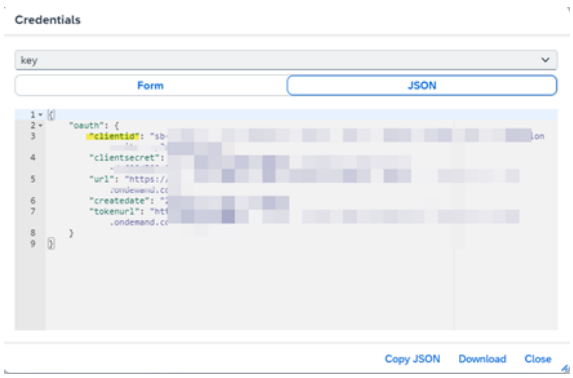
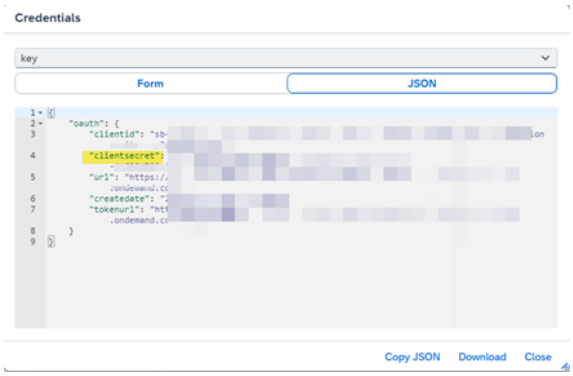
Data Space Integration

Information about the initial setup of Data Space Integration can be found on SAP Help.

Accessing DSI from CI

- In SAP BTP Cockpit, create a dedicated service key in the corresponding subaccount (see Using APIs To Work With Data Space Integration | SAP Help Portal and Creating Service Instance and Service Key for Inbound Authentication | SAP Help Portal) of type “OAuth2 ClientId/Secret” for a new or existing service instance “Data Space Integration API access”, the plan “api” and at least role “AuthGroup_DataspaceConsumer”; to create service instance and keys you need the following prerequisites:
    - Subaccount admin role
    - Cloud Foundry organization member
    - Assignment to Cloud Foundry space
- In SAP Integration Suite under “Security Material”, create a new security material for CI (see Managing Security Material | SAP Help Portal) of type “OAuth2 Client Credentials” with a name, containing the use case name.



Parameter Name	Parameter Value
Name	e.g. DSI_{UseCaseName}_ServiceKey; replace {UseCaseName} with the name of your use case.
Token Service URL	 Paste here value from parameter “tokenurl” from the service key configured in the step above
Client ID	 Paste here value from parameter “clientid” from the service key configured in the step above
Client Secret	 Paste here value from parameter “clientsecret” from the service key configured in the step above
Client Authentication	Send as Request Header
Content Type	application/json

Use this security material for authenticating the “Negotiate and Access Asset from Asset Provider” integration flow for communication with Data Space Integration. If you use the integration flow for different use cases, you should create multiple service keys and corresponding security materials, one for each use case, so that the right company policy is applied for each use case.

Add and Assign Company Policy

Add new company policy corresponding to your use case (see Create and Edit Company Policies | SAP Help Portal). You can reuse an existing company policy template. Assign the client id from the step before to the company policy (see Assign Company Policies | SAP Help Portal). If you use the integration flow for different use cases, you can create multiple company policies, one for each use case.

Cloud Integration

Information about the initial setup of Cloud Integration can be found on SAP Help.

Accessing the Integration Flow on CI

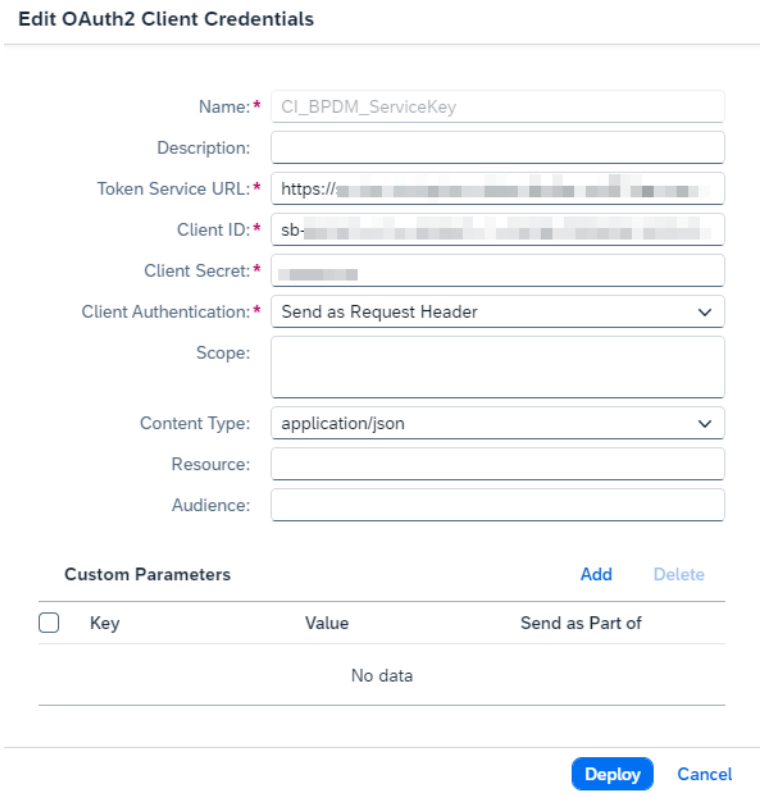
- In SAP BTP Cockpit, create a dedicated service key in the corresponding subaccount (see Creating Service Instance and Service Key for Inbound Authentication | SAP Help Portal) of type “OAuth2 ClientId/Secret” for a new or existing service instance “Process Integration Runtime”, the plan “integration-flow” and at least role “ESBMessaging.send”; to create service instance and keys you need the following prerequisites:
    - Subaccount admin role
    - Cloud Foundry organization member
    - Assignment to Cloud Foundry space

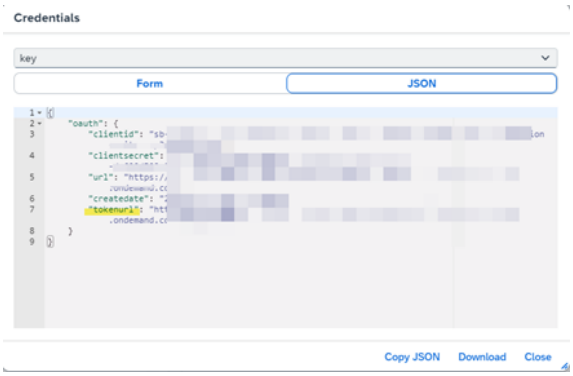
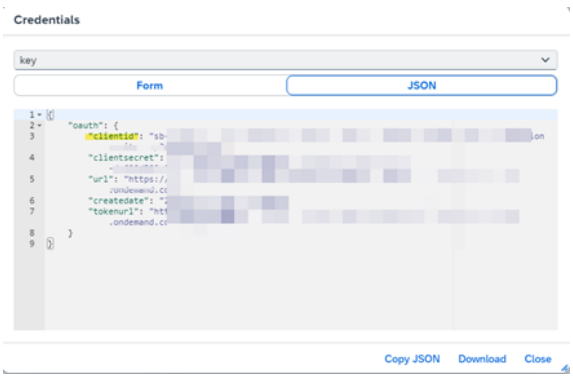
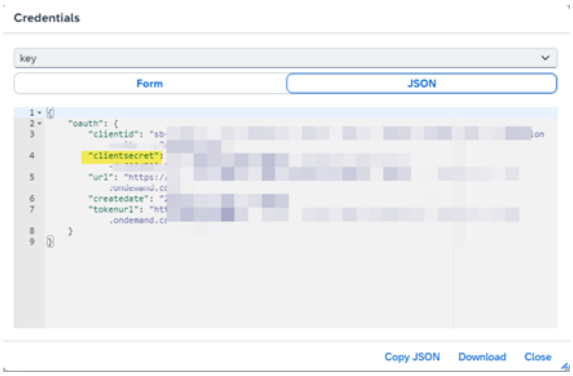
Use this service key for authenticating your application for communication with the “Negotiate and Access Asset from Asset Provider” integration flow. If you use the integration flow for different use cases, you can create multiple service keys, one for each use case.

Communication between Integration Flows on CI

If the integration flow is called by another integration flow, you need to configure the following:

- In SAP Integration Suite under “Security Material”, create a new security material for CI (see Managing Security Material | SAP Help Portal) of type “OAuth2 Client Credentials” with a name, containing the use case name.



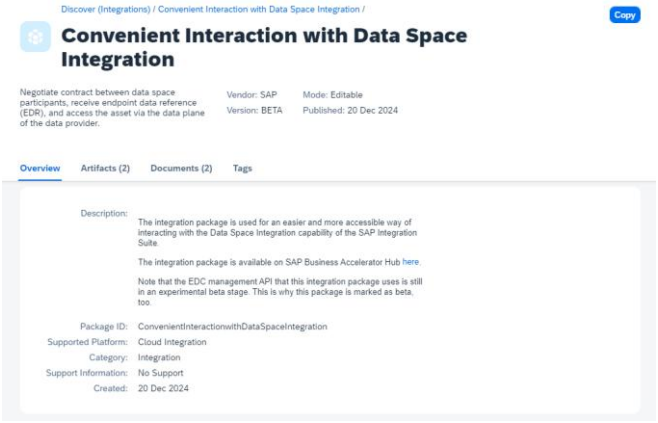
Parameter Name	Parameter Value
Name	e.g. CI_{UseCaseName}ServiceKey; replace {UseCaseName} with the name of your use case.
Token Service URL	 Paste here value from parameter “tokenurl” from the service key configured in the step above
Client ID	 Paste here value from parameter “clientid” from the service key configured in the step above
Client Secret	 Paste here value from parameter “clientsecret” from the service key configured in the step above
Client Authentication	Send as Request Header
Content Type	application/json

Use this security material for authenticating your integration flow for communication with the “Negotiate and Access Asset from Asset Provider” integration flow. If you use the integration flow for different use cases, you can create multiple security materials, one for each use case.

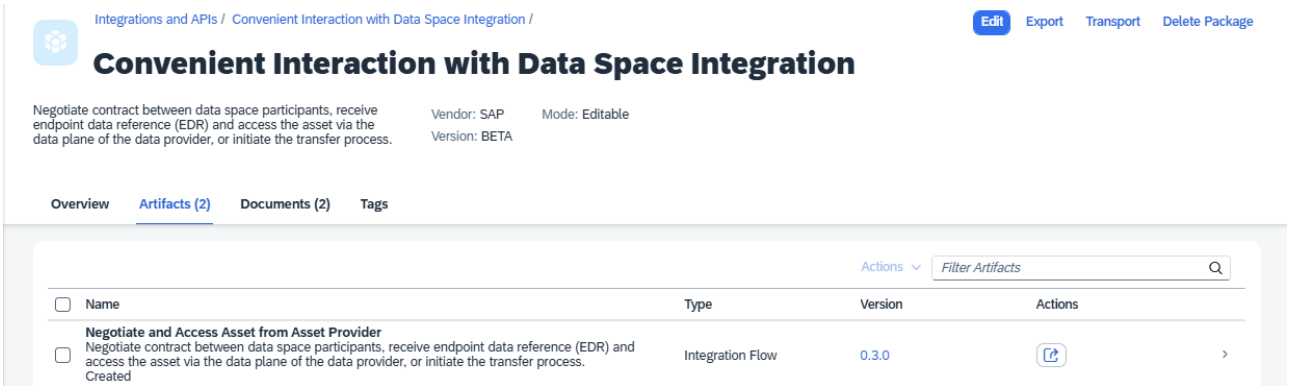
Integration Package

1. Connect to SAP Integration Suite
2. Go to **Discover (Integrations)** view.

3. Search for the package **Convenient Interaction with Data Space Integration (Beta)**.
4. Select the package and copy it to your **Design** workspace by clicking the **Copy** button in the right corner.



5. After copying go to **Design** (Integrations and APIs) open the package **Convenient Interaction with Data Space Integration (Beta)**

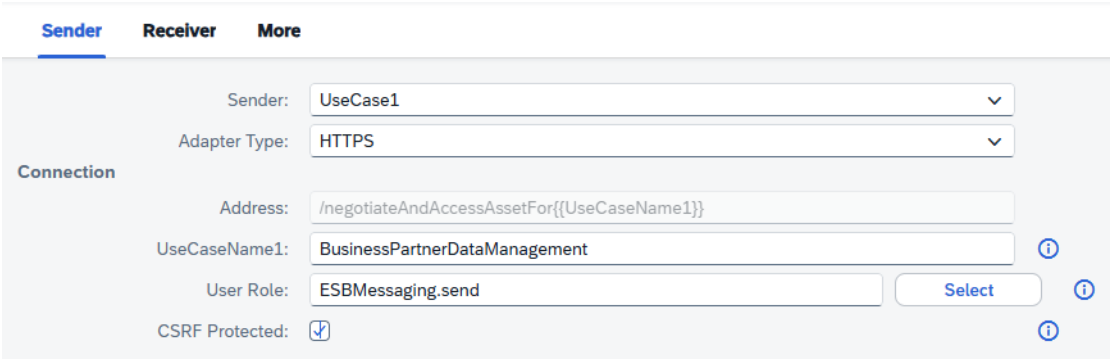


6. The integration flow(s) and the script collection(s) will be shown / listed under the **Artifacts** section of the page.
7. Select the integration flow **Negotiate and Access Asset from Asset Provider** described in the table below, choose the **Actions** button on the right side and select **Configure**.

Negotiate and Access Asset from Asset Provider

Sender Use Case

The Use Case # sender adapter exposes a hardcoded (but partially configurable) HTTPS endpoint "/negotiateAndAccessAssetFor{{UseCaseName#}}" through which a calling external integration flow or application can negotiate and access an asset for a specific use case. Currently, at total of 3 use cases are supported, which are enumerated by the appended integer number in the sender adapter name. The use case name is used to differentiate between different use cases in the integration flow. For each use case, a separate HTTPS endpoint is created, which can be used by the sender system to send messages to the integration flow.



Parameter	Description
Use Case Name #	The name of the use case that shares data via a data space. The name is appended to the HTTPS endpoint.
User Role	Choose **Select** to get a list of all available roles and select one user role per use case. You can use the role **ESBMessaging.send** .[4]
CSRF Protected	Tick the check box to enable cross-site request forgery attack protection. [5]

Receiver Connector Use Case

Via the Connector Use Case # receiver adapter the following messages are sent to the control plane / management API of DSI for a specific use case (set dynamically by the integration flow using ${exchangeProperty.connectorPath}):

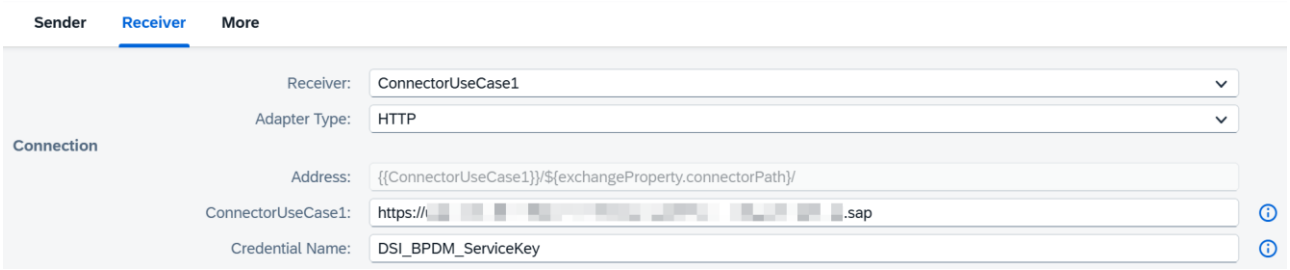
- Get Catalog (POST /api/management/v2/catalog/request)
- Get Contract Agreement (POST /api/management/v2/contractagreements/request)
- Get / Poll for EDR (POST /api/management/v2/edrs/request)

[4] It is a predefined role provided by SAP that authorizes a sender system to process messages on a tenant. However, using SAP BTP Cockpit, you can also define custom roles for the runtime node as well (we recommend the creation of an individual sender role per use case). When you choose **Select**, a selection of all custom roles defined that way is offered.

[5] This is recommended if the integration flow is called by simple browser-based applications, see the documentation.

- Initiate (Simple) Negotiation (POST /api/management/v2/contractnegotiations) -or-
- Initiate EDR Negotiation (POST /api/management/v2/edrs)
- Get / Poll for Contract Negotiation (State) (GET /api/management/v2/contractnegotiations/${property.contractNegotiationId})
- Get / Poll for Data Address (GET /api/management/v2/edrs/${property.transferProcessId}/dataaddress?auto_refresh=true)
- Initiate Transfer Process (POST /api/management/v2/transferprocesses)
- Get / Poll for Transfer Process (State) (GET /api/management/v2/transferprocesses/${property.transferProcessId})

Note that the receiver adapter is configured to communicate with the DSI control plane / management API for a specific use case. So, for each use case, a separate DSI instance can be used. Moreover, the receiver adapter is configured to use a specific security material, so that the use case specific company policy can be applied.



Parameter	Description
Connector Use Case #	The URI of DSI for the use case #, in the form <protocol>://<host>.
Credential Name	The name of the security material, containing the user credentials used for Accessing DSI from CI for use case #.

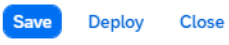
Additional Parameters

Parameter	Description
Contract Negotiation Polling Interval	Time interval in milliseconds for polling **Get Contract Negotiation** for FINALIZED after **Initiate EDR Negotiation**. Default value is configured to 3000.

Parameter	Description
Data Address Polling Interval	Time interval in milliseconds for polling **Get Data Address** for a valid EDR token after **Get EDR**. Default value is configured to 1000.
Edr Token Minimum Validity	Minimum validity of an EDR token in milliseconds for being used in **Access Asset**. Default value is configured to 5000.
Edr Token Polling Interval	Time interval in milliseconds for polling **Get EDR** after **Get Contract Negotiation** is FINALIZED. Default value is configured to 3000.
Transfer Process Polling Interval	Time interval in milliseconds for polling **Get Transfer Process** for STARTED and COMPLETED after **Initiate Transfer Process** for Azure Storage transfers. Default value is configured to 3000.
Use Plain Json Body In Response Use Case #	Controls response body format for Use Case #.

Note that the maximum number of polling iterations is set to 20 in all polling processes. This means that the maximum time for polling is 20 times the polling interval.

Note that for Azure Storage transfers, the integration flow successfully returns to the sender of the “Negotiate and Access Asset” request if a transfer process has been successfully STARTED, regardless of file size. This way it is ensured that the sender can continue processing even if the transfer process is still in progress. The sender of the “Negotiate and Access Asset” request can then poll for the transfer process state itself, if needed, using the DSI management API or poll for the file storage location of the transferred file. Explicit polling for COMPLETED is not performed by the integration flow.

After configuration you can deploy the integration flow. 

Appendix

Example Payloads

Body Format

The integration flow supports both Base64 (bodyAsBytes) and structured JSON (body) formats for HTTP bodies.

Request: Consumers can send either body (structured JSON) or bodyAsBytes (Base64) for application/json content. Non-JSON content always uses bodyAsBytes.

Response: Controlled by per-use-case integration flow configuration properties UsePlainJsonBodyInResponseUseCase1, UsePlainJsonBodyInResponseUseCase2, UsePlainJsonBodyInResponseUseCase3 (default: false):

- **UsePlainJsonBodyInResponseUseCase# = false:** Always return bodyAsBytes (Base64-encoded content)
- **UsePlainJsonBodyInResponseUseCase# = true:** Return body (structured JSON) for application/json content, bodyAsBytes for other content types

Use Case Configuration: Each use case can be configured independently with its own property (UseCase1, UseCase2, UseCase3) to support different response formats based on their specific requirements.

Request for HTTP Data Plane (Base64 Body)

```
{
  "counterPartyId": "{{PROVIDER_BPNL}}",
  "counterPartyAddress": "{{PROVIDER_CONNECTOR_DATASPACE_API}}",
  "catalogFilterProperties": [
    {
      "name": "{{ASSET_TYPE_NAME}}",
      "value": "{{ASSET_TYPE_VALUE}}"
    },
    {
      "name": "{{ASSET_SUBJECT_NAME}}",
      "value": "{{ASSET_SUBJECT_VALUE}}"
    },
    {
      "name": "{{ASSET_VERSION_NAME}}",
      "value": "{{ASSET_VERSION_VALUE}}"
    }
  ],
  "forceNegotiation": false,
  "contractAgreementId": "{{CONTRACT_AGREEMENT_ID}}",
  "httpDataPlaneRequest": {
    "httpMethod": "{{HTTP_METHOD}}",
```

```
    "path": "{{PATH}}",
    "query": "{{QUERY}}",
    "httpBody": {
      "characterEncoding": "{{CHARACTER_ENCODING}}",
      "contentType": "{{CONTENT_TYPE}}",
      "bodyAsBytes": "{{BASE64_ENCODED_BODY}}"
    }
  }
}
```

Request for HTTP Data Plane (Plain JSON Body)

```
{
  "counterPartyId": "{{PROVIDER_BPNL}}",
  "counterPartyAddress": "{{PROVIDER_CONNECTOR_DATASPACE_API}}",
  "catalogFilterProperties": [
    {
      "name": "{{ASSET_TYPE_NAME}}",
      "value": "{{ASSET_TYPE_VALUE}}"
    },
    {
      "name": "{{ASSET_SUBJECT_NAME}}",
      "value": "{{ASSET_SUBJECT_VALUE}}"
    },
    {
      "name": "{{ASSET_VERSION_NAME}}",
      "value": "{{ASSET_VERSION_VALUE}}"
    }
  ],
  "forceNegotiation": false,
  "contractAgreementId": "{{CONTRACT_AGREEMENT_ID}}",
  "httpDataPlaneRequest": {
    "httpMethod": "{{HTTP_METHOD}}",
    "path": "{{PATH}}",
    "query": "{{QUERY}}",
    "httpBody": {
      "characterEncoding": "{{CHARACTER_ENCODING}}",
      "contentType": "{{CONTENT_TYPE}}",
      "body": {
        "my": "structured",
        "json": "object"
      }
    }
  }
}
```

Request for Azure Storage Push Data Plane

```
{
  "counterPartyId": "{{PROVIDER_BPNL}}",
  "counterPartyAddress": "{{PROVIDER_CONNECTOR_DATASPACE_API}}",
  "catalogFilterProperties": [
    {
      "name": "{{ASSET_TYPE_NAME}}",
      "value": "{{ASSET_TYPE_VALUE}}"
    },
    {
      "name": "{{ASSET_SUBJECT_NAME}}",
      "value": "{{ASSET_SUBJECT_VALUE}}"
    },
    {
      "name": "{{ASSET_VERSION_NAME}}",
      "value": "{{ASSET_VERSION_VALUE}}"
    }
  ],
  "forceNegotiation": false,
  "contractAgreementId": "{{CONTRACT_AGREEMENT_ID}}",
  "azureStoragePushDataPlaneRequest": {
    "accountName": "{{AZURE_STORAGE_PUSH_ACCOUNT_NAME}}",
    "containerName": "{{AZURE_STORAGE_PUSH_CONTAINER_NAME}}",
    "folderName": "{{AZURE_STORAGE_PUSH_FOLDER_NAME}}"
  }
}
```

Response for HTTP Data Plane (Base64 Body)

```
{
  "statusCode": "{{STATUS_CODE}}",
  "statusText": "{{STATUS_TEXT}}",
  "errorMessage": "{{ERROR_MESSAGE}}",
  "assetId": "{{ASSET_ID}}",
  "offerId": "{{OFFER_ID}}",
  "contractNegotiationId": "{{CONTRACT_NEGOTIATION_ID}}",
  "contractAgreementId": "{{CONTRACT_AGREEMENT_ID}}",
  "transferProcessId": "{{TRANSFER_PROCESS_ID}}",
  "httpBody": {
    "characterEncoding": "{{CHARACTER_ENCODING}}",
    "contentType": "{{CONTENT_TYPE}}",
    "bodyAsBytes": "{{BASE64_ENCODED_BODY}}"
  }
}
```

Response for HTTP Data Plane (Plain JSON Body)

```
{
  "statusCode": "{{STATUS_CODE}}",
  "statusText": "{{STATUS_TEXT}}",
  "errorMessage": "{{ERROR_MESSAGE}}",
  "assetId": "{{ASSET_ID}}",
  "offerId": "{{OFFER_ID}}",
  "contractNegotiationId": "{{CONTRACT_NEGOTIATION_ID}}",
  "contractAgreementId": "{{CONTRACT_AGREEMENT_ID}}",
  "transferProcessId": "{{TRANSFER_PROCESS_ID}}",
  "httpBody": {
    "characterEncoding": "{{CHARACTER_ENCODING}}",
    "contentType": "{{CONTENT_TYPE}}",
    "body": {
      "my": "structured",
      "json": "object"
    }
  }
}
```

Response for Azure Storage Push Data Plane

```
{
  "statusCode": "{{STATUS_CODE}}",
  "statusText": "{{STATUS_TEXT}}",
  "errorMessage": "{{ERROR_MESSAGE}}",
  "assetId": "{{ASSET_ID}}",
  "offerId": "{{OFFER_ID}}",
  "contractNegotiationId": "{{CONTRACT_NEGOTIATION_ID}}",
  "contractAgreementId": "{{CONTRACT_AGREEMENT_ID}}",
  "transferProcessId": "{{TRANSFER_PROCESS_ID}}",
  "transferProcessState": "{{TRANSFER_PROCESS_STATE}}"
}
```