Appendix G – Connector Development Kit

1. Introduction

This guide aims to guide through the different steps of developing a **Connector**. Before getting started with this guide, make sure your chosen process and data source are suitable for a Process Mining implementation. Process Mining requires an activity to define the performed events, a timestamp to show when each event happened, and an identifier to tie all events in one process execution. Without one of these three mandatory elements, a Process Mining analysis is not possible. Therefore, it is important to know the tables and fields of your data before getting started.

Data transformation

UiPath Process Mining is a solution that transforms data from your IT systems into visual interactive dashboards, allowing you to see existing value decreases, bottlenecks, and discrepancies, as well as understanding the root-causes and possible risks. The Process Mining developer has to execute several steps for these dashboards to be available.

Many of these steps are part of the **data transformation** process. This transformation is necessary since UiPath Process Mining requires the data in a certain format to be able to visualize it. This format is referred to as the **data model**. The data model describes the tables and attributes that need to be part of the output dataset generated by the connector. However, most source systems do not provide data directly in the required format. Therefore, it is necessary to transform the data.

Connectors

To help with the data transformation, UiPath Process Mining has created several connectors which can be used as template for loading, cleaning, and transforming data. These pre-built connectors are specific for a certain system and process such as the SAP Connector for Purchase-to-Pay Discovery Accelerator.

When starting with a new project, we recommend you check whether you can make use of UiPath's pre-built connectors**.** In the simplest cases, the input data fits directly with one of the connectors and there is no need for a custom connector. If none of the connector can be used, you need to build one yourself.

Building a Connector

The goal of a connector is to produce a dataset that fits the data model. The most standard Process Mining data model consists of a **Cases** and an **Event log** table. Alternative data models consist of several **Entity** and **Events** tables. Independent of the data model, we recommend structuring the first transformation steps of the connector as follows:

1. **Input**: define the raw data from the source system in the connector.
2. **Entities**: define the entities relevant to the process.
3. **Events**: define the events that happen for the entities.

In the scenario that the Event log, Tags, or Due dates are part of the data model, the following two transformation steps follow:

4. **Event log**: combine events for different entities in one event log.
5. **Business data**: define tags and due dates.

The last transformation step in any connector is defining the output that corresponds to the data model.

6. **Output**: transformed data that fits the data model.

About this guide

This guide leads you through the different transformation steps and provides you with conceptual explanations as well as tips and examples for the actual implementation. The guide is written in a generic way so that you can apply it to any kind of process and system. To successfully follow this guide, make sure you are familiar with your input data and the data model.

2. Input

The first group of any connector is the group of input tables. In this group, we define the raw data from the source system. This is the only group in which there is a dependency on how the data is stored in the source system. Later transformations are built on top of these input tables. The input tables are therefore the best place to filter on required data and to cast attributes to the correct data types.

Tip: we advise to always apply the filtering that is required for correctly transforming the data. Also in the scenario that filtering is already applied when extracting the data. In this way, the connector can more easily be reused when other extraction methods are used.

The first step when building a connector is to consider the input data and the data model we want to transform the data to. In general, there are two types of input data which are important for Process Mining:

- **Transactional data**: contains the mandatory data for Process Mining, used for generating the process graph and most other views.
- **Master data:** additional information and context about the transactional data.

Master data is usually quite straightforward, while the transactional data can have many different formats. However, almost all formats of transactional data can be brought down to the following two options:

- One or more tables with multiple timestamp columns.
- One or more tables with one timestamp column (transaction log).

Compare the two formats below in which we show an example of invoice data.

Table with multiple timestamp columns:

Invoice ID	Creation time	Deletion time	Price changed time
1001	Jan 15, 2020	Jan 17, 2020	NULL
1002	Feb 01, 2020	NULL	Feb 02, 2020

Each record in this table represents one invoice and the columns contain the information about when a certain event took place. This way of data storage has its limitations since a column will only allow for one value and typically these values will be overridden when a field is updated. A lot of useful Process Mining information may be lost. For example, the price on an invoice can be changed multiple times, but only the last price change is available in the data. In this way, information about previous price changes is lost.

Transaction log table:

Invoice ID	Transaction time	Transaction description
1001	Jan 15, 2020	Creation
1001	Jan 17, 2020	Deletion
1002	Feb 01, 2020	Creation
1002	Feb 02, 2020	Price changed

Each record in this table represents one event for a certain invoice. One column indicates when an event took place and another column indicates which event took place.

Define input tables

It is good practice to consider the input format of the tables in the source system already before you continue with the next transformation steps. This will help you in understanding which input tables are required and which transformations are needed to transform the input to the data model.

Define the input tables that are needed and group these tables together in a logical way. This improves the readability of the connector and makes it easier to maintain it. Think about which attributes should be available on the input tables. Often your data will contain attributes that are not needed, which you can exclude from the connector.

Tip: rename the attributes as available in the source system to human readable names.

It is important to check the data types for the input attributes. For most attributes this will be text, but they can also be numeric values or dates. Common issues with the data types are:

- Text fields consisting of numbers with leading zeroes (00348) can be recognized as doubles or integers while it should be text. These leading zeroes are removed when these attributes are cast to a numeric data type, which can lead to issues when joining tables.
- Dates and datetimes being recognized as integers or text. This can lead to problems when trying to set up logic that assumes values are dates. For example, when you subtract one date from another, the result may be different than if you subtract the integer representations of both from each other.

3. Entities

The second group of the connector are the entity tables. To understand what should be defined in such a table, we first describe what entities are. An **entity** can be seen as an object of interest for which you want to see how it behaves through the process.

Examples

Consider the following examples to get a better understanding of entities.

Example 1

In an invoicing process, invoices are entered in the system, next they are checked, more data may be requested, and finally, invoices are paid.

- In this example, the invoice is the object of interest for which we can identify interesting steps in the process. Therefore, the invoice is an entity.

Example 2

Before the invoicing process starts, a purchase is made. For this part of the process, a purchasing request is entered in the system and the purchase is approved.

- The purchasing document is also an entity in the process.
- The purchasing document relates to the invoice, but it is a separate entity on which steps in the process happen independently of the invoice.

Note: invoices can be of several types, such as employee declarations or customer invoices. The invoice type is a property of the invoice entity, but it is **not** an entity on itself.

Define entity tables

From a technical point of view, each entity needs to be stored in a separate table, where each record in a table describes exactly one instance of that entity. To recognize each of the instances individually, there should be a unique identifier available. This **Entity ID** is later used to define for which entity an event happens.

In the first step of the connector, the input tables are defined. These input tables are the base of the entity tables. Dependent on the source system either one input table or multiple input tables combined defines the entity.

Tip: make sure that in the entity tables the Entity ID is always unique. Joining input tables incorrectly may result in duplication. This will give incorrect results in later steps of the transformation, such as duplicate cases and events.

Entity properties

An entity has several properties that describe it and are specific to that entity. An invoice has other properties than, for example, a purchasing document. The resulting entity table should contain all the properties that are of interest for analysis. For example, these are the invoice ID (Entity ID), the user who entered the invoice in the system, and the creation date.

Example entity table for an invoice:

Invoice ID	User ID	Creation date
1001	User0123	Jan 15, 2021
1002	User0214	Feb 08, 2021

Extra information about entity properties can be stored in different tables. For example, information related to the user is stored in a separate user table. If we want to have the name of the user instead of the id, we should take this information from the user table. Such a user table is an example of **master data**.

Example user table that stores additional information on users:

User ID	User name	Department

User0123	Macy Hurst	Finance
User0214	Tarik Cohen	Administration

The user information can be made available on the entity table by joining the user table with a **left join** to the entity table. A left join makes sure that the entity table is enriched with the master data when available, but does not removes records from the entity table itself. Make sure that after this join, the entity ID remains unique on the entity table and no data duplication is caused.

Tip: create separate tables for master data based on the input tables. This helps to understand and maintain the connector more easily. It allows you to abstract from the original table name in the source system to a more generic name, while keeping the original name for traceability as well.

Multiple entities

The number of entities that are of interest in a process can vary depending on the complexity of the process.

Simple processes

In a relatively simple process, only one entity may be involved. For example, in an invoicing process in which you are only interested in events that take place related to the invoice.

Complex processes

In more complex processes, multiple entities can be of interest. An example is a Purchase-to-Pay process in which the process from requesting a product up to paying the invoice covers events related to multiple entities. In a process with multiple entities, you need to make sure that the relation between these entities is present on the entity tables. If an invoice is created for a certain purchasing document, the purchasing document identifier should be a property on the invoice entity table.

4. Events

In this transformation step, we define the events for the entities. Each record in an event table represents one event that took place for the entity. In most situations, more data records are created. This is the result of the joins and unions to link the events to each entity they belong to.

Prior to building the event tables, you should understand the main activities in the process. Prioritize adding activities based on how often they will likely occur and how meaningful they are to the end user. Defining the activities should be an iterative process. Start with the high value activities, which are the main activities in the process that describe the core of the process. With only the main activities you can already start validating the process.

Ideal number of activities

An ideal number of activities is between 10 and 20. With a small number of activities there will not be a lot of possible variation in the process. More activities increase the number of process variants. With more process variants, you have to reduce the complexity of the process graphs to understand the flow. This also makes it harder to come to general conclusions.

Number of activities	Result
<10	Low analysis complexity, low number of potential improvements.
10-20	Optimal balance of analysis complexity and potential improvements.
>20	High analysis complexity, high number of small improvements.

Define event tables

In event tables, each record should represent exactly one event. On high level there are two scenarios on how the data is structured, which each requires different transformations:

- One or more tables with multiple timestamp columns.
- One or more tables with one timestamp column (transaction log).

Multiple timestamp columns

In this scenario, multiple timestamp properties could be defined on the entity table. Consider the example in which an invoice entity is described with three timestamp columns. Each column stores the timestamps of an activity: one for the creation of the invoice, one for the deletion, and one for a price change.

Invoice ID	Creation time	Deletion time	Price changed time
1001	Jan 15, 2021	Jan 17, 2021	NULL
1002	Feb 01, 2021	NULL	Feb 02, 2021

For each of the columns, we create a new table that represents the events of that activity. In each of the event tables, we need an activity definition to indicate which event is performed, a timestamp that indicates when this happened, and the invoice ID to know for which invoice the event is performed. This results in the following three tables:

Invoice ID	Activity	Event end
1001	Create invoice	Jan 15, 2021
1002	Create invoice	Feb 01, 2021

Invoice ID	Activity	Event end
1001	Delete invoice	Jan 17, 2021

Invoice ID	Activity	Event end
1002	Change invoice price	Feb 02, 2021

The resulting event table for the invoice entity can be created by a **union** of the separate event tables. Make sure to only have records in each table for which an activity name is available and a timestamp is known.

Transaction log

This scenario applies when the input data contains a transaction or audit log containing a record for each event that is happening. A transaction log may contain events corresponding to several entities. Consider the example that the transaction log stores events related to purchases and to invoices.

Entity	ID	Transaction description	Transaction time
Purchase	P12	Creation	Jan 3, 2021
Purchase	P12	Approval	Jan 10, 2021
Invoice	1001	Creation	Jan 15, 2021
Invoice	1001	Deletion	Jan 17, 2021
Invoice	1002	Creation	Feb 01, 2021
Invoice	1002	Price changed	Feb 02, 2021

The solution here is to create an event table per entity. This increases the number of tables in the connector, but it will keep the structure clear. In the example, we split the log in purchase events and invoice events and rename the columns. This results in the following two tables:

Invoice ID	Activity	Event end
1001	Create invoice	Jan 15, 2021
1001	Delete invoice	Jan 17, 2021
1002	Create invoice	Feb 01, 2021
1002	Change invoice price	Feb 02, 2021

Purchase ID	Activity	Event end
1001	Create purchase	Jan 3, 2021

1001	Approve purchase	Jan 10, 2021

Tip: event tables are usually the largest table in the connector. Ensure you filter out any unwanted records in the joins. A wrong join condition can easily blow up the data volume you have. Look at the number of records in the event table. Is the number higher or lower than expected, then review the join conditions.

Event attributes

Mandatory attributes on the event tables are the **Entity ID**, **Activity** and **Event end**. These attributes describe for which entity an event occurs, what the event is, and when the event was performed.

Entity ID

A unique key is required to identify the entity for which a certain event happens. In the scenario of multiple entities in your process, you will also have multiple event tables. The entity ID on each of the event tables is important if you want to create the end-to-end event log. The creation of the event log will be discussed in Chapter 5: Event log.

Activity

The activity describes which event took place. If the name of the activity is not derived from the values in the data, you need to make an expression to define the name yourself. You can infer the activity name from the column names. For example, you can define the activity 'Create document' based on the column 'document_creation_date'. If the column names are less descriptive, you should consult with the process expert for a meaningful name that resonates with the end users.

When naming the activity yourself, best practice is to use the 'Verb Noun' format, such as 'Create document'. Most activities are transactional and this naming convention conveys that. See the table below for some other advice on activity naming.

Not ideal	**Why not ideal**	**Best practice**
Order order	Ambiguous activity name	Order material
Ticket	Singular activity name, which does not state what happened to what.	Create ticket
Document cancelled	Passive tense	Cancel document
Approve credit control check on the sales order	Too long activity name, which will be hard to read in visualizations.	Approve SO credit check

Event end (timestamp)

The event end attribute indicates when the specific event was performed. For analysis, we take the event end as the timestamp of the event, because usually in the data the time is available of when the event is finished. The event end should be as detailed as possible. A date is generally not sufficient since multiple events can happen on the same date.

Optional attributes

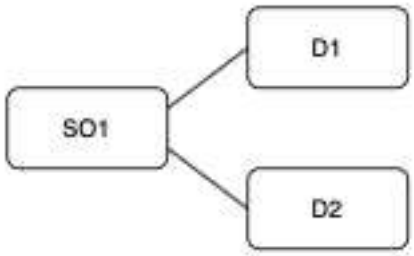
In the event tables you can add additional properties of events. For example, the event detail or supporting properties to define an open or closed case. We advise to only add the additional attributes after the event logic is validated. Optional attributes may be required for specific visualizations. The

attributes 'Event cost' and 'Event processing time' (also referred to as 'work estimates') are, for example, required for analyzing automation opportunities.

5. Event log

The event log describes the events of the end-to-end process. Creating an event log is only necessary when the process contains more than one entity. Only one entity in the process can function as the **main entity** which will be tracked through the process. This main entity is named the **case** in the process. Each record in the event log table represents one event for a certain case.

To illustrate the difference between the events tables as described in the previous chapter and the event log table, consider the following example from an Order-to-Cash process. We have two entities: sales order and delivery. One sales order (SO1) is linked to two deliveries (D1 and D2).



For the sales order and the delivery entity we have two separate event tables, which contain the following information:

Sales order ID	Activity	Event end
SO1	Create SO	Jan 15, 2021

Delivery ID	Activity	Event end
D1	Create delivery	Feb 01, 2021
D1	Change date	Feb 03, 2021
D2	Create delivery	Feb 15, 2021

Each of these event tables describe the events per entity. For the event log table, we want to describe the events for the main entity while taking into accounting the related entities. If we consider the delivery entity as our main entity, the event log table contains the following information:

Case ID	Activity	Event end
D1	Create SO	Jan 15, 2021
D1	Create delivery	Feb 01, 2021
D1	Change date	Feb 03, 2021
D2	Create SO	Jan 15, 2021
D2	Create delivery	Feb 15, 2021

We could not simply union the separate events tables together, because we would not have the correct Case ID for all events. Note the 'Create SO' event for SO1. This event happens only once in the process, but since delivery is our main entity, the event occurs twice in the event log.

Define the entity relation table

To create the end-to-end event log based on the separate event tables, we need to know how the entities in the process relate to each other. This information can be stored in the **entity relation** table. The entity relation table helps to centralize the relationships between all entities.

The information to link entities together should be available on the entity tables. Continuing the example of sales order and deliveries, that means that the delivery should have the property to which sales order it relates to. The entity relation table holds all the relationships between instances. This can be structured in the following way:

Sales order ID	Delivery ID
SO1	D1
SO1	D2

Each column represents one entity in the process. Each row contains a unique combination of entity IDs that indicate they are related. Because multiple deliveries can be related to one sales order, the sales order SO1 occurs in multiple records of this table.

The entity relation table can be created by joining the entity tables:

- A **full join** is used for the main entities that can function as the case. The full join ensures that the instances of the main entity are included in the entity relations, also when the instance has no link to others.
- A **left join** is used for the other entities that will not be cases.

Having more than one main entity in the entity relation table is only needed when the connector is created that allows for multiple perspectives. For example, a Purchase-to-Pay process where either the purchase order can be set as the case or alternatively the invoice. Only one entity can be the main entity at the same time in an event log.

Example

An entity relation table is created to link the following four entities: sales order, delivery, delivery cancellation, and the invoice. The sales order indicates the start of the process and has no reference to another entity. The three tables below show the other three entity tables that contain a reference to another entity. Other properties of these entities are not considered in this example.

Delivery entity table:

Delivery ID	Sales order ID
D1	SO1
D2	SO1

Delivery cancellation entity table:

Delivery cancellation ID	Delivery ID
C1	D1

Invoice entity table:

Invoice ID	Delivery ID
I1	D1
I2	D1
I3	D2
I4	NULL

In words, the following information is stored in these table:

- Both deliveries D1 and D2 are linked to sales order SO1.
- Delivery cancellation C1 cancels delivery D1.
- Invoices I1 and I2 are created for delivery D1, invoice I3 is created for delivery D2, and invoice I4 has no link to a delivery.

We want to have the flexibility to set either the sales order, delivery, or invoice entity as a case in the process. The delivery cancellation does not have to be a possible case in this process.

For this process, we join the entities in the following way:

```
Sales_order
full outer join Delivery
    on Sales_order."Sales_order_ID" = Delivery."Sales_order_ID"
left join Delivery_cancellation
    on Delivery."Delivery_ID" = Delivery_cancellation."Delivery_ID"
full outer join Invoice
    on Delivery."Delivery_ID" = Invoice."Delivery_ID"
```

The use of the different join types is shown. A left join is used for the delivery cancellation, since in this process, this entity cannot be a case ID. All other entities are joined with a full join. The table below shows the result of joining the entities.

Sales order ID	Delivery ID	Delivery cancellation ID	Invoice ID
SO1	D1	C1	I1
SO1	D1	C1	I2
SO1	D2	NULL	I3
NULL	NULL	NULL	I4

With the information in this table we know for every instance how it relates to other instances. Consider that we want to have the delivery as our case. We know that:

- For delivery D1, sales order SO1, delivery cancellation C1 and invoices I1 and I2 are related.
- For delivery D2, sales order SO1 and invoice I3 are related.

Define the event log table

The entity relation table is used to create the event log table. For this, we first define which entity is the main entity. Together with a process expert, you need to define what this main entity in the process is. The identifier of that entity will function as the case ID of your process.

Consider the example that we choose the delivery as the case. We see in two records of the entity relation table the same information with respect to the sales order relation: the first two records indicate that sales order SO1 is linked to case D1. If we are going to generate events based on this information, we do not want to have twice all the events for SO1. We want to create an event log table in such a way that events of each instance are created once per case ID.

When we generate the events for case D1, we generate events once for SO1, D1, C1, I1 and I2. For case D2, we generate events once for SO1, D2, and I3. Note that for SO1 events are generated twice, but only once per case ID. Invoice I4 is not linked to any delivery and therefore events will not be generated.

Case ID	Sales order ID	Delivery ID	Delivery cancellation ID	Invoice ID
D1	SO1	D1	C1	I1
D1	SO1	D1	C1	I2
D2	SO1	D2	NULL	I3
NULL	NULL	NULL	NULL	I4

Example

To illustrate the creation of the event log table, consider the example that sales order SO1 is linked to two deliveries D1 and D2. We consider the delivery as our main entity in the process. The entity relation table looks in that scenario as follows:

Sales order ID	Delivery ID
SO1	D1
SO1	D2

We have the following event tables for the two entities:

Sales order ID	Activity	Event end
SO1	Create SO	Jan 15, 2021

Delivery ID	Activity	Event end
D1	Create delivery	Feb 01, 2021

D1	Change date	Feb 03, 2021
D2	Create delivery	Feb 15, 2021

Each event table is joined with the entity relation table on the entity ID. We only need to make sure that we join on the entity relation table where once the entity ID is present per case. These joins are then unioned together to create the complete event log. In SQL statements, this would look as follows:

```
select
    Entity_relations_delivery."Delivery_ID" as "Case_ID",
    Delivery_events."Activity",
    Delivery_events."Event_end"
from Delivery_events inner join (select distinct "Delivery_ID" from Entity_relations) as Entity_relations_delivery
    on Delivery_events."Delivery_ID" = Entity_relations_delivery."Delivery_ID"
union all
select
    Entity_relations_sales_order."Delivery_ID" as "Case_ID",
    Sales_order_events."Activity",
    Sales_order_events."Event_end"
from Sales_order_events inner join (select distinct "Delivery_ID", "Sales_order_ID" from Entity_relations) as Entity_relations_sales_order
    on Sales_order_events."Sales_order_ID" = Entity_relations_delivery."Delivery_ID"
```

Since the delivery is the main entity, we rename the delivery ID column to case ID. Also the other two mandatory attributes activity and event end are defined. With the use of the 'select distinct' construction in the join we make sure that we do not cause event duplication in the event log.

The result of this union will be the event log table that looks as follows:

Case ID	**Activity**	**Event end**
D1	Create SO	Jan 15, 2021
D1	Create delivery	Feb 01, 2021
D1	Change date	Feb 03, 2021
D2	Create SO	Jan 15, 2021
D2	Create delivery	Feb 15, 2021

6. Business data

We have two additional concepts in UiPath Process Mining that enrich the data with business logic to support the analysis of a process:

- **Tags**: properties of cases, which indicate whether a case complies to a certain business rule. These are, for example, process KPI's which are present in the business. Tags can be considered as unwanted behavior in the process.
- **Due dates**: properties of events, which indicate whether an event is performed on time.

Define the tags table

Each record in the tags table represents one tag that occurred in the data for a specific case. The mandatory attributes on this table are the **Case ID** and the **Tag**. These are mandatory, because we need to know which tag occurs for which case.

To illustrate which case properties can be considered a tag, see the following examples:

- An invoice is approved and paid by the same person.
- The time until approval is more than 10 days.

- The activity to check the invoice is skipped in the process.

Not all cases will have a tag and some cases may even have multiple tags. For example, we have two cases: one case has two tags and the other case has one tag. The resulting tags table will have three records as in the table below:

Case ID	Tag
1001	Approved and paid by the same person
1001	Approval time is more than 10 days
1002	Approval time is more than 10 days

Tip: many tags will need the information about which events are executed for a case. Therefore, make sure that the event log is already created before the tags logic is defined.

Define the due dates table

Each record in the due dates table represents one due date for a certain event. The mandatory attributes on this table are the **Event ID**, **Due date**, **Actual date**, and **Expected date**. With these mandatory attributes, we know to which event a certain due date belongs and what the actual and expected date are. This also implies that when due dates are part of the data model, the Event ID is also a mandatory attribute on the events table.

Example due dates are a payment deadline for a payment event or an approval deadline for an approval event. Similar as for tags, not all events shall have a due date and some events may have multiple due dates. For example, we have two events: one event has two due dates and the other event has one due date. The resulting due dates table will have three records as in the table below:

Event ID	Due date	Actual date	Expected date
1001-5	Payment deadline	Feb 15, 2021	Feb 16, 2021
1001-5	Payment deadline with discount	Feb 15, 2021	Feb 10, 2021
1002-2	Approval deadline	Mar 03, 2021	Mar 02, 2021

When we compare the actual and expected dates of these due dates, we see that for event 1001-5 the due date ‘payment deadline’ is made by 1 day. For the same event, the ‘payment deadline with discount’ is late by 5 days.

7. Output

The output tables of the connector represent the tables as given in the data model. At this transformation step, the main transformations are already implemented in the Entity and Events tables and optionally also in the Event log, Tags, and Due dates tables. When applicable, the output tables will be used to bring all relevant information together such that the connector output adheres to the defined data model.

Mandatory and optional attributes

In the data model, some attributes are mandatory and some are optional. For example, the case ID, activity, and event end are mandatory in an event log output table. That means that the values of these

attributes all need to contain values. If one of the records in the tables does not have a value for either of the mandatory attributes, a proper analysis cannot be performed.

Optional attributes need to be part of the output tables. It is, however, not required that these attributes contain values.

Next steps

Developing a connector can be an iterative process of continuously analyze the output of the dataset, getting feedback from users, and changing the connector accordingly.