TRACELINK UNIVERSITY

Home
Resources
TraceLink University

Custom Transforms Development Guide

Overview

This guide covers the steps necessary to create a custom transform:

1. Familiarize yourself with relevant terminology
2. Plan your transform by mapping schema fields
3. Validate your planned field mappings
4. Select which method of transform creation to use
5. Create your transform
6. Upload and use your transform

Transform Terminology

Term	Definition
Canonical	TraceLink's system agnostic data format. The canonical data model allows information to be exchanged with any partner on the network, regardless of their data format.
Standard Transform	Data mapping of standard EDI formats with TraceLink canonicals
Custom Transform	Data mapping of external systems with TraceLink canonicals
Transaction Type	B2B messages relating to a business process (e.g. purchase orders, invoices, etc.) All supported transaction types can be found **here**.

Inbound Transform	Data is transformed from an external system to TraceLink
Outbound Transform	Data is transformed from TraceLink to an external system

Plan Your Transform

At a high level, a transform defines data mappings between two schemas. The goal of this step is to manually plan how fields from an external system schema map to fields of a transaction's TraceLink canonical schema.

The table below reviews the materials needed to plan your transform:

Requirement	Description	Link
External system schema file	A file providing the schema from the external system. This file should provide the structure of fields in the external system for the designated transaction type.	N/A
External system base file	This file should allow you to review example data for every field in the schema.	N/A
TraceLink's canonical schema file	Provided by TraceLink, this is a JSON file providing the schema of a TraceLink canonical. Each transaction type has its own canonical. This field should allow you to understand each field in the TraceLink canonical.	**Link**
TraceLink's canonical reference documentation	Provided by TraceLink, this documentation provides usage details on each field in TraceLink's canonical schema. For example, which fields are required.	**Link**

Leveraging the external system's schema and TraceLink's canonical reference documentation for reference, map fields between each in the schema files.

We recommend creating a spreadsheet file, similar to the table below, to do this. Creating a field mapping spreadsheet upfront makes the transform creation step easier.

External System Field	TraceLink Canonical Field
addr1	address1
createdDate	transactionDate
companyName	supplierName
amount	pricingAmount

Please note that you may need to create multiple transforms. Each transform supports one transaction type and one direction. For example, if you need to transform data for purchase orders inbound and outbound, you will create two separate transforms.

Once you have completed your mapping plan you can proceed to validation.

Validate the Transform Mapping

This step provides you with test cases to manually validate against your map plan.

- Evaluate that every required field has a value
- Evaluate that required fields will error correctly if there is no provided value
- Evaluate that non-required fields will not error if no value is provided
- Evaluate that if an invalid value is given it will error
- Evaluate that field having a required number of characters are the correct length (e.g. serial numbers)

Methods of Custom Transform Creation

There are two methods currently available for building transforms: OpenText Contivo and JavaScript. The table below provides an overview of each option.

	OpenText Contivo Transforms	**JavaScript Transforms**
Output	.jar file	.js file
Benefits	• Centralized visual tool • Low code	• Highly customizable • Open source code
Prerequisites	• OpenText Contivo version 21 license • Training • EDI expertise	• Most recent version of JavaScript ES6 • Development experience

Transform Extensions

Chained Transforms

Chained transforms are useful when you need to perform multiple, distinct data transformations in a specific sequence, ensuring that each step builds on the output of the previous one. This allows you to streamline complex processes that involve various data formats or transformations.

Chained transforms use a list of child transforms and are executed in a defined order. The initial input file is input into the first child and its output is used as the input for the next child and so on until the end of the chain is reached.

For example:

- Transform A converts CSV to XML
- Transform B converts XML to JSON
- Transform C converts JSON to EDI
- Transform D chains A, B, and C together to convert CSV to EDI

When creating chained transforms, you have the flexibility to combine those built with OpenText Contivo and JavaScript, as well as mix custom and standard transforms. However, chained transforms cannot be chained together.

If you are creating a chained transform, please create each transform in the chain and then proceed with the upload and use step, where the execution order is defined. For the example above you would send us details such as:

Please execute these transforms in the order below

- Transform A
- Transform B
- Transform C
- Transform D

At this point you should have your validated mapping specification file and know which method you will use to create your transform.

Create a Transform

In this step you will create the transform itself using your chosen method of creation. At the end of this step your transform will be a .jar or .js file.

Creating Transforms with OpenText Contivo

The table below reviews the required materials for creating a transform using OpenText Contivo:

Requirement	Description	Link
External system schema file	A JSON file providing the schema from the external system. This file should provide the structure of fields in the external system for the designated transaction type.	N/A
External system base file	This file should allow you to review example output for every field in the schema.	N/A
TraceLink's canonical schema file	Provided by TraceLink, this is a JSON file providing the schema of a TraceLink canonical. Each transaction type has its own canonical. This file should allow you to understand each field in the TraceLink canonical.	**Link**
TraceLink's canonical reference documentation	Provided by TraceLink, this documentation provides usage details on each field in TraceLink's canonical schema. For example, which fields are required.	**Link**

Import the external system schema file and TraceLink's canonical schema file to create your mappings in OpenText Contivo, using your mapping specification file for reference.

When complete, compile into a .jar file. This is what you will need to provide TraceLink in the upload and use step.

Creating Transforms with JavaScript

This step walks you through how to create a transform using Javascript, with the output being a .js file.

The code sample below shows the overall structure of a transform created with JavaScript.

```
let response = {};
    try {
    // get input file contents
    // transform content
    // put output file
      response.result = "SUCCESS";
    } catch (e) {
      context.log().error(`Transform Javascript Activity: Error
encountered while executing Javascript transform: ${e}`);
      response.result = "FAILURE";
      response.resultErrorText = e.message;
    }
    return response;
```

The following attributes are available for use:

`context.log` allows you to log messages

```
context.log().info('Transform Javascript Activity: Executing
Javascript transform');
```

`context.s3.getInputFileString` returns a `promise<string>` to get the contents of the input file to be transformed

```
const inputFileString = await context.s3.getInputFileString();
```

`context.s3.uploadOputFileString(data);` returns a `promise<S3PutOpjectResponse>` to upload the transformed data to an object in S3

```
const uploadFileResponse = await
context.s3.uploadOutputFileString(data);
```

`context.sendEvent(event, appName, data)` sends event handler requests and returns a `Promise<object>` for the event's response. This can be used with any endpoints in our API reference documentation. For example, if your external system can provide a product's ID, but not the name, you can make a call to product master data to retrieve and insert it.

```
const data = {
```

```
      transformName: 'TL_JSONtoCSV_1_0_OB',
      transformVersion: 1
    };
    const eventResponse = await context.sendEvent('masterdata-
manager:get-product-by-itemcode:v1', 'masterdata-manager', data);
```

All JavaScript transforms should return a response object that is used to communicate success or failure of the transform execution. The response object should contain a result string attribute with either success or failure. If the result is `FAILURE`, the response should also have a `resultErrorText` string attribute to provide details about the cause.

```
let response = {};
    try {
    // get input file contents
    // transform content
    // put output file
      response.result = "SUCCESS";
    } catch (e) {
      context.log().error(`Transform Javascript Activity: Error
encountered while executing Javascript transform: ${e}`);
      response.result = "FAILURE";
      response.resultErrorText = e.message;
    }
    return response;
```

Sample JavaScript Transform

```
//Log the start of the transform activity
    context.log().info('Transform Javascript Activity: Executing
Javascript transform');
    //Function to convert JSON object to XML format
    function JSONtoXML(obj) {
      let xml = '';
      for (let prop in obj) {
        xml += obj[prop] instanceof Array ? '' : '<' + prop + '>';
        if (obj[prop] instanceof Array) {
          for (let array in obj[prop]) {
            xml += '\n<' + prop + '>\n';
            xml += JSONtoXML(new Object(obj[prop][array]));
            xml += '</' + prop + '>';
          }
        } else if (typeof obj[prop] == 'object') {
          xml += JSONtoXML(new Object(obj[prop]));
```

```
        } else {
          xml += obj[prop];
        }
        xml += obj[prop] instanceof Array ? '' : '</' + prop + '>\n';
      }
      xml = xml.replace(/<\/?[0-9]{1,}>/g, '');
      return xml;
    }
    let response = {};
    try {
      //Get the input file as a string
      const inputFileString = await context.s3.getInputFileString();
      //Parse the input file string into a JSON object
      const inputFileObject = JSON.parse(inputFileString);
      //Convert the inputFileObject to XML
      const xml = JSONtoXML(inputFileObject);
      //Upload the resulting XML string as the output file
      const uploadFileResponse = await
context.s3.uploadOutputFileString(xml);
      //Log the response from the upload output file indicating a
SUCCESS or a FAILURE with an error message
      context.log().info(`Transform Javascript Activity: Response
returned from upload output file api is:
${JSON.stringify(uploadFileResponse)}`);
      response.result = "SUCCESS";
    } catch (e) {
      context.log().error(`Transform Javascript Activity: Error
encountered while executing Javascript transform: ${e}`);
      response.result = "FAILURE";
      response.resultErrorText = e.message;
    }
    return response;
```

Your transform should now be a .js file, ready for upload to TraceLink.

Upload and Use a Transform

Now that you have a .jar or .js containing your transform and are ready to upload it for use, please send an email with the information below below to partner-operations [at] tracelink.com (partner-operations[at]tracelink[dot]com).

Information	Required?	Details

Environment	Required	Specify which environment you would like your transform uploaded to: validation and/or production. Our best practice recommendation is to use your transform in validation before production. You may request deployment to both validation and production to avoid submitting another request, requiring additional turnaround time, for us to enable an additional environment.
Chained Transform Sequence	Use Case Dependent	If you are creating a chained transform, indicate the necessary sequence of execution.

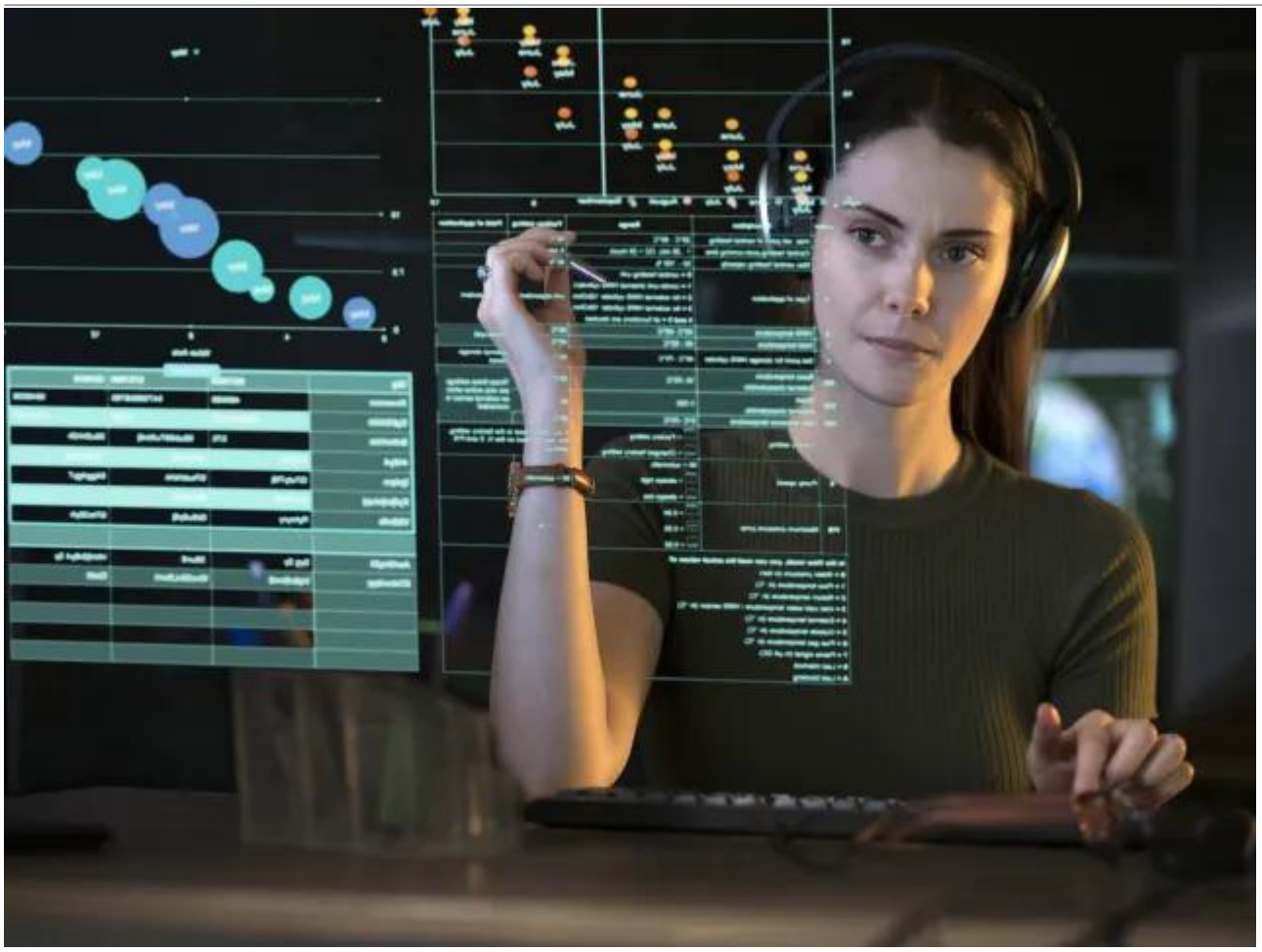
You should receive an immediate, automated response if we successfully receive your request. A member of our team will follow up with you validating the required details and requesting your transform file.

Once we receive the file and upload it to our system, we will confirm when your transform is ready for use. At that time, we recommend testing your transform in your validation environment before proceeding to production. Please let us know if you encounter any issues that require our assistance.

Change Management

Your custom transform cannot automatically account for any new or modified fields. Therefore, you will need to go through the steps above to generate a new .js or .jar file containing the changes. The uploaded changes will overwrite previous files unless otherwise specified. If you have any questions about this process please email partner-operations [at] tracelink.com (partner-operations[at]tracelink[dot]com).

Related Content



Canonical Reference

Canonical objects for transform development

View More

Use Case: Custom Transforms

Data mapping with TraceLink
View More