Nodes and Fields

Introduction

Nodes and fields are contained within the schemas that are specified either during activity configuration or in the transformation. While configuring a transformation, nodes and fields are displayed the same regardless of the transformation display mode.

Data Structures

Data structures are displayed in a tree format that can be expanded and collapsed to show either the entire tree or just a portion of it. (For an explanation of types of data structures, see Data Structure Types under Instance and Multiple Mapping.)

Data structures are commonly provided as file schemas during activity configuration. These file schemas are inherited by the transformation using the activity as a source or target in the operation. A schema can also be defined within the transformation itself.

Each tree consists of nodes and fields, where fields within the source data structure can be mapped to fields within the target data structure.

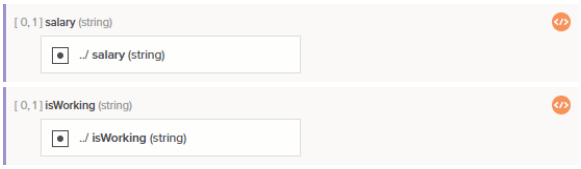
Nodes have a disclosure triangle to the left of the node name that is used to collapse or expand the node. Once expanded, nodes display any contained child nodes and fields. Nodes can be considered as folders with child nodes as sub-folders. Fields are contained within nodes and are listed with their data type (`boolean`, `integer`, `double`, `binary`, `string`).

For example, in the target structure shown below, the node `json` includes the child node `item`, which contains the fields `employeeId`, `name`, and `title`. The node `item` also contains the child node `employeeDetails`, which contains the fields `salary`, `isWorking`, and `status`.

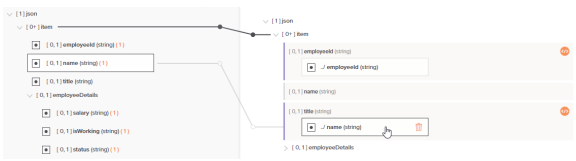
Mapped Fields

A transformation mapping consists of target fields or nodes and their corresponding scripts. These scripts may contain references to source fields or nodes or to project components, use functions, or contain other valid script logic. A mapping does *not* include target fields that are not mapped.

When source objects and variables are defined within the target field, they appear as blocks within the target field. The mapped target field is displayed with a purple vertical line along the left of the target field block:



When both a source and a target schema are visible on the screen and you are in mapping mode, a visual line shows the connection with the source object. Hover over a mapped target field to show a light gray line that connects from the source object(s) used in the mapping to the mapped target field:



If a collapsed node contains target field mappings, that node is shown in bold to indicate it contains mapping.

On This Page

Related Articles

- Instance and Multiple Mapping
- Instance Functions

Related Topics

- Cloud Studio
- Schemas
- Schemas Defined in a Transformation
- Schemas Defined in an Activity
- Transformation Basics
- Transformation Display Modes
- Transformations

Last updated: Aug 30, 2019

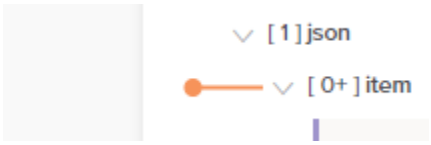
Loop Nodes

A *loop node* is a source or target node with repeating data values, such as line items in an invoice or a set of customer records.

When loop node fields are mapped, a solid black *iterator* line will automatically appear, indicating that the transformation process will loop through the source data set. A transformation can have zero or more iterator lines.

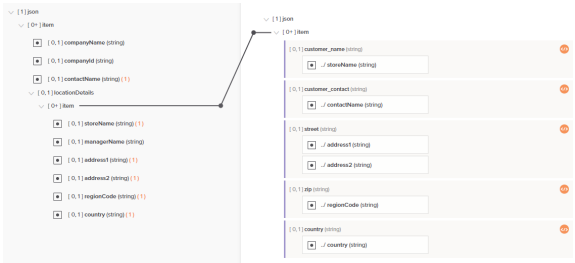
To toggle the display of an individual iterator line, click directly on the circle shape that is closest to the target node:

The individual loop node line will then become an orange stub that when clicked again displays the full line:



As an example of a loop node mapping, consider the following hierarchical source structure containing a top-level node with fields that provide information about a company. A child node, `locationDetails.item`, includes an array of objects with fields for multiple store locations within a company. Both the parent and child node are considered loop nodes because the data may contain multiple company records with multiple store location records for each company.

Now consider that this data is being mapped to a flat target structure, resulting in a record for each store location. As you map fields, an iterator line will automatically appear connecting source and target loop nodes. This line indicates that the target will loop as many times as there are repeating sets of data in the source, or in this example will loop through each store location record for each company.



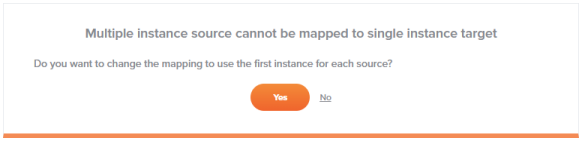
Mapping from Multiple Source Loop Nodes

Generating the iterator line for the target loop node depends on the multiplicity of the corresponding source loop node. Sometimes, when the generated target loop node depends on more than one source loop node, you may need to resolve a multiple occurrence conflict with the mapping. This will be shown as an invalid mapping (see Transformation Mapping Validity).

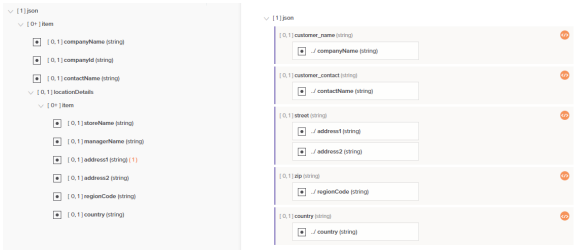
To resolve this error, first check that the schemas being used for the mapping are defined correctly. If all the nodes have a cardinality key of `[0+]`, indicating zero or more elements are valid, this indicates a possible mislabeling of the multiplicities. If you have confirmed the schemas are correctly defined and the conflict still exists, you can resolve the conflict by using the instance-resolving functions (see Instance Functions).

Mapping from a Multi-Instance Source to Single-Instance Target

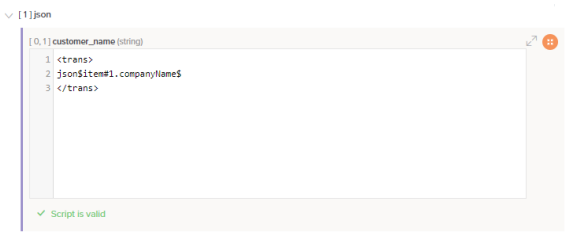
If the source data structure is a multi-object array and is being mapped to a target data structure with a single object, this popup message will be displayed:



To use the first instance of the source in the mapping, select **Yes**. This means that only the first record will be mapped. For example, given the following mapping, only the first customer record in the array is mapped to the target structure containing only a single customer. Notice that each mapped target field now contains a script as indicated with the script icon .



When you toggle to script mode for any mapped field, you will see that a `#1` has been added within the path of the mapped source object to indicate that the first instance is mapped.



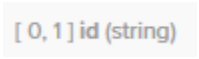
If you do not want the first instance of the source to be used, you can specify other logic using the instance-resolving functions (see Instance Functions).

Nodes and Fields

Each node or field is displayed with these components:

- Cardinality key for nodes and fields
- Name for nodes and fields
- Data type for fields only
- Attribute and value indicators (if applicable)

For example, the field below has a cardinality key of `[0, 1]`, a name of `id`, and a data type of `string`:



Cardinality Key

Cardinality keys notate the relationship for nodes and (if the key is present) fields. The notation is derived from the concept of multiplicity as used in Unified Modeling Language (UML).

Cardinality Key	Usage	Definition
`[1]`	Nodes only	One and only one element is valid.
`[1+]`	Nodes only	One or more than one element is valid.
`[0,1]`	Nodes or fields	Zero or one element, value, or attribute is valid. This key means that a given node or field is effectively optional.
`[0+]`	Nodes only	Zero or more elements is valid.

Name

Each node or field has a name that comes directly from the data schema.

Data Type

Each field has an associated data type, such as `integer`, `float`, `double`, `long`, `string`, `boolean`, or `binary`.

Attribute and Value Indicators

Some types of structures, such as XML or SOAP, may have additional symbols that represent whether a field is a value or an attribute. Elements, represented by nodes, do not have any visual indicators. Elements may have one or more other elements, attributes, values, a combination of these, or none depending on the cardinality key.

Visual Indicator	Definition
@	Attribute as represented by a field. This field is used to indicate a node has an attribute, and may or may not also have a value. There may also be more than one attribute for each node.
#	Value as represented by a field. The field is used to indicate a node has a value and zero or more attributes.

This example shows a node (`color_swatch`) that has both a value (`text`) and an attribute (`image`):