Transforms by Example - the Simpler Way to Integrate Healthcare Applications

Transforms By Example (TBX) *is a new way to integrate healthcare applications, which reduces costs, while improving quality and maintainability. TBX is easy to learn and easy to use. If you have been building interfaces and transforms in other ways, you can reverse engineer them to TBX form, and switch to TBX without losing your investment. Run a trial now to confirm that TBX is the best way to integrate your applications.*

Healthcare application integration now centres on the use of HL7 FHIR as the interchange standard. This requires transforming healthcare data accurately between native data formats and FHIR. The transforms are costly to develop and maintain.

Presently, data transforms are built in two ways:

- By writing transform code in a language such as Java or C#
- By using a mapping and transformation toolset - to generate the transform automatically from mappings.

Neither approach is satisfactory. Code is expensive to develop, and key design decisions are buried in code - so it is hard to reuse or maintain. Mapping tools require learning some new mapping language, and usually have limited capability.

There is now another method which is simpler and more cost-effective - **Transforms By Example** (**TBX**).

To build a transform with TBX, you focus on your requirement. You make a small number of **Example Pairs**, which illustrate what the transform is required to do. Each example pair is an instance of the source data, together with the target instance (such as a FHIR resource or bundle) which the transform is required to produce. The same data values occur in the source instance and the target instance.

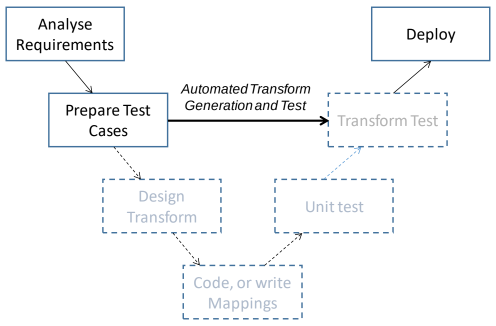
If the example pairs are accurate, the Open Mapping Software **Transform By Example** Toolset does the rest. These Eclipse-based tools:

1. Generate the transform which converts the source to the target (and the inverse transform)
2. Test the transform, giving it the source examples to create the outputs.
3. Compare the outputs with the supplied target examples, to summarise what is missing or incorrect - what the transform does not yet do.
4. Output the transform in deployable forms, such as Java code or FHIR Mapping Language.
5. Display detailed diagnostics in the TBX Example Editor, to help refine the transform

Large numbers of example pairs are not required. An accurate transform can be generated from one good example pair, if that example embodies the requirement. Creating good example pairs requires only an understanding of the requirement.

Most of the work of building a transform - the design, coding, and running test cases - is done automatically by the tools. The parts you have to do - providing the example pairs - are things you must do in any case, to understand the requirement and to test the transform.

TBX simplifies the transform development process. The cost savings are shown on a 'V' lifecycle diagram for software development:



The full 'V' diagram shows how you would develop a transform by other methods. The steps in the dashed boxes are done automatically by the TBX tools. TBX reduces the costs of building transforms, as far as they can be reduced - because understanding requirements and creating test cases are always necessary, and TBX requires no other manual effort.

When source and target data representations are clearly defined, **development cost savings of 50% or more** can be expected, compared with conventional methods. For maintenance of existing transforms, greater savings can be expected. Improvements in quality matter as much as the cost savings. Further:

- TBX tools work in complex cases - where there there are large structure differences between the source and the target, and where data values need to be converted from source to target.
- Sometimes special transform code is required. Because the tools generate transforms as Java code, you can modify or subclass the code to meet special requirements.
- The TBX development process is requirement-driven and test-driven, with a rapid iterative testing cycle. The focus on requirements and testing leads to high quality transforms.
- If you have been developing transforms in other ways, you can reverse engineer these transforms into TBX form - by using their inputs and outputs as the example pairs for TBX. You can move to TBX without losing your investment.

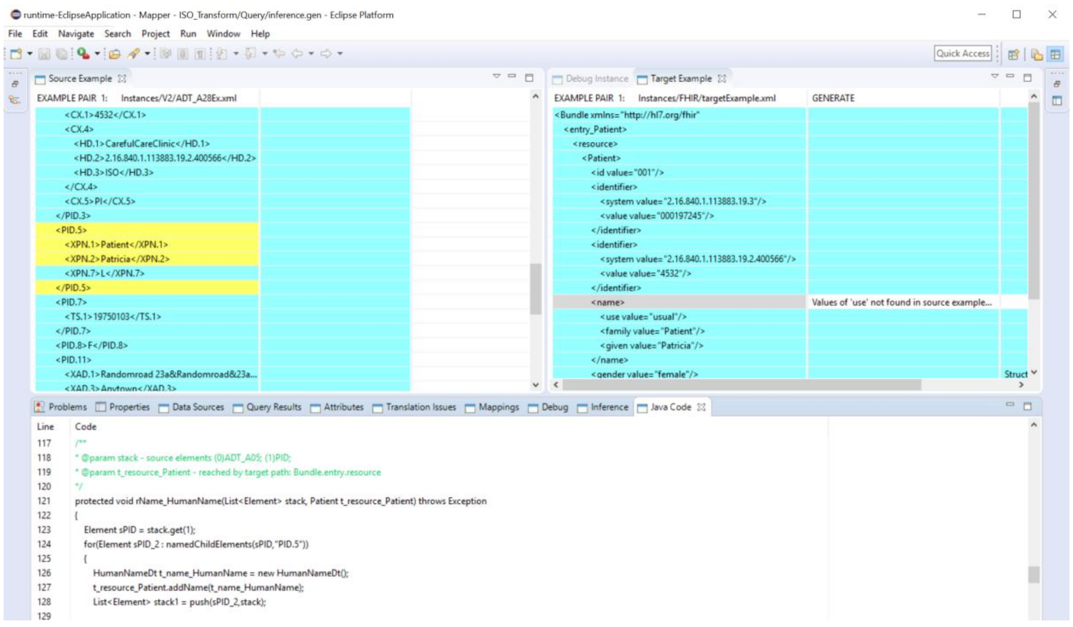
TBX Transforms can be created for a wide range of sources and targets, including:

- Relational databases
- HL7 Version 2
- FHIR Bundles and Resources
- HL7 CDA
- OpenEHR
- Other XML formats

Transforms By Example is the best way to control the costs of healthcare application integration, and to deliver high quality.

The screenshot below shows the TBX Example Editor, showing:

- On the left, an example of the source (HL7 Version 2 XML)
- On the right, the target example (a FHIR Patient resource) that the transform is required to produce from the source example. The patient name is selected.
- Yellow highlights show the parts of the source which will give the selected 'name' part of the target
- Below, generated Java code which creates the patient name part (by calling the HAPI FHIR libraries)



EXAMPLE PAIR 1: Instances/V2/ADT_A28Ex.xml

```
    <CX.1>4532</CX.1>
    <CX.4>
      <HD.1>CarefulCareClinic</HD.1>
      <HD.2>2.16.840.1.113883.19.2.400566</HD.2>
      <HD.3>ISO</HD.3>
    </CX.4>
    <CX.5>PI</CX.5>
  </PID.3>
  <PID.5>
    <XPN.1>Patient</XPN.1>
    <XPN.2>Patricia</XPN.2>
    <XPN.7>L</XPN.7>
  </PID.5>
  <PID.7>
    <TS.1>19750103</TS.1>
  </PID.7>
  <PID.8>F</PID.8>
  <PID.11>
    <XAD.1>Randomroad 23a&Randomroad&23a...
    <XAD.3>Anytown</XAD.3>
```

EXAMPLE PAIR 1: Instances/FHIR/targetExample.xml — GENERATE

```
<Bundle xmlns="http://hl7.org/fhir"
  <entry_Patient>
    <resource>
      <Patient>
        <id value="001"/>
        <identifier>
          <system value="2.16.840.1.113883.19.3"/>
          <value value="000197245"/>
        </identifier>
        <identifier>
          <system value="2.16.840.1.113883.19.2.400566"/>
          <value value="4532"/>
        </identifier>
        <name>                         Values of 'use' not found in source example...
          <use value="usual"/>
          <family value="Patient"/>
          <given value="Patricia"/>
        </name>
        <gender value="female"/>       Struct
```

Line Code

```
/**
* @param stack - source elements (0)ADT_A05; (1)PID;
* @param t_resource_Patient - reached by target path: Bundle.entry.resource
*/
protected void rName_HumanName(List<Element> stack, Patient t_resource_Patient) throws Exception
{
  Element sPID = stack.get(1);
  for(Element sPID_2 : namedChildElements(sPID,"PID.5"))
  {
    HumanNameDt t_name_HumanName = new HumanNameDt();
    t_resource_Patient.addName(t_name_HumanName);
    List<Element> stack1 = push(sPID_2,stack);

```

For more information, see www.OpenMapSW.com or contact rpworden@me.com.

OMS Open Mapping Software
Simply Connected