transformer

External Object Import

Introduction

Java Classes can be imported into Transformer and then viewed like any other definitions.

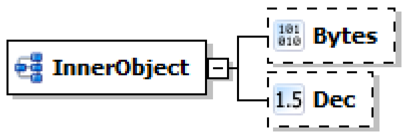
Transformer uses FieldDiscoverer TBeans to work out which fields to generate for a given class. The default FieldDiscoverer (PublicSetAndGetMethods) will generate fields if they have public Set and Get methods and so is designed to work with classes that have a standard Java Bean structure.

This is easiest to illustrate by example. Starting with the following two simple classes:

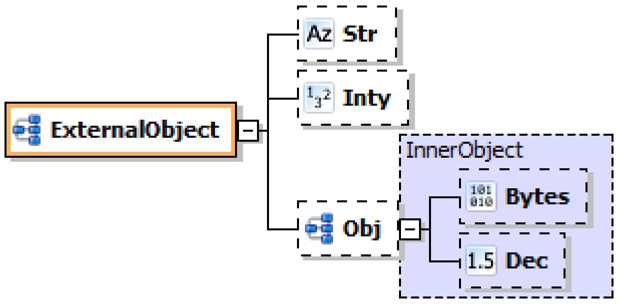
```
package com.acme.objects;

public class ExternalObject {
  String str;
  int inty;
  InnerObject obj;

  public ExternalObject() {

  }

  public String getStr() {
    return str;
  }

  public void setStr(String str) {
    this.str = str;
  }

  public int getInty() {
    return inty;
  }
  public InnerObject getObj() {
    return obj;
  }

  public void setInty(int inty) {
    this.inty = inty;
  }

  public void setObj(InnerObject obj) {
    this.obj = obj;
  }
}

package com.acme.objects;

public class InnerObject {

    Float dec;
    byte[] bytes;

    public InnerObject() {

    }

    public Float getDec() {
      return dec;
    }

    public void setDec(Float dec) {
      this.dec = dec;
    }

    public byte[] getBytes() {
      return bytes;
    }

    public void setBytes(byte[] bytes) {
      this.bytes = bytes;
    }

  }
```

If the user chooses to import ExternalObject then two Transformer ComponentTypes will be created in a Message Definition Group called com.acme.objects. The two Types are InnerObject and External Object and they will appear like so:

InnerObject



ExternalObject

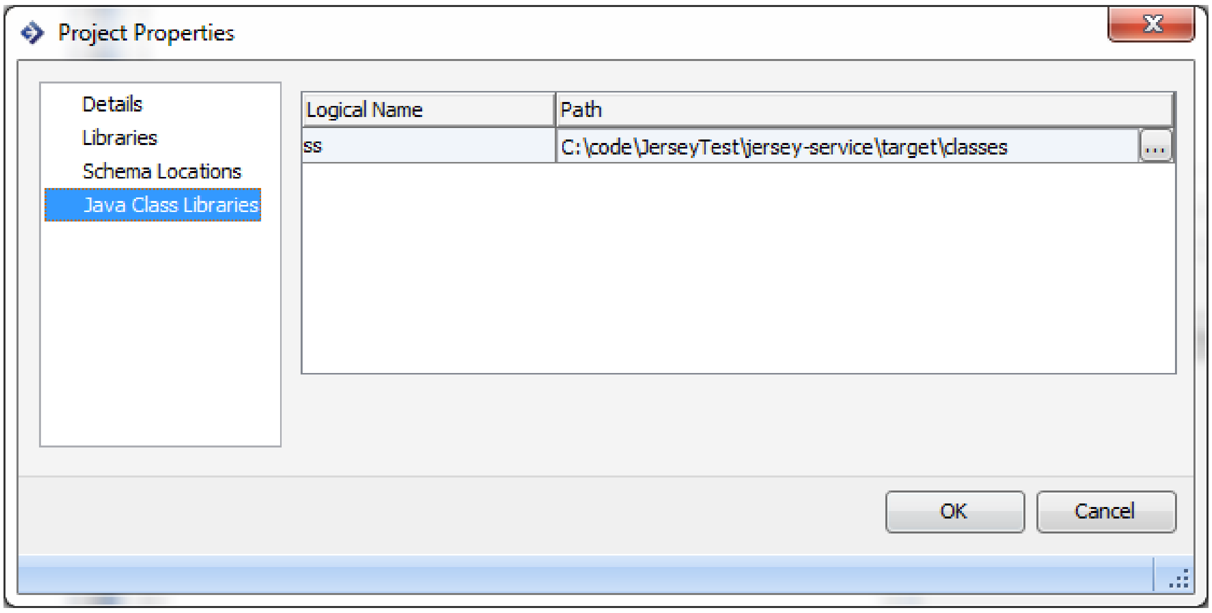


A few things should be noted from this simple import procedure:

- The types of each of the fields have been determined. Dec is a decimal, Str is a String etc.
- Referred to composite objects are also imported. In this example ExternalObject has an InnerObject so the definition for that is also created.
- All the fields are created as optional – there is no well-defined mechanism in Java to specify if a field is mandatory or optional.
- All the composite objects are generated with ALL compositors – so that the order in which the fields are defined is not important.

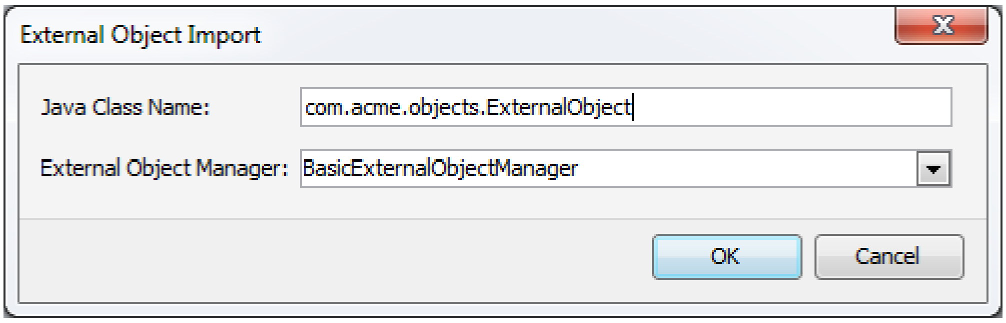
How to Perform An Import

The first thing a user must do before performing an import is add the Classes to be imported to Transformer's classpath. To do so they must go to the "Project" menu and select "Project Properties". They should then select the "Java Class Libraries" Tab. Any class directories or Jar files can then be added to the table. The logical Name is just a place holder but must still be supplied e.g.



Logical Name	Path
ss	C:\code\JerseyTest\jersey-service\target\classes

Now to perform the import the user should go to the "Tools -> Add Ins" menu and choose "External Object Import". The following dialog will be displayed.



Java Class Name: com.acme.objects.ExternalObject

External Object Manager: BasicExternalObjectManager

The user must select an External Object Manager and type in the full name of the class being imported. If they wish to change or create a new External Object Manager then please see the next section.

Configuring the Import Procedure

A number of the aspects of the import procedure are configurable. The import is performed by a Transformer TBean called an ExternalObjectManager. This in turn is comprised of a number of other TBeans which alter how a class is imported. So to import a class in a different way a new ExternalObjectManager can be created and various TBean values adjusted. An ExternalObjectManager is made up of:

- Field Discoverer – this is how the import procedure detects which fields to create in a definition. There are three inbuilt discoverers:
    - PublicSetAndGetMethods – this will import any field that has public set and get Accessors. This is the default FieldDiscoverer. A simple field named XXX will correspond to conventional getXXX and setXXX methods, except for Boolean members for which the get method may be isXXX or just XXX.
    - PublicFields – this will import any field in the class which is a public member variable.
    - NonPrivateSetAndGetWithCollections – this is similar to the PublicSetAndGetMethods discover accept any Public, Protected or Package visible set or get method will be considered. Additionally collections will also be considered. A collection named XXX will correspond to a

method getXXXs which has a return value whose type is a Collection, plus either a setXXXs whose parameter is a Collection or a pair of methods removeAllXXXs and addXXX.

- Group Namer – this is how the Message Definition Group is worked out from the Class being imported. There is only one inbuilt namer "BasicGroupNamer" which just uses the class' package name as the Group name.
- ComponentType Namer – this is how the name of the created Component Type is worked out from the Class being imported. There is only one inbuilt namer "BasicComponentTypeNamer" which uses the Class name as the Component Type name. For inner classes, the "$" is changed to a ".".
- Root Object Factory – this is how a new instance of the class is created at run time. There is one inbuilt Root Object Factory "PublicParameterlessConstructor" which requires the imported class to have a parameterless constructor.
- Class Importer – this controls the basic functionality of the import procedure. Again there is a single inbuilt Importer "BasicClassImport" which imports Java classes as complex NamedComponentTypes with ALL compositor; imports Java enums as NamedComponentTypes which are restrictions of xsd:string with an enumeration facet
- A List of Simple Object Converters (SOC). A SOC is responsible for converting a Java Type to a Transformer Business Object Type. For example there is a SOC that converts a Sql Timestamp (java.sql.Timestamp) to a Transformer DateTime. There is a long list of built in SOCS:
    - BigDecimal – Converts between Java BigDecimal and Transformer's Decimal BOT
    - BigInteger – Converts between Java BigInteger and Transformer's Integer BOT
    - Boolean – Converts between Java Boolean and Transformer's Boolean BOT
    - ByteArray – Converts between Java byte[] and Transformer's Binary BOT
    - CharArray – Converts between Java char[] and Transformer's String BOT
    - Date – Converts between java.util.Date and Transformer's DateTime BOT
    - Double – Converts between Java Double and Transformer's Decimal BOT
    - EnumName – Converts between a Java Enum and Transformer's String field with enum constants
    - Float – Converts between Java Float and Transformer's Decimal BOT
    - Integer – Converts between Java Integer and Transformer's Integer BOT
    - Long – Converts between Java Long and Transformer's Integer BOT
    - PrimitiveBoolean – Converts between Java boolean and Transformer's Boolean BOT
    - PrimitiveDouble – Converts between Java double and Transformer's Decimal BOT
    - PrimitiveFloat – Converts between Java float and Transformer's Decimal BOT
    - PrimitiveInteger – Converts between Java int and Transformer's Integer BOT
    - PrimitiveLong – Converts between Java long and Transformer's Integer BOT
    - PrimitiveShort – Converts between Java short and Transformer's Integer BOT

- Short – Converts between Java Short and Transformer's Integer BOT
- SqlTimestamp – Converts between Java java.sql.Timestamp and Transformer's DateTime BOT
- String – Converts between Java String and Transformer's String BOT

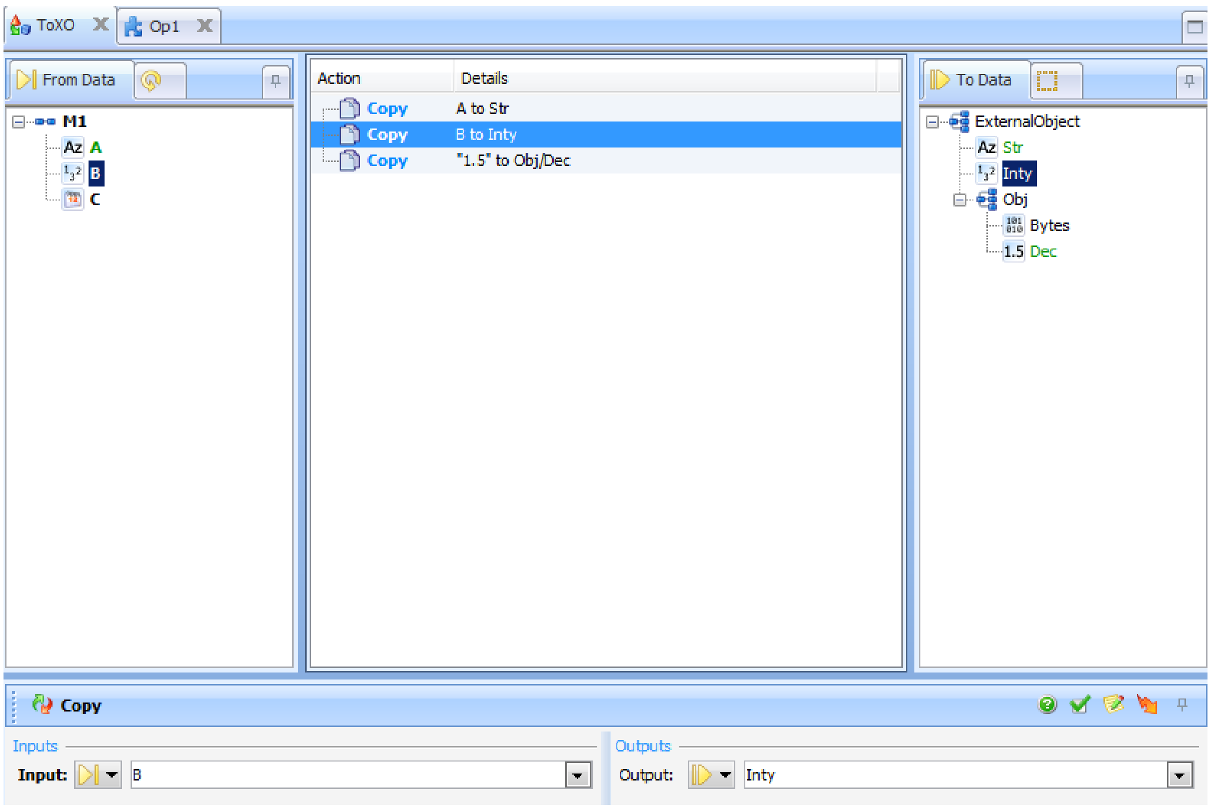
The default ExternalObjectManager is called BasicExternalObjectManager and it uses the PublicSetAndGetMethods field discoverer. If the user wishes to change its behaviour they could override it or copy it and change the details.

A user could also write their own version of any of the previously mentioned TBeans or if they have a specific requirement they could contact Trace Support and request a new TBean.

At Runtime

As with any other definitions objects created via an External Object Import can be used as Inputs or Outputs in mappings. To call the mapping at runtime the user can set up an Exposed Service Operation that calls that mapping and that will expose that mapping to the outside world.

To illustrate assume that the user wishes to create and populate an instance of the ExternalObject class shown in the Overview section. So the mapping might look something like this (the mapping is called ToXO and it is in a mapping group called ToExtObj):

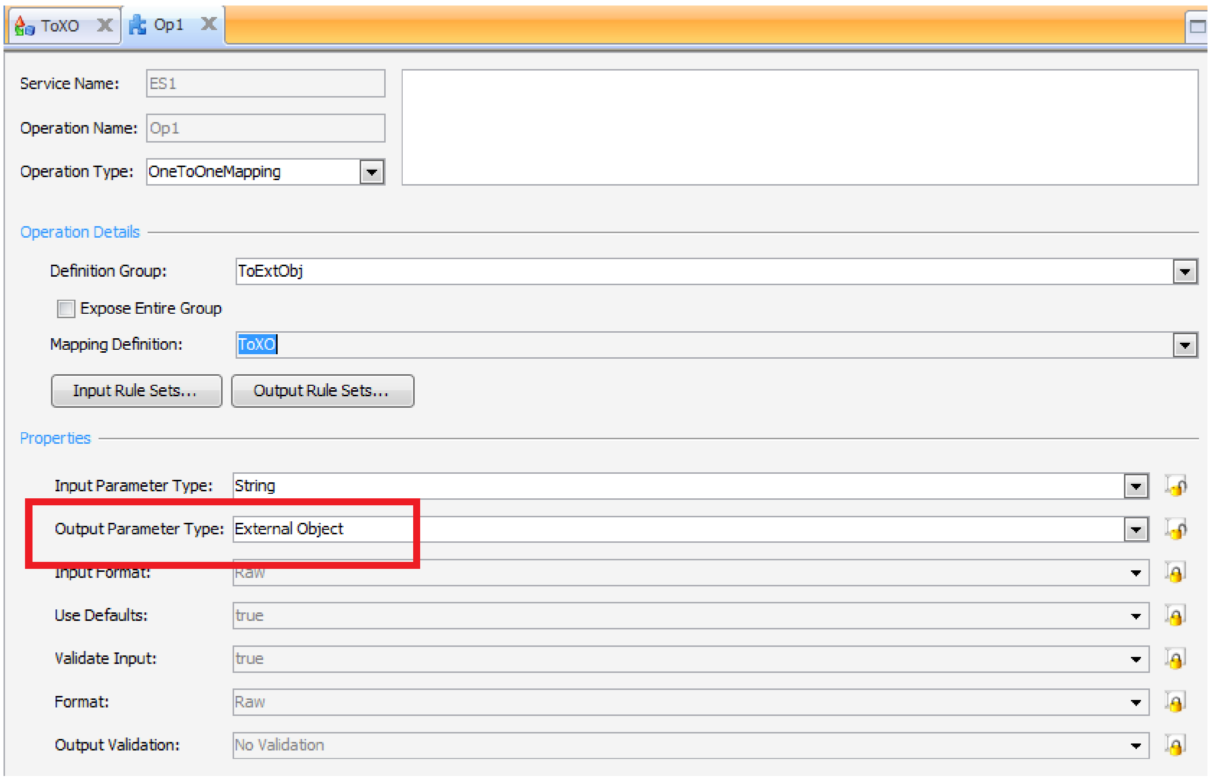


Action	Details
Copy	A to Str
Copy	B to Inty
Copy	"1.5" to Obj/Dec

Input: B

Output: Inty

The user can then create an Exposed Service (called ES1 in this case) that uses the Coarse API Service Builder and within that create an Exposed Service Operation called Op1. The Operation would look something like this:


ToXO | Op1

Service Name:	ES1
Operation Name:	Op1
Operation Type:	OneToOneMapping

Operation Details

Definition Group:	ToExtObj
Expose Entire Group	☐
Mapping Definition:	ToXO

Input Rule Sets... | Output Rule Sets...

Properties

Input Parameter Type:	String
Output Parameter Type:	External Object
Input Format:	Raw
Use Defaults:	true
Validate Input:	true
Format:	Raw
Output Validation:	No Validation

To automatically create an ExternalObject as the output from the Operation the user should choose “External Object” as the Output Parameter Type. If the user chooses to build the code for this service then the generated code (that the user can use to call the mapping) will look like this:

```
public Object op1(String messageData)
```

The returned object can be safely cast to an ExternalObject.

If the user wishes to use the detailed API then two classes are available to convert Transformer MomNodes to their corresponding External Objects. The classes are called ObjectToMom and MomToObject and more details are available upon request.