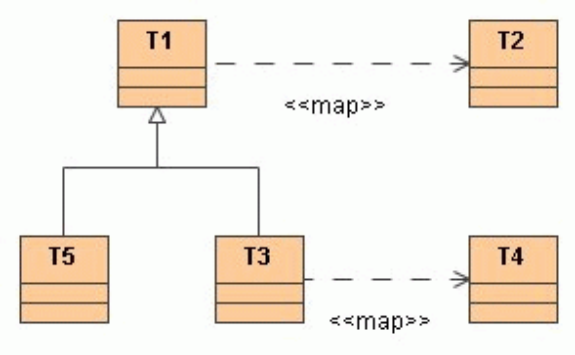
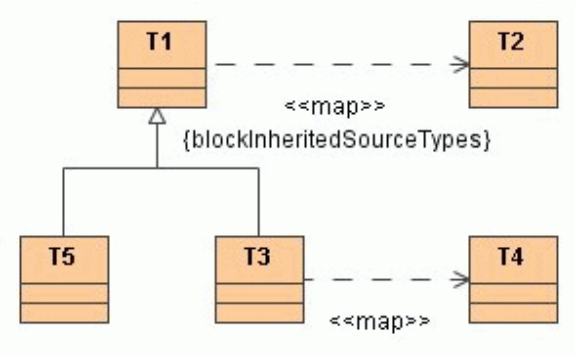
Controlling type inheritance, any, and empty types

You can also control mapping behavior for the type inheritance. By default, derived subtypes are also mapped by the rule governing the parent type (unless, of course, they have their own rules for mapping). If the **blockInheritedSourceTypes** tagged value is set, derived types are not affected by this rule. Let's review the following example:



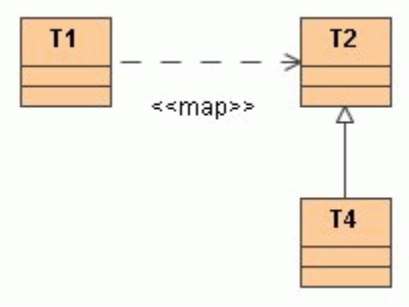
Here T1, T2, and T5 are types in the source domain, while T2 and T4 are types in the destination domain. Given these two mappings (T1 > T2 and T3 > T4), the following statement is true: T1, and all types derived from it (such as T5), are mapped to the T2 type, except T3 and any of the types derived from it. These types are
mapped to T4.

Now consider an example where blockInheritedSourceTypes is set:



In this case, T3, along with the types derived from it, are still mapped to T4. T1 is still mapped to T2. However, unlike the previous example, T5 and all the types inherited from T1 are NOT mapped to T2.

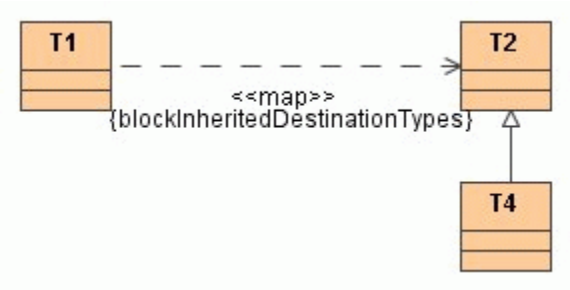
You can also control the mapping behavior of the type inheritance in the destination model. This is only effective on the transformation updates, the second (and successive) reapplications of the transformation. By default, derived subtypes in the destination model are not overwritten, since they are considered suitable substitutes of their parent. Let's review the following example:



Here T1 is a type in the source domain, while T2 and T4 are types in the destination domain. Given this mapping (T1 > T2), on the first application of the transformation, type T1 residing in the source model will be mapped to type T2 in the destination model.

Now let us look at a case, where the user refines the destination model by changing the type on the destination model attribute from T2 to T4. This situation is quite common, for example, the user refines an attribute type from string to basic URI in the XML schema, or from Integer to *nonNegativeInteger*, and so forth. The essence is that the mapping for inherited types of T2 is performed as if there was a mapping T1 > T2 (default), T1 > T4, T1 > <any_other_type_inherited_from_T2>.

Now consider what happens, when we apply the **blockInheritedDestinationTypes** tagged value:



In this case, type T4 has no special treatment. If the user applies the transformation, T1 is mapped to T2. Afterwards the user refines the destination model, changing the attribute type from T2 to T4. If the user now updates the transformation, the attribute type is overwritten: T4 is reset back to T2.

When the user loads the type map in the reverse direction, the roles of the **blockInheritedSourceTypes** and **blockInheritedDestinationTypes** are transposed (unless of course the **direction** tag mandates that this mapping is not used in the reverse direction).

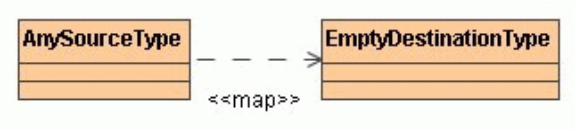
The special type **EmptySourceType** (residing in the *Model_Transformation_Profile.xml.zip*) is used in type maps to indicate that the attributes with no type should be mapped with this dependency.

The special type **EmptyDestinationType** (residing in the *Model_Transformation_Profile.xml.zip*) is used to indicate that the attributes in the destination classes should have no type after remapping (type removal).

The special type **AnySourceType** is a template that matches any type in the source model (see mapping rules for type inheritance). By using this type, together with the inheritance mapping rules, the user can specify that any other types not defined by the mapping should be interpreted by this mapping.

The special type **AnyDestinationType** is a template that matches any type in the destination model (see mapping rules for type inheritance).

Here is an example of template type usage:



According to this rule, any types in the source model for which there are no other mapping rules should be stripped in the destination model.