FreezeScript Transformation XML Reference

This page describes the XML elements comprising the FreezeScript transformation descriptors.

On this page:

`<transformdb>` Descriptor Element

The top-level descriptor in a descriptor file. It requires at least one `<database>` descriptor, and supports any number of `<transform>` and `<init>` child descriptors. This descriptor has no attributes.

`<database>` Descriptor Element

The attributes of this descriptor define the old and new key and value types for the database to be transformed, and optionally the name of the database to which these types apply. It supports any number of child descriptors, but at most one `<record>` descriptor. The `<database>` descriptor also creates a global scope for user-defined symbols.

The attributes supported by the `<database>` descriptor are described in the following table:

Name	Description
`name`	Specifies the name of the database defined by this descriptor. (Optional)
`key`	Specifies the Slice types of the old and new keys. If the types are the same, only one needs to be specified. Otherwise, the types are separated by a comma.
`value`	Specifies the Slice types of the old and new values. If the types are the same, only one needs to be specified. Otherwise, the types are separated by a comma.

As an example, consider the following `<database>` descriptor. In this case, the Freeze map to be transformed currently has key type `int` and value type `::Employee`, and is migrating to a key type of `string`:

XML

```
<database key="int,string" value="::Employee">
```

`<record>` Descriptor Element

Commences the transformation. Child descriptors are executed for each record in the database, providing the user with an opportunity to examine the record's old key and value, and optionally modify the new key and value. Default transformations, as well as `<transform>` and `<init>` descriptors, are executed before the child descriptors. The `<record>` descriptor introduces the following symbols into a local scope: `oldkey`, `newkey`, `oldvalue`, `newvalue`, `facet`. These symbols are accessible to child descriptors, but not to `<transform>` or `<init>` descriptors. The `oldkey` and `oldvalue` symbols are read-only. The `facet` symbol is a string indicating the facet name of the object in the current record, and is only relevant for Freeze evictor databases.

Use caution when modifying database keys to ensure that duplicate keys do not occur. If a duplicate database key is encountered, transformation fails immediately.

Note that database transformation only occurs if a `<record>` descriptor is present.

<transform> Descriptor Element

Customizes the transformation for all instances of a type in the *new* Slice definitions. The children of this descriptor are executed after the optional default transformation has been performed. Only one <transform> descriptor can be specified for a type, but a <transform> descriptor is not required for every type. The symbols old and new are introduced into a local scope and represent the old and new values, respectively. The old symbol is read-only. The attributes supported by this descriptor are described in the following table:

Name	Description
type	Specifies the Slice type ID for the type's new definition.
default	If false, no default transformation is performed on values of this type. If not specified, the default value is true.
base	This attribute determines whether <transform> descriptors of base class types are executed. If true, the <transform> descriptor of the immediate base class is invoked. If no descriptor is found for the immediate base class, the class hierarchy is searched until a descriptor is found. The execution of any base class descriptors occurs after execution of this descriptor's children. If not specified, the default value is true.
rename	Indicates that a type in the old Slice definitions has been renamed to the new type identified by the type attribute. The value of this attribute is the type ID of the old Slice definition. Specifying this attribute relaxes the strict compatibility rules for enum, struct and class types.

Below is an example of a <transform> descriptor that initializes a new data member:

XML

```
<transform type="::Product">
    <set target="new.salePrice" value="old.listPrice * old.discount"/>
</transform>
```

For class types, transformdb first attempts to locate a <transform> descriptor for the object's most-derived type. If no descriptor is found, transformdb proceeds up the class hierarchy in an attempt to find a descriptor. The base object type, Object, is the root of every class hierarchy and is included in the search for descriptors. It is therefore possible to define a <transform> descriptor for type Object, which will be invoked for every class instance.

Note that <transform> descriptors are executed recursively. For example, consider the following Slice definitions:

Slice

```
struct Inner {
    int sum;
};
struct Outer {
    Inner i;
};
```

When transformdb is performing the default transformation on a value of type Outer, it recursively performs the default transformation on the Inne member, then executes the <transform> descriptor for Inner, and finally executes the <transform> descriptor for Outer. However, if default transformation is disabled for Outer, then no transformation is performed on the Inner member and therefore the <transform> descriptor for Inne is not executed.

<init> Descriptor Element

Defines custom initialization rules for all instances of a type in the new Slice definitions. Child descriptors are executed each time the type is instantiated. The typical use case for this descriptor is for types that have been introduced in the new Slice definitions and whose instances require default values different than what transformdb supplies. The symbol value is introduced into a local scope to represent the instance. The attributes supported by this descriptor are described in the following table:

Name	Description
type	Specifies the Slice type ID of the type's new definition.

Here is a simple example of an <init> descriptor:

XML

```
<init type="::Player">
    <set target="value.currency" value="100"/>
</init>
```

Note that, like `<transform>`, `<init>` descriptors are executed recursively. For example, if an `<init>` descriptor is defined for a `struct` type, the `<init>` descriptors of the `struct`'s members are executed before the `struct`'s descriptor.

`<iterate>` Descriptor Element

Iterates over a dictionary or sequence, executing child descriptors for each element. The symbol names selected to represent the element information may conflict with existing symbols in the enclosing scope, in which case those outer symbols are not accessible to child descriptors. The attributes supported by this descriptor are described in the following table:

Name	Description
`target`	The sequence or dictionary.
`index`	The symbol name used for the sequence index. If not specified, the default symbol is `i`.
`element`	The symbol name used for the sequence element. If not specified, the default symbol is `elem`.
`key`	The symbol name used for the dictionary key. If not specified, the default symbol is `key`.
`value`	The symbol name used for the dictionary value. If not specified, the default symbol is `value`.

Shown below is an example of an `<iterate>` descriptor that sets the new data member `reviewSalary` to `true` if the employee's salary is greater than $3000:

XML

```
<iterate target="new.employeeMap" key="id" value="emp">
    <if test="emp.salary > 3000">
        <set target="emp.reviewSalary" value="true"/>
    </if>
</iterate>
```

`<if>` Descriptor Element

Conditionally executes child descriptors. The attributes supported by this descriptor are described in the following table:

Name	Description
`test`	A boolean expression.

Child descriptors are executed if the expression in `test` evaluates to true.

`<set>` Descriptor Element

Modifies a value. The `value` and `type` attributes are mutually exclusive. If `target` denotes a dictionary element, that element must already exist (i.e., `<set>` cannot be used to add an element to a dictionary). The attributes supported by this descriptor are described in the following table:

Name	Description
`target`	An expression that must select a modifiable value.
`value`	An expression that must evaluate to a value compatible with the target's type.
`type`	If specified, set the target to be an instance of the given Slice class. The value is a type ID from the new Slice definitions. The class must be compatible with the target's type.

`length`	An integer expression representing the desired new length of a sequence. If the new length is less than the current size of the sequence, elements are removed from the end of the sequence. If the new length is greater than the current size, new elements are added to the end of the sequence. If `value` or `type` is also specified, it is used to initialize each new element.
`convert`	If `true`, additional type conversions are supported: between integer and floating point, and between integer and enumeration. Transformation fails immediately if a range error occurs. If not specified, the default value is `false`.

The `<set>` descriptor below modifies a member of a dictionary element:

XML

```
<set target="new.parts['P105J3'].cost" value="new.parts['P105J3'].cost * 1.05"/>
```

This `<set>` descriptor adds an element to a sequence and initializes its value:

XML

```
<set target="new.partsList" length="new.partsList.length + 1" value="'P105J3'"/>
```

As another example, the following `<set>` descriptor changes the value of an enumeration:

XML

```
<set target="new.ingredient" value="::New::Apple"/>
```

Notice in this example that the value refers to a symbol in the new Slice definitions.

<add> Descriptor Element

Adds a new element to a sequence or dictionary. It is legal to add an element while traversing the sequence or dictionary using `<iterate>`, however the traversal order after the addition is undefined. The `key` and `index` attributes are mutually exclusive, as are the `value` and `type` attributes. If neither `value` nor `type` is specified, the new element is initialized with a default value. The attributes supported by this descriptor are described in the following table:

Name	Description
`target`	An expression that must select a modifiable sequence or dictionary.
`key`	An expression that must evaluate to a value compatible with the target dictionary's key type.
`index`	An expression that must evaluate to an integer value representing the insertion position. The new element is inserted before `index`. The value must not exceed the length of the target sequence.
`value`	An expression that must evaluate to a value compatible with the target dictionary's value type, or the target sequence's element type.
`type`	If specified, set the target value or element to be an instance of the given Slice class. The value is a type ID from the new Slice definitions. The class must be compatible with the target dictionary's value type, or the target sequence's element type.
`convert`	If `true`, additional type conversions are supported: between integer and floating point, and between integer and enumeration. Transformation fails immediately if a range error occurs. If not specified, the default value is `false`.

Below is an example of an `<add>` descriptor that adds a new dictionary element and then initializes its member:

XML

```
<add target="new.parts" key="'P105J4'"/>
<set target="new.parts['P105J4'].cost" value="3.15"/>
```

<define> Descriptor Element

Defines a new symbol in the current scope. The attributes supported by this descriptor are described in the following table:

Name	Description
`name`	The name of the new symbol. An error occurs if the name matches an existing symbol in the current scope.
`type`	The name of the symbol's formal Slice type. For user-defined types, the name should be prefixed with `::Old` or `::New` to indicate the source of the type. The prefix can be omitted for primitive types.
`value`	An expression that must evaluate to a value compatible with the symbol's type.
`convert`	If `true`, additional type conversions are supported: between integer and floating point, and between integer and enumeration. Execution fails immediately if a range error occurs. If not specified, the default value is `false`.

Below are two examples of the `<define>` descriptor. The first example defines the symbol `identity` to have type `Ice::Identity`, and proceeds to initialize its members using `<set>`:

XML

```
<define name="identity" type="::New::Ice::Identity"/>
<set target="identity.name" value="steve"/>
<set target="identity.category" value="Admin"/>
```

The second example uses the enumeration we first saw in our discussion of custom database migration to define the symbol `manufacturer` and assign it a default value:

XML

```
<define name="manufacturer" type="::New::BigThree" value="::New::Daimler"/>
```

`<remove>` Descriptor Element

Removes an element from a sequence or dictionary. It is legal to remove an element while traversing a sequence or dictionary using `<iterate>`, however the traversal order after removal is undefined. The attributes supported by this descriptor are described in the following table:

Name	Description
`target`	An expression that must select a modifiable sequence or dictionary.
`key`	An expression that must evaluate to a value compatible with the key type of the target dictionary.
`index`	An expression that must evaluate to an integer value representing the index of the sequence element to be removed.

`<fail>` Descriptor Element

Causes transformation to fail immediately. If `test` is specified, transformation fails only if the expression evaluates to `true`. The attributes supported by this descriptor are described in the following table:

Name	Description
`message`	A message to display upon transformation failure.
`test`	A boolean expression.

The following `<fail>` descriptor terminates the transformation if a range error is detected:

XML

```
<fail message="range error occurred in ticket count!" test="old.ticketCount > 32767"/>
```

<delete> Descriptor Element

Causes transformation of the current database record to cease, and removes the record from the transformed database. This descriptor has no attributes.

<echo> Descriptor Element

Displays values and informational messages. If no attributes are specified, only a newline is printed. The attributes supported by this descriptor are described in the following table:

Name	Description
message	A message to display.
value	An expression. The value of the expression is displayed in a structured format.

Shown below is an <echo> descriptor that uses both message and value attributes:

XML

```
<if test="old.ticketCount > 32767">
    <echo message="deleting record with invalid ticket count: " value="old.ticketCount"/>
    <delete/>
</if>
```

See Also

- Custom Database Migration
- Freeze Maps
- Freeze Evictors
- Automatic Database Migration
- FreezeScript Descriptor Expression Language