Using transformdb

On this page:

Execution Modes for `transformdb`

The tool operates in one of three modes:

- Automatic migration
- Custom migration
- Analysis

The only difference between automatic and custom migration modes is the source of the transformation descriptors: for automatic migration, `transformdb` internally generates and executes a default set of descriptors, whereas for custom migration the user specifies an external file containing the transformation descriptors to be executed.

In analysis mode, `transformdb` creates a file containing the default transformation descriptors it would have used during automatic migration. You would normally review this file and possibly customize it prior to executing the tool again in its custom migration mode.

Using Database Catalogs during Transformation

Freeze maintains schema information in a catalog for each database environment. If necessary, `transformdb` will use the catalog to determine the names of the databases in the environment, and to determine the key and value types of a particular database. There are two advantages to the tool's use of the catalog:

- Allows `transformdb` to operate on all of the databases in a single invocation
- Eliminates the need for you to specify type information for a database.

For example, you can use automatic migration to transform all of the databases at one time, as shown below:

```
$ transformdb [options] old-env new-env
```

Since we omitted the name of a database to be migrated, `transformdb` uses the catalog in the environment `old-env` to discover all of the databases and their types, generates default transformations for each database, and performs the migration. However, we must still ensure that `transformdb` has loaded the old and new Slice types used by *all* of the databases in the environment.

Slice Options for `transformdb`

The tool supports the standard command-line options common to all Slice processors, with the exception of the include directory (`-I`) option. The options specific to `transformdb` are described below:

- `--old` *`SLICE`*
  `--new` *`SLICE`*
  Loads the old or new Slice definitions contained in the file *`SLICE`*. These options may be specified multiple times if several files must be loaded. However, it is the user's responsibility to ensure that duplicate definitions do not occur (which is possible when two files are loaded that share a common include file). One strategy for avoiding duplicate definitions is to load a single Slice file that contains only `#include` statements for each of the Slice files to be loaded. No duplication is possible in this case if the included files use include guards correctly.

- `--include-old` *DIR*
  `--include-new` *DIR*
  Adds the directory *DIR* to the set of include paths for the old or new Slice definitions.

Type Options for `transformdb`

In invocation modes for which `transformdb` requires that you define the types used by a database, you must specify one of the following options:

- `--key` *TYPE*`[,`*TYPE*`]`
  `--value` *TYPE*`[,`*TYPE*`]`
  Specifies the Slice type(s) of the database key and value. If the type does not change, then the type only needs to be specified once. Otherwise, the old type is specified first, followed by a comma and the new type. For example, the option `--key int,string` indicates that the database key is migrating from `int` to `string`. On the other hand, the option `--key int,int` indicates that the key type does not change, and could be given simply as `--key int`. Type changes are restricted to those allowed by the compatibility rules, but custom migration provides additional flexibility.

- `-e`
  Indicates that a Freeze evictor database is being migrated. As a convenience, this option automatically sets the database key and value types to those appropriate for the Freeze evictor, and therefore the `--key` and `--value` options are not necessary. Specifically, the key type of a Freeze evictor database is `Ice::Identity`, and the value type is `Freeze::ObjectRecord`. The latter is defined in the Slice file `Freeze/EvictorStorage.ice`; however, this file does not need to be loaded into your old and new Slice definitions.

General Options for `transformdb`

These options may be specified during analysis or migration, as indicated below:

- `-i`
  Requests that `transformdb` ignore type changes that violate the compatibility rules. If this option is not specified, `transformdb` fails immediately if such a violation occurs. With this option, a warning is displayed but `transformdb` continues the requested action. The `-i` option can be specified in analysis or automatic migration modes.

- `-p`
  During migration, this option requests that `transformdb` purge object instances whose type is no longer found in the new Slice definitions.

- `-c`
  Use catastrophic recovery on the old Berkeley DB database environment prior to migration.

- `-w`
  Suppress duplicate warnings during migration. This option is especially useful to minimize diagnostic messages when `transformdb` would otherwise emit the same warning many times, such as when it detects the same issue in every record of a database.

Database Arguments for `transformdb`

In addition to the options described above, `transformdb` accepts as many as three arguments that specify the names of databases and database environments:

- `dbenv`
  The pathname of the old database environment directory.

- `db`
  The name of an existing database file in `dbenv`. `transformdb` never modifies this database.

- `newdbenv`
  The pathname of the database environment directory to contain the transformed database(s). This directory must exist and must not contain an existing database whose name matches a database being migrated.

Performing an Automatic Migration

You can use `transformdb` to automatically migrate one database or all databases in an environment.

Migrating a Single Database

Use the following command line to migrate one database:

```
$ transformdb [slice-opts] [type-opts] [gen-opts] dbenv db newdbenv
```

If you omit `type-opts`, the tool obtains type information for database `db` from the catalog. For example, consider the following command, which uses automatic migration to transform a database with a key type of `int` and value type of `string` into a database with the same key type and a value type of `long`:

```
$ transformdb --key int --value string,long dbhome data.db newdbhome
```

Note that we did not need to specify the Slice options `--old` or `--new` because our key and value types are primitives. Upon successful completion, the file `newdbhome/data.db` contains our transformed database.

Migrating All Databases

To migrate all databases in the environment, use a command like the one shown below:

```
$ transformdb [slice-opts] [gen-opts] dbenv newdbenv
```

In this invocation mode, you must ensure that `transformdb` has loaded the old and new Slice definitions for all of the types it will encounter among the databases in the environment.

Performing a Migration Analysis

Custom migration is a two-step process: you first write the transformation descriptors, and then execute them to transform a database. To assist you in the process of creating a descriptor file, `transformdb` can generate a default set of transformation descriptors by comparing your old and new Slice definitions. This feature is enabled by specifying the following option:

- `-o` *`FILE`*
  Specifies the descriptor file *`FILE`* to be created during analysis. No migration occurs in this invocation mode.

Generated File

The generated file contains a `<transform>` descriptor for each type that appears in both old and new Slice definitions, and an `<init>` descriptor for types that appear only in the new Slice definitions. In most cases, these descriptors are empty. However, they can contain XML comments describing changes detected by `transformdb` that may require action on your part.

For example, let us revisit the enumeration we defined in our discussion of custom database migration:

Slice

```
enum BigThree { Ford, DaimlerChrysler, GeneralMotors };
```

This enumeration has evolved into the one shown below. In particular, the `DaimlerChrysler` enumerator has been renamed to reflect a corporate name change:

Slice

```
enum BigThree { Ford, Daimler, GeneralMotors };
```

Next we run `transformdb` in analysis mode:

```
$ transformdb --old old/BigThree.ice --new new/BigThree.ice --key string \
--value ::BigThree -o transform.xml
```

The generated file `transform.xml` contains the following descriptor for the enumeration `BigThree`:

XML

```
<transform type="::BigThree">
    <!-- NOTICE: enumerator `DaimlerChrysler' has been removed -->
</transform>
```

The comment indicates that enumerator `DaimlerChrysler` is no longer present in the new definition, reminding us that we need to add logic in this `<transform>` descriptor to change all occurrences of `DaimlerChrysler` to `Daimler`.

The descriptor file generated by `transformdb` is well-formed and does not require any manual intervention prior to being executed. However, executing an unmodified descriptor file is simply the equivalent of using automatic migration.

Invocation Modes

The sample command line shown in the previous section specified the key and value types of the database explicitly. This invocation mode has the following general form:

```
$ transformdb [slice-opts] [type-opts] [gen-opts] -o FILE
```

Upon successful completion, the generated file contains a `<database>` descriptor that records the type information supplied by `type-opts`, in addition to the `<transform>` and `<init>` descriptors described earlier.

For your convenience, you can omit `type-opts` and allow `transformdb` to obtain type information from the catalog instead:

```
$ transformdb [slice-opts] [gen-opts] -o FILE dbenv
```

In this case, the generated file contains a `<database>` descriptor for each database in the catalog. Note that in this invocation mode, `transformdb` must assume that the names of the database key and value types have not changed, since the only type information available is the catalog in the old database environment. If the tool is unable to locate a new Slice definition for a database's key or value type, it emits a warning message and generates a placeholder value in the output file that you must modify prior to migration.

Performing a Custom Migration

After preparing a descriptor file, either by writing one completely yourself, or modifying one generated by the analysis mode described in the previous section, you are ready to migrate a database. One additional option is provided for migration:

- `-f` *`FILE`*
  Execute the transformation descriptors in the file *`FILE`*.

To transform one database, use the following command:

```
$ transformdb [slice-opts] [gen-opts] -f FILE dbenv db newdbenv
```

The tool searches the descriptor file for a `<database>` descriptor whose `name` attribute matches `db`. If no match is found, it searches for a `<database>` descriptor that does not have a `name` attribute.

If you want to transform all databases in the environment, you can omit the database name:

```
$ transformdb [slice-opts] [gen-opts] -f FILE dbenv newdbenv
```

In this case, the descriptor file must contain a `<database>` element for each database in the environment.

Continuing our enumeration example from the analysis discussion above, assume we have modified `transform.xml` to convert the `Chrysler` enumerator, and are now ready to execute the transformation:

```
$ transformdb --old old/BigThree.ice --new new/BigThree.ice -f transform.xml \
dbhome bigthree.db newdbhome
```

`transformdb` Usage Strategies

If it becomes necessary for you to transform a Freeze database, we generally recommend that you attempt to use automatic migration first, unless you already know that custom migration is necessary. Since transformation is a non-destructive process, there is no harm in attempting an automatic migration, and it is a good way to perform a sanity check on your `transformdb` arguments (for example, to ensure that all the necessary Slice files are being loaded), as well as on the database itself. If `transformdb` detects any incompatible type changes, it displays an error message for each incompatible change and terminates without doing any transformation. In this case, you may want to run `transformdb` again with the `-i` option, which ignores incompatible changes and causes transformation to proceed.

Pay careful attention to any warnings that `transformdb` emits, as these may indicate the need for using custom migration. For example, if we had attempted to transform the database containing the `BigThree` enumeration from previous sections using automatic migration, any occurrences of the `Chrysler` enumerator would display the following warning:

```
warning: unable to convert 'Chrysler' to ::BigThree
```

If custom migration appears to be necessary, use analysis to generate a default descriptor file, then review it for `NOTICE` comments and edit as necessary. Liberal use of the `<echo>` descriptor can be beneficial when testing your descriptor file, especially from within the `<record>` descriptor where you can display old and new keys and values.

Transforming Objects

The polymorphic nature of Slice classes can cause problems for database migration. As an example, the Slice parser can ensure that a set of Slice definitions loaded into `transformdb` is complete for all types but classes (and exceptions, but we ignore those because they are not persistent). `transformdb` cannot know that a database may contain instances of a subclass that is derived from one of the loaded classes but whose definition is not loaded. Alternatively, the type of a class instance may have been renamed and cannot be found in the new Slice definitions.

By default, these situations result in immediate transformation failure. However, the `-p` option is a (potentially drastic) way to handle these situations: if a class instance has no equivalent in the new Slice definitions and this option is specified, `transformdb` removes the instance any way it can. If the instance appears in a sequence or dictionary element, that element is removed. Otherwise, the database record containing the instance is deleted.

Now, the case of a class type being renamed is handled easily enough using custom migration and the `rename` attribute of the `<transform>` descriptor. However, there are legitimate cases where the destructive nature of the `-p` option can be useful. For example, if a class type has been removed and it is simply easier to start with a database that is guaranteed not to contain any instances of that type, then the `-p` option may simplify the broader migration effort.

This is another situation in which running an automatic migration first can help point out the trouble spots in a potential migration. Using the `-p` option, `transformdb` emits a warning about the missing class type and continues, rather than halting at the first occurrence, enabling you to discover whether you have forgotten to load some Slice definitions, or need to rename a type.

Using `transformdb` on an Open Environment

It is possible to use `transformdb` to migrate databases in an environment that is currently open by another process, but if you are not careful you can easily corrupt the environment and cause the other process to fail. To avoid such problems, you must configure both `transformdb` and the other process to set `Freeze.DbEnv.`*`env-name`*`.DbPrivate=0`. This property has a default value of one, therefore you must explicitly set it to zero. Note that `transformdb` makes no changes to the existing database environment, but it requires exclusive access to the new database environment until transformation is complete.

If you run `transformdb` on an open environment but neglect to set `Freeze.DbEnv.`*`env-name`*`.DbPrivate=0`, you can expect `transformdb` to terminate immediately with an error message stating that the database environment is locked. Before running `transformdb` on an open environment, we strongly recommend that you first verify that the other process was also configured with `Freeze.DbEnv.`*`env-name`*`.DbPrivate=0`.

See Also

- Automatic Database Migration
- Custom Database Migration
- Using the Slice Compilers
- Freeze Catalogs
- Freeze Evictors
- Freeze Properties