U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology

NIST PUBLICATIONS

NISTIR 4944

NAT'L INST. OF STAND & TECH R.I.C.
A11103 874068

REFERENCE

National PDES Testbed Report Series

Transformr: A Prototype STEP Exchange File Migration Tool

Stephen Nowland Clark

NATIONAL PDES TESTBED

QC
100
.U56
4944
1992

NIST

NIST-IR
QC
100
.U56
4944
1992

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology

NISTIR 4944

National PDES Testbed

NATIONAL PDES TESTBED

Transformr: A Prototype STEP Exchange File Migration Tool

Stephen Nowland Clark

U.S. DEPARTMENT OF COMMERCE
Barbara Hackman Franklin,
Secretary of Commerce

Technology Administration
Robert M. White,
Under Secretary for Technology

National Institute of Standards and Technology
John W. Lyons, Director

October 13, 1992

Transformr: A Prototype STEP Exchange File Migration Tool

Stephen Nowland Clark

1 Introduction

Transformr is a prototype tool for migrating STEP exchange files [ISO21] between different versions of an EXPRESS [ISO11] schema.[1] Its inputs are source and target EXPRESS schemas and a specification of the transformations which must be applied to map a model from the source to the target schema. *Transformr* then reads exchange files corresponding to the source schema and writes files corresponding to the target schema.

This document is primarily a user's guide to *Transformr*. It describes the transformation specification language used to specify the correspondences between the two schemas and the command-line syntax for invoking *Transformr*. After discussing some limitations of the tool, a brief overview of the theory of operation of the transformation engine is given, to give the user some context for understanding the messages which the tool may produce.

It must be understood that *Transformr* is only a prototype. There are features of the transformation specification language which are currently unimplemented, and while these missing features and other known limitations are usually reported to the user when encountered, this is not always the case. Similarly, *Transformr* is not yet robust, and may encounter unexpected circumstances.

1.1 Motivation

STEP information models tend to undergo constant testing during their development. As a result, a problem arises which is similar to the legacy data problem: As a model changes, test cases written against it (which may exist in databases or in STEP exchange files) must constantly be either recreated from scratch or massaged to take into account these changes. This process is time consuming and error prone.

Given some information about the changes which have been made to a model, it is often possible to automatically transform at least some of the associated instance data, easing the task of testing a model in a constant state of flux.

---

1. Funding for the work described has been provided by the Department of Defense's Computer-Aided Acquisition and Logistic Support (CALS) Office. The work is funded by the United States Government and is not subject to copyright.

Clearly, it is not always possible to completely transform all data as required. To take an extreme case, entities might be added which represent information which was simply not represented in the earlier revision; instance data for these entities certainly cannot be automatically conjured out of thin air. Nevertheless, there are a number of changes which can be handled automatically. *Transformr* is a tool which addresses this problem.

2 The *Transformr* Correspondence Specification Language

The *Transformr* Correspondence Specification Language (TCSL) is a language for specifying mappings between related EXPRESS schemas. For various reasons, it owes a large debt to EXPRESS itself. TCSL builds tokens in the same way as EXPRESS, so that identifiers, character strings, numbers, and so forth all look familiar. Comments are written as in EXPRESS, enclosed between `(*` and `*)`. TCSL also borrows its entire expression syntax from EXPRESS. Here the similarities end, however, and a TCSL specification looks quite different from an EXPRESS schema.

There are two primary operations in TCSL: `COPY` and `BUILD`. `COPY` establishes a direct mapping from instances of one entity in the source schema to instances of one entity in the target schema. It specifies that each instance of the source entity is to be transformed into an instance of the target entity. The resulting instance retains its prior identity: in an exchange file, this means that it retains the same numeric identifier. `BUILD`, on the other hand, specifies a construction for instances of one entity in the target schema based on tuples of instances from the source schema. Source instances are left untouched, and an instance with a new identity is constructed. This distinction will become clearer in the following discussion of the two commands.

A TCSL specification consists simply of a series of instances of these two commands placed one after the other:[1]

```
tcsl-spec = { copy-decl | build-decl }
```

2.1 The `COPY` Command

The `COPY` command has the following syntax:

```
copy-decl = bulk-copy | single-copy .
bulk-copy = 'COPY' copy-from { ',' copy-from } ';' .
single-copy = 'COPY' copy-from ';' { modifier ';' } .
copy-from = target-entity-id [ 'FROM' source-entity-id ] .
```

1. The syntax of TCSL is given formally in Wirth Syntax Notation (WSN) [Wirth]. In WSN, a grammar is represented as a collection of productions, where the language element to the left of '=' can be rewritten as the sequence on the right. Literals of the language are written between single quotes. Square brackets indicate an optional element, and curly braces indicate an element which may be repeated 0 or more times. The vertical bar indicates that a selection is to be made between the two elements surrounding it.

In its simplest form, 'COPY <target-entity-id>', the COPY command specifies that instances of the named entity in the source schema will become instances of the entity of the same name in the target schema. Any explicit attribute which is declared locally in either the source or the target entity, and which appears with the same name and compatible or coercible types in both entities (whether declared locally or inherited) will be copied; others will not. In general, types which are assignment compatible in either direction in EXPRESS itself are considered to be coercible in TCSL. This is discussed in more detail in section 4.

A COPY command inherits any COPY commands in place between any supertypes of its source and target entities.

Example 1: Simple COPY with inheritance

Suppose we had these two schemas:

Source Schema	Target Schema

`SCHEMA people_schema;`
 	
`ENTITY person`
`SUPERTYPE OF (ONEOF(man,`
`                  woman));`
`  name : STRING;`
`  age : INTEGER;`
`END_ENTITY;`
 	
`ENTITY man`
`SUBTYPE OF (person);`
`  masculinity : INTEGER;`
`  size : INTEGER;`
`END_ENTITY;`
 	
`ENTITY woman`
`SUBTYPE OF (person);`
`  size : INTEGER;`
`  femininity : INTEGER;`
`END_ENTITY;`
 	
`END_SCHEMA; -- people_schema`
	`SCHEMA people_schema;`
	
	`ENTITY person`
	`SUPERTYPE OF(ONEOF(man,`
	`                 woman));`
	`  age : REAL;`
	`  name : STRING;`
	`  size : INTEGER;`
	`END_ENTITY;`
	
	`ENTITY man`
	`SUBTYPE OF (person);`
	`  masculinity : INTEGER;`
	`END_ENTITY;`
	
	`ENTITY woman`
	`SUBTYPE OF (person);`
	`  femininity : INTEGER;`
	`END_ENTITY;`
	
	`END_SCHEMA; -- people_schema`

In migrating instances from the first schema to the second, we'd like to keep all of the attribute values: clearly, we expect age and name to be retained for instances of person, and masculinity/femininity for instances of man/woman. In addition, size can be retained for instances of man and of woman, since this attribute simply moves up into the common supertype, person. We can write the following simple TCSL specification:

```
COPY person;
COPY man;
COPY woman;
```

This instructs *Transformr* to keep any instances of person, man, or woman. The new person instances will have age and name copied from the old instances (the only attributes common to the source and target person entities). Note that age is retained despite its change in type, since INTEGER is coercible to REAL. New man instances will retain these two attributes (by inheriting the COPY command from person) as well as masculinity (common to the source and target man entities) and size (common to the source man and target person entities). New woman instances will similarly be fully populated. Note in particular that the COPY command need not concern itself with changes in the order of attributes, or with simple movement of attributes up or down the class hierarchy.

Adding 'FROM source-entity-id' to a COPY command simply generalizes the operation so that the source and target entities need not have the same name. Provided that no other modifications need to be made to the mapped instances, several of these simple specifications (with or without FROM specifications) may be strung together separated by commas following a single COPY keyword.

Example 2: Bulk COPY with renaming

In the previous example, we could more compactly have written:

```
COPY person, man, woman;
```

achieving the same effect.

Suppose now that the target schema defines subtypes male and female of person, rather than the original man and woman. We still want to retain instances of man/woman, but they must be transformed into new instances of male/female. This is again straightforward. We can simply write:

```
COPY person, male FROM man, female FROM woman;
```

Once again, the desired instances will be retained with all of their proper attribute values.

Many of the interesting changes which are made to schemas involve rearranging, adding, or deleting attributes from entities. These sorts of operations can be specified by adding modifiers to a single COPY command. We now turn to these modifiers.

2.1.1 Modifiers (Derive and Drop)

There are two forms of modifiers in TCSL, derivations and drops. Both can used in the COPY command, while only derivations are meaningful to the BUILD command (see section 2.2). These modifiers have the following syntax:

```
modifier = derive | drop .
derive = special-ref ':=' expression .
drop = 'DROP' special-ref { ',' special-ref } .
special-ref = attr-id { qualifier } .
entity-id = source-entity-id | 'SELF' .
```

As previously mentioned, TCSL expressions are the same as EXPRESS expressions. Similarly, TCSL qualifiers are identical to EXPRESS qualifiers, although group qualification and aggregate indexing are currently unsupported in the prototype implementation.

The only entity identifiers which are valid in either type of modifier are the names of the target (left-hand side of a derive modifier) and source (elsewhere) entities. SELF is synonymous with the name of the source entity within a COPY command.

2.1.1.1 Derive

A derive modifier (derivation) is used to specify a value for an attribute of the target instance or of one of its attributes which cannot or should not be directly copied from an attribute of the source instance. It specifies an expression in terms of attributes of the source instance which is used to compute the new attribute value.

Example 3: Derive modifiers

Imagine that we have a geometry schema which uses polar coordinates, and we wish to change the schema to use rectangular coordinates instead. Then we might have the schema excerpts below:

Source Schema	Target Schema

`SCHEMA geometry;`
 	
`ENTITY point;`
`  r : REAL;`
`  theta : REAL;`
`END_ENTITY;`
 	
`...`
 	
`END_SCHEMA; -- geometry`
	`SCHEMA geometry;`
	
	`ENTITY point;`
	`  x_coord : REAL;`
	`  y_coord : REAL;`
	`END_ENTITY;`
	
	`...`
	
	`END_SCHEMA; -- geometry`

In order to properly transform instances of `point`, we must provide derivations for the new attributes. Thus, we might write:

```
COPY point;
x_coord := r*cos(point.theta); -- derivation for x_coord
y_coord := r*sin(SELF.theta); -- derivation for y_coord
```

Note that `SELF` and `point` may be used interchangeably to refer to the instance being transformed.

2.1.1.2 Drop

A drop modifier is used to specify that a particular attribute or component thereof which may appear to be the same in both the source and target entities actually is not, and so values of this attribute must not be propagated to new instances. This operation is provided for completeness; it is not clear how useful it is in practice, as the old and new versions of the attribute concerned will most likely not be of compatible types, and so will not be copied anyway.

2.2 The BUILD Command

The `BUILD` command has the following syntax:

```
build-decl = 'BUILD' target-entity-id
    'FROM' source-entity-id { ',' source-entity-id }
    [ 'WHERE' expression { ';' expression } ] ';'
    [ 'DERIVE' derive ';' { derive ';' } ] .
```

A `BUILD` command specifies that instances of the named target entity are to be built from tuples of source instances. Each tuple consists of one instance of each named source entity. In the presence of a `WHERE` clause, only tuples which do not violate any of the expressions in the `WHERE` clause are included. Note that, as in EXPRESS, a clause which produces an `UNKNOWN` result is not considered to be violated.

The derivations in a `BUILD` command specify how to compute the values of the various attributes of the target instances. No data is copied into the target instances by default, even if there are attributes available with the same name and compatible types. This is one respect in which the `BUILD` command differs from the `COPY` command.

Another difference has to do with instance identity. Recall that the `COPY` command maps source instances to target instances with the same identity, i.e., having the same physical file identifiers as the source instances. The `BUILD` command creates entirely new instances with new identifiers.

Example 4: A basic BUILD command

Let us revisit our earlier example. Suppose that the target schema contains an additional entity:

```
ENTITY couple;
   him : man;
   her : woman;
   compatibility : int;
END_ENTITY;
```

Now, we would like to build an instance of `couple` for every "compatible" `man-woman` pair. Supposing that a compatible pair is one whose `masculinity/femininity` scores differ by no more than two, we could write:

```
BUILD couple FROM man, woman
WHERE
ABS(man.masculinity - woman.femininity) <= 2;
DERIVE
him := man;
her := woman;
compatibility := ABS(man.masculinity -
    woman.femininity);
```

The `WHERE` clause will be evaluated against each possible `man-woman` pair, and for each pair which does not violate it, an instance of `couple` will be produced. Note that this example assumes that there are `COPY` commands in place for `man` and `woman`: The derivations for `him` and `her` simply refer to the source entity instances themselves, and it is assumed that the same instances will be available in the target model.

Note that drop modifiers are not meaningful for a `BUILD` command, since no attribute values are copied into the new instances by default. Also note that `SELF` is not meaningful in a derivation within a `BUILD` command, as there is typically more than one source entity.

Another use of the `BUILD` command is to simulate a conditional `COPY`. This usage is actually a kludge dictated by a limitation of the language, but expresses a useful operation nonetheless. This usage is basically equivalent to a `COPY` command with a `WHERE` clause, a construct which ought to appear in the language in the future, except that the latter would preserve instance identity, while the `BUILD` does not. The problem this construct addresses is the problem of mapping instances of a particular source entity into instances of different target entities, according to some criterion. This can be expressed by writing several `BUILD` commands, each building instances of one of the target entities from the source entity, and having as its `WHERE` clause the relevant portion of the selection criterion.

Example 5: BUILD as a conditional COPY

Suppose our original people_schema did not have the entities man and woman, but rather represented gender as an attribute. We might wish to write a new schema which does have these separate subtypes:

Source Schema	Target Schema

SCHEMA people_schema;
 	
TYPE gender_type = ENUMERATION
    OF (male, female);
 	
ENTITY person;
   name : STRING;
   age : INTEGER;
   gender : gender_type;
   inity : INTEGER;
END_ENTITY;
 	
END_SCHEMA; -- people_schema
	SCHEMA people_schema;
	
	ENTITY person
	SUPERTYPE OF (ONEOF(man,
	                    woman));
	   age : REAL;
	   name : STRING;
	END_ENTITY;
	
	ENTITY man
	SUBTYPE OF (person);
	   masculinity : INTEGER;
	END_ENTITY;
	
	ENTITY woman
	SUBTYPE OF (person);
	   femininity : INTEGER;
	END_ENTITY;
	
	END_SCHEMA; -- people_schema

Now, we want to transform person instances according to the value of their gender attribute. We can do this by writing:

```
BUILD man FROM person
WHERE
person.gender = male;
DERIVE
age := person.age;
name := person.name;
masculinity := person.inity;

BUILD woman FROM person
WHERE
person.gender = female;
DERIVE
age := person.age;
name := person.name;
femininity := person.inity;
```

It quickly becomes evident why a conditional `COPY` command would be useful: `BUILD` does not copy any attribute values by default, nor does it provide for inheritance of other `BUILD` or `COPY` commands which might address some of the inherited attributes. Nonetheless, this example shows that the desired operation can be performed in TCSL in its current form.

---

3 Invoking Transformr

Transformr runs in a Unix™ environment, and is invoked by the command

```
% transformr   -d <difference spec>
               -e <source schema>
               -t <target schema>
               {-s <step file> | -o <output file>}
```

The `-d` option is used to specify the TCSL file specifying the mapping to be used. The `-e` option specifies the source EXPRESS schema for the mapping, and `-t` the target schema. The `-s` and `-o` options must be specified in equal numbers. Each `-s` specifies a STEP exchange file to be transformed, and the corresponding `-o`, the output file to be produced from this transformation.

Example 6: Typical *Transformr* invocation

```
transformr     -d geom_diffs.xform
               -e old_geometry.exp
               -t new_geometry.exp
               -s part1.stp -o part1.new
```

`part1.stp` must be a STEP exchange file conforming to the EXPRESS information model in `old_geometry.exp`. The product it defines will be transformed according to the specification in `geom_diffs.xform` into a product conforming to the EXPRESS information model in `new_geometry.exp`, which will then be written in exchange file format to `part1.new`.

---

4 Basic Theory of Operation

This section briefly describes *Transformr*'s theory of operation, in hopes of illuminating various behavioral idiosyncracies which may be encountered. It may be skipped without seriously hampering your ability to use *Transformr*.

There are two primary data structures used to represent the TCSL specification within *Transformr*. The first, `Correspondence`, is used to represent `COPY` commands, while `Construction` is used to represent `BUILD` commands. The semantics of these data structures very closely parallel those of the corresponding TCSL commands. All of the `Correspondences` and `Constructions` are collected together in a single `Mapping`.

To transform a particular model, *Transformr* first examines each instance in turn, checking for an applicable `Correspondence` and applying it if one is found. After all of the instances have been examined and transformed as necessary, *Transformr* next performs each `Construction`. To do this, the `WHERE` clause is evaluated for each candidate tuple in turn; for each tuple which does not violate the `WHERE` clause, a new instance is created and populated.

All derivations, whether for `BUILD` or `COPY` commands, are evaluated in the universe of the source model. The resulting values are then coerced into the target universe. As suggested above, the rules for type coercibility are similar to the rules for assignment compatibility in EXPRESS.

All numeric types (`INTEGER`, `NUMBER`, and `REAL`) are inter-coercible. Real numbers are truncated when they are coerced into integers.

An enumeration value in the source universe can only be coerced into an enumeration value of the same name in the target universe. The coercion process is done symbolically, so that reordering of enumeration values in the type definition is automatically handled. A value which does not appear in the target model will disappear in the coercion process, leaving the attribute in question with a missing value.

An aggregate value can only be coerced into an aggregate of the same class (array, bag, list, or set), and only if its base type is coercible into the base type of the target. Multi-dimensional aggregates currently are not handled.

When coercing a reference to an entity instance into the target universe, *Transformr* verifies that the instance has indeed been preserved in the target model by a `COPY` command; if it has not, the reference is deleted and the attribute left missing) and a warning message produced.

5 Some Known Limitations

This section highlights several known limitations of *Transformr*. These limitations will be addressed by future work.

Being based on the NIST PDES Toolkit [Clark90], *Transformr* inherits the limitations of this foundation. Notable among these for users of *Transformr* are:

- The Toolkit's STEP exchange file parser, STEPparse, currently does not allow forward references to entity instances.
- The Toolkit's EXPRESS parser, Fed-X, currently does not allow a particular enumeration value to appear in more than one enumeration type.

Both of these limitations will be removed from the Toolkit, and so from *Transformr*, in the near future.

Transformr does not implement all of TCSL as described. Also, the design and implementation of TCSL itself has been a learning experience in the requirements for such a language, and TCSL is not entirely adequate to the task. For the purposes of this discussion, these two classes of limitations are combined.

- TCSL lifts its expression syntax from EXPRESS. However, several types of expressions are currently unimplemented, though some are accepted by the *Transformr* parser. Unimplemented expressions include: aggregate constructors, aggregate operations (including indexing), entity instance constructors, group qualifiers, references to derived attributes, and most function invocations (only built-in arithmetic functions of a single argument are currently implemented).
- There is no way to define new functions in TCSL (of course, they couldn't be invoked even if they could be defined).
- There is no way of converting an aggregate value from one class (array, bag, list, or set) into another, short of writing a function, which currently cannot be done.
- As mentioned above, it would be immensely useful to be able to attach a `WHERE` clause to the `COPY` command.
- There is no way of referring to the instances constructed by a `BUILD` command, e.g. to insert them as attribute values into another instance. This problem could be alleviated by making the `COPY` command more powerful, e.g. adding conditional `COPY`, which would reduce the language's reliance on the `BUILD` command. The remaining usages of the `BUILD` command would still require that this problem be addressed.
- Some form of conditional expression or conditional assignment is needed within the derivation syntax.

6 Summary

TCSL in its current form is able to express many of the kinds of transformations which tend to be made to STEP information models. The prototype *Transformr* implementation has shown that this approach is an appropriate way of addressing the problem of keeping STEP instance data synchronized with changing EXPRESS information models.

The prototyping work has provided valuable insight into limitations of the approach as a whole and of the TCSL language in particular. There are two primary areas in which further work needs to be done: the TCSL implementation in *Transformr* (this basically means the expression evaluator) needs to be completed; and TCSL itself needs to be extended, in particular to provide for conditional `COPY`s and for referencing of the results of `BUILD`s. Future work on *Transformr* will address these limitations of the current prototype.

A References

[Clark90] Clark, S. N., An Introduction to the NIST PDES Toolkit, NISTIR 4336, National Institute of Standards and Technology, Gaithersburg, MD, May 1990.

[ISO11] Spiby, P., ed., ISO 10303 Industrial Automation Systems -- Product Data Representation and Exchange -- Part 11: Description Methods: The EXPRESS Language Reference Manual, Committee Draft N14, ISO TC184/SC4, April 29, 1991.

[ISO21] Altemueller, J., The STEP File Structure, ISO TC184/SC4/WG1 Document N279, September, 1988.

[Wirth] Wirth, N., in a letter in Communications of the ACM, 20:11, pp. 822-823, November, 1977.

NIST-114A
(REV. 3-90)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NISTIR 4944

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
OCTOBER 1992

4. TITLE AND SUBTITLE

Transformr: A Prototype STEP Exchange File Migration Tool

5. AUTHOR(S)

Stephen N. Clark

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)

OASD/CALS Evaluation and Integration Office
Department of Defense
Pentagon, Room 2B322
Washington, DC 20301-8000

10. SUPPLEMENTARY NOTES

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

This paper is a User's Guide to the Transformr STEP exchange file translation tool. Transformr transforms STEP exchange files between successive versions of their underlying EXPRESS information models. The Transformr Correspondence Specification Language (TCSL), which is used to establish the mapping between these schema versions, is described. The usage of Transformr is also described.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

data migration; EXPRESS; PDES; schema evolution; schema versioning; STEP; STEP exchange file; Transformr

13. AVAILABILITY

- [X] UNLIMITED
- [ ] FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).
- [ ] ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.
- [X] ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.

14. NUMBER OF PRINTED PAGES

16

15. PRICE

A02

ELECTRONIC FORM