Express Engine User Manual

v4.3.4

Express Engine Team

Table of Contents

1 Introduction

Express Engine is a software tool to help with the development and use of EXPRESS data models and STEP data populations. This document assumes the reader is familiar with EXPRESS (ISO 10303-11) and with the STEP physical file format (ISO 10303-21). The EXPRESS parser used in this tool conforms to the second edition of Part 11 (ISO 10303-11:2004). The physical file parser conforms to ISO 10303-21:2002. A file containing Part 11 definitions will be referred to as an EXPRESS File. A file containing Part 21 instances will be referred to as a STEP File or a data population. An EXPRESS file contains one or more schema declarations; schema is defined in Part 11.

2 Installation

2.1 Downloading

Using your favorite web browser, go to

http://exp-engine.SourceForge.net/

and select the download option for your particular platform. The archive that you download will contain the executable for the platform that you chose and the User Manual for the Application.

2.2 Windows Install

Copy the executable into a directory that is on the %PATH% enviornment variable.

2.3 Mac Install

To install it for a specific user, just put it into a directory on that user's PATH (`echo $PATH`). A common choice is `~/bin/`.

To install it so that everyone using the machine can get to it, put it into the `/usr/local/bin/` directory since most users have that directory on thier path.

2.4 Linux Install

To install it for a specific user, just put it into a directory on that user's PATH (`echo $PATH`). A common choice is `~/bin/`.

To install it so that everyone using the machine can get to it, put it into the `/user/local/bin/` directory since most users have that directory on their path.

3 Operating Modes

This version will only operate on the command line. Eventually, we will re-establish the graphical interface and add a server interface.

3.1 Non-Interactive

This mode is invoked from the command line, runs, provides output, and/or writes files, and exits.

3.2 Server

Not available.

This mode starts up, listens on the specified network port and/or the Unix domain port (file). It can be configured to preload a set of EXPRESS Files.

3.3 Interactive

Not available.

This mode starts with a window where the user can load EXPRESS Schema files and STEP Physical files.

4 Overview

Express Engine is sort of a Swiss Army knife for EXPRESS[1] and STEP[2]. It can perform multiple functions. Following is the current list.

4.1 Express Engine Functions

Function Description

–check Check EXPRESS Schema.

–compare Compare two EXPRESS Schemata.

–concat_schema
Create a file with the specified schema and all of the interfaced schemata such that no -stepmod is needed when using this file.

–dot Generate a Dot Graph file that can be processed by some other software to generate the diagram.

–flat Read an EXPRESS Schema in and write the Schema out with the ENTITY declarations flattened and the EXTENSIBLE SELECTs and EXTENSIBLE ENUMERATIONs flattened.

–lexp Write a lispified version of the EXPRESS Schema to a file.

–list List the contents of an EXPRESS Schema.

–pretty Read an EXPRESS schema file and organize it into a specific form according to defined layout rules.

–validate Read a STEP data population and its associated EXPRESS schema and write a report about how well the data population conforms to the EXPRESS schema.

–xref Reads a STEP data population and its associated EXPRESS schema and write a report about which instances reference which other instances.

4.1.1 Check EXPRESS File (–check)

Initial implementation. Not complete

```
eengine --check -schema <schema.exp>
```

Only one value is supported for -schema.

<*schema.exp*>
The EXPRESS Schema file.

This function reads an EXPRESS File, sorts it alphabetically, then writes out a report about any declarations that are either not used, not defined, or used inconsistently with the definition.

---

[1] ISO 10303-11

[2] ISO 10303-21

4.1.2 Compare EXPRESS Files (–compare)

```
Call with one stepmod
eengine --compare
   -mode <mode>
   -stepmod <stepmod/path>
   -stepmod_vcs <vcs_mode>
   -reference_schema <reference.exp>
   -trial_schema <trial.exp>
   --xml-output


Call with two stepmod
eengine --compare
   -mode <mode>
   -reference_schema <reference.exp>
   -reference_stepmod <stepmod/path>
   -reference_stepmod_vcs <vcs_mode>
   -trial_schema <trial.exp>
   -trial_stepmod <stepmod/path>
   -trial_stepmod_vcs <vcs_mode>
   --xml-output
```

This function compares two EXPRESS schemata. It is most useful for comparing two schemata that are different versions of the same schema. In this case, the output produced by the –xml-output option can be used in the new ISO 10303 standards document as a change log of what changed between the two schemata.

4.1.3 Concatenate EXPRESS Schemata (–concat_schema)

```
eengine --concat_schema \
   -schema <schema.exp> \
   -stepmod <stepmod/path> \
   -stepmod_vcs <vcs_mode>
```

This function generates a file which includes the schema specified in **<schema.exp>** and all schemata called out in any of the interface clauses of the included schemata.

Multiple values for -schema are supported. If more than one value is specified for -schema then each file is loaded and made part of the concatenated set to be written to the final file. Any additional schemata that are interfaced by the additional -schema values will also be added to the concatenated set.

4.1.4 Draw Dot Graph (–dot)

```
eengine --dot -graph <graph>
              -schema <schema.exp>
              -schema_name <name>
              -stepmod <dir>
              -prune <schema-list>
              -depth <depth>
              -iface <iface>
              -cluster <schema-list>
              -rf_color <color>
              -uf_color <color>
              -sub_color, <color>
              -super_color, <color>
              -attribute_color, <color>
              -select_color, <color>
              -leaf <schema-list>
              -path <declaration-list>
```

This function generates a graph based on the input specified. Options are provided to enable the user to tailor the graph for the specific presentation needs whilst using only one set of source schemata. The graph may be of schemata or may be of ENTITY and SELECT types.

Only one value for -schema is supported.

> **example:** Multiple schemata may compose a STEP AP. There is not a one-to-one match between the schemata and the engineering or manufacturing domains supported by the AP. That set of schemata may be used as source for multiple graphs, each of which is focused on a different sub-domain of the AP.
>
> **example:** A path of ENTITY and SELECT types may illustrate a particular initial mapping specification reference path (i.e., omitting mapping constraints), where the types extend over several schemata.

4.1.5 Write Flattened EXPRESS Schema (–flat)

```
eengine --flat
   -mode <mode>
   -schema <schema.exp>
   -stepmod <stepmod/path>
   -stepmod_vcs <vcs_mode>
```

schema.exp
: The EXPRESS schema definition file.

This function reads an EXPRESS Schema file and then writes it out to a new file that has the same name as the original file, but has the extension “flat” instead of “exp”.

Only one value for -schema is supported.

ENTITY objects will show their inherited attributes as comments.

EXTENSIBLE SELECT objects will show their base SELECT values as comments.

EXTENSIBLE ENUMERATIONS objects show their base ENUMERATION values as comments.

4.1.6 Lisp EXPRESS FILE (–lexp)

Initial implementation. Not complete.

```
eengine --lexp -schema <schema.exp>
```

schema.exp
: The EXPRESS Schema definition file.

  Only One value for -schema is supported.

This function reads an EXPRESS File and writes the Schema out to a file as a Lispified EXPRESS schema. The idea is that this lispified form should be faster to read into Express Engine since it is a Lisp based application.

4.1.7 List Schema Contents (–list)

```
eengine --list -schema <schema.exp>
```

schema.exp
: The EXPRESS Schema definition file.

  Only one value for -schema is supported.

This function reads an EXPRESS File and writes out a list of all of the declarations found in the file. This list includes the type of the declaration and the name of the declaration.

```
TYPE     schema1.foo;
ENTITY   schema1.bar;
FUNCTION schema1.baz;
   .
   .
   .
```

4.1.8 Pretty Printer for EXPRESS Schema (–pretty)

```
eengine --pretty \
   -mode <mode> \
   -schema <schema.exp> \
   -stepmod <stepmod/path> \
   -stepmod_vcs <vcs_mode>
```

schema.exp
: The schema that is to be Pretty Printed. The output will we written to a file that has "-pretty" appended to the name. This means that if the file 'schema.exp' is read in, the output will go to 'schema-pretty.exp'.

  Only one value for -schema is supported.

stepmod/path
: The root directory for stepmod for the case of shortform processing.

mode **arm_shortform**, **mim_shortform**, **arm_longform** and **mim_longform**.

This function reads the EXPRESS File specified by <*schema.exp*>, sorts the contents into a specific order, and writes out a new EXPRESS file in the format specified in the *SC4 Supplementary Directives*.

4.1.9 Test Compare (–test_compare)

In Development

```
eengine --test_compare \
        -trial_stepmod </path/to/trl-stepmod> \
        -reference_stepmod </path/to/ref-stepmod> \
        -cr_name <name_of_cr> \
        -cr_pub_dir <dir/to/published/cr>
```

<*/path/to/trl-stepmod*>
: The path to the trial STEPMod.

<*/path/to/ref-stepmod*>
: The path to the Reference STEPMod.

<*name_of_cr*>
: This is either an integer or a string.

<*dir/to/published/cr*>
: The path to the Published Change Request.

This function does a regression test on the –compare function using the contents of STEPMod and a specified Change Request.

4.1.10 Validate STEP File (–validate)

Not fully functional.

```
eengine --validate -schema <schema.exp> -population <population.stp>
```

schema.exp
: The EXPRESS Schema definition file.

  Only one value for -schema is supported.

population.stp
: The STEP population file.

This function reads an EXPRESS File, reads a STEP File, runs all of the WHERE clauses and RULE's, then writes out a report about what failed.

4.1.11 Cross Reference Instance Data (–xref)

```
eengine --xref -schema <schema.exp> -population <data.step>
```

schema.exp
: The EXPRESS Schema definition file.

  Only one value for -schema is supported.

data.step The STEP data population file.

This function reads an EXPRESS File, reads a STEP File that conforms to the EXPRESS file, then writes out a description of each instance in the file, which other instances reference it, and which other instances it references.

5 Check EXPRESS Schema

```
eengine --check -schema <schema>
```

WARNING!! the Check Schema function is not complete. While it can be useful to run Check Schema, passing Check Schema does not mean that your schema is clean. It may have a problem that Check Schema does not yet catch.

> **NOTE:** Running Check Schema on a shortform schema can be very time consuming as it looks through multiple schemata to identify TYPE and ENTITY references.

This function checks the content of the EXPRESS Schema to make sure that it adheres to the specifications laid out in the *SC4 Supplementary Directives* (SC4SD) and the *ISO 10303-11 EXPRESS Language manual* (ISO 10303-11:2004). It cannot check any of the textual layout specifications, but it can check naming conventions and other content related specifications. As it works, it will write messages out to the console telling what it is doing and what it is finding.

When it completes the check, it writes the name of the input file, the date and time the input file was written, the comments from the header of the file containing the schema, the version string that immediately follows the schema name if present, and all the messages about what it found to a file.

If the schema that was checked was loaded from a file named `my_schema.exp` then the results will be written to a file named `my_schema-check.txt`.

5.1 Status of Check Function

This section contains a list of what checks are working and which ones are known not to work. This is a preliminary list. More effort will be expended later to make a more exhaustive list.

5.1.1 Working Checks

- that WHERE clauses are named 'WR0' where '0' is some number.
- that UNIQUE clauses are named 'UR0' where '0' is some number.
- that INVERSE attributes name a valid ENTITY
- that INVERSE attributes invert an attribute of an ENTITY that exists
- the inverted attribute exists on the specified ENTITY and is an explicit attribute and not DERIVE or INVERSE.
- that fully qualified attribute reference strings contain a valid SCHEMA name, ENTITY name, and ATTRIBUTE name.
- that no attributes have the same name as any of the accessible FUNCTION declarations.
- that the SUBTYPE OF elements name accessible ENTITY declarations
- that the SUPERTYPE OF elements reference valid ENTITY declarations

- that no two interface specifications interface the same object or objects

  Note: The reports of improper things found should all start with an identifier for the document and the section within that document relevant to the issue.

5.1.2 Non-Working Checks

- Check for EXPRESS syntax pecularities.

  Check for cases where ENTITY declarations have the supertype and subtype clauses switched.
- Check for references to undefined TYPE, ENTITY, FUNCTION, or PROCEDURE declarations.
- Check for unreferenced TYPE, FUNCTION, and PROCEDURE declarations.
- Check for attribute references that are not qualified but should be.
- Check for attribute references that are qualified but don't need to be.

5.2 Example Output from –check

This output is from running **eengine –check** on the AP210 MIM_LF schema.

```
Report for EXPRESS Schema Check

  Schema: AP210_MIM_LF
  File:   /home/craig/Desktop/exp-step/ap210_mim_lf.exp
  Timestamp: 2016-3-31  1:23:32 GMT

Schema ap210_mim_lf
(*
   $Id: mim_lf.exp,v 1.61 2014/03/24 17:04:25 thomasrthurman Exp $
   ISO TC184/SC4/WG3 8232 - ISO/TS 10303-410 AP210 electronic assembly
       interconnect and packaging design - EXPRESS MIM Long form
   Supersedes ISO TC184/SC4/WG3 N2601
    *)
(* ================================================================= *)
(* Long form schema generated by The EXPRESS Data Manager
   compiler version 9.8.3B 20121030 *)
(* Fri Feb 07 11:23:49 2014 *)
(* The schema is converted from ISO10303 P11 2003 to 1994            *)
(* ================================================================= *)
(* This file was generated by the EXPRESS Pretty Printer exppp,
   part of STEPcode (formerly NIST's SCL). exppp version:
   git commit id: g0a46b86, build timestamp 26 Feb 2014 19:15        *)
(* patched for bug 4665, 4992, 5004 *)

Checking SCHEMA ap210_mim_lf

Checking TYPE maths_value
```

```
  SC4SD 6.3.2: WHERE rule label 'CONSTANCY' doesn't match pattern 'WR0'
End Checking TYPE maths_value;

Checking TYPE nonnegative_integer
  SC4SD 6.3.2: WHERE rule label 'NONNEGATIVITY' doesn't match
               pattern 'WR0'
End Checking TYPE nonnegative_integer;

Checking TYPE one_or_two
  SC4SD 6.3.2: WHERE rule label 'IN_RANGE' doesn't match pattern 'WR0'
End Checking TYPE one_or_two;

Checking TYPE positive_integer
  SC4SD 6.3.2: WHERE rule label 'POSITIVITY' doesn't match pattern 'WR0'
End Checking TYPE positive_integer;

Checking TYPE zero_or_one
  SC4SD 6.3.2: WHERE rule label 'IN_RANGE' doesn't match pattern 'WR0'
End Checking TYPE zero_or_one;

Checking ENTITY assembly_component_usage_substitute
  SC4SD 6.2.2.6: The attribute 'substitute' of ENTITY
                 'assembly_component_usage_substitute' has the same
                 name as a FUNCTION
End Checking ENTITY assembly_component_usage_substitute;

Checking ENTITY characterized_item_within_representation
  SC4SD 6.3.2: UNIQUE rule label 'WR1' doesn't match the pattern 'UR0'
End Checking ENTITY characterized_item_within_representation;
End Checking SCHEMA ap210_mim_lf;
```

NOTE: Lines have been manually folded and the schema name has been shortened for readability of the example.

Note that the output includes the file write date and header comments from the input file.

5.3 Checks that may be added in the Future

Same as non-working checks above.

- Check for a number adjacent to a logical operator

  ```
  "1 OR" is a common typographical error when
  "...=1) OR" is intended.
  ```

- Check for "SELF."

6 Compare EXPRESS Schemata

```
Calling compare with one stepmod instance
eengine --compare \
        -stepmod <stepmod/path> \
        -stepmod_vcs <vcs_access> \
        -mode <mode> \
        -reference_schema <reference.exp> \
        -trial_schema <trial.exp> \
        --xml_output

Calling compare with two stepmod instances
eengine --compare \
        -reference_stepmod <stepmod/path> \
        -reference_stepmod_vcs <vcs_access> \
        -reference_schema <reference.exp> \
        -mode <mode> \
        -trial_stepmod <stepmod/path> \
        -trial_stepmod_vcs <vcs_access> \
        -trial_schema <trial.exp> \
        --xml_output
```

reference.exp
: The file containing the reference schema. Includes complete path to file.

trial.exp
: The schema definition file that we want to compare to the reference. Includes complete path to file.

mode
: One of **arm_longform**, **mim_longform**, **arm_shortform**, **mim_shortform**, **arm_concatenated**, or **mim_concatenated**.

stepmod/path
: This is the location of a stepmod directory.

schema_name
: If present, this is the name of the schema that is to be compared. This is most useful when one of the schema files is a **arm_concatenated** or **mim_concatenated** schema file. This is the name of the schema that will be used for the compare. Shortform and Longform files don't pose a problem since they only contain a single schema anyway. This option is available as a backup in case only concatenated files are available as one of the comparison data input sets, because the primary method is to directly use stepmod content.

vcs_access
: This is the type of access that should be used when getting VCS info. The possible values are exaplained in the following table:

  online Call the VCS executable to access the relevant server.

  offline Use info in the header of the file in question.

  off Don't access any VCS info.

–xml-output
: If present, this indicates that the XML file should be written.

This function compares a trial schema to a reference. As it works, it writes a log to standard output (the console). First it checks to make sure that the two files to be compared are present. If one of them is not present it will output a message saying which one is missing.

If either of the file is missing it will exit without doing anything.

When it finishes, it writes the file "output.txt" that contains all of the findings. If **–xml-output** is included it will write an XML file that can be used as the change record from the reference schema to the trial schema. If <mode> is **arm_longform** the "output.txt" is "output-arm.txt". If <mode> is **mim_longform** then "output.txt" is "output-mim.txt". If **–xml-output** is included then it writes "output-arm.xml" or "output-mim.xml" depending on the value of <mode>.[1] If <mode> is **arm_shortform** or **mim_shortform**, there are three options for the interfacing schemas:

• -stepmod, used where one stepmod is used for both versions,

• (-reference_stepmod,-trial_stepmod), used where each schema has its own instance of stepmod,

• concatenated files, used for shortform compare but two instances of stepmod are not used.

This function is primarily intended for the case where the two EXPRESS files being compared are different versions of the same schema. The reference is the older schema and the trial is the new schema that is being developed.[2]

As an example, let us compare the AM ISO/TS 10303-1026 Assembly 105 structure arm.exp file to a prior released version using two instances of stepmod, one for reference and one for trial locations.

• pre-condition:

• The local reference stepmod directory is `/usr/<user>/.../SMRLv6`.

• The local reference schema file is `/usr/<user>/.../SMRLv6/data/modules/assembly_structure/arm.exp`.

• The local trial stepmod directory is `/usr/<user>/.../stepmod`.

• The local trial schema file is `/usr/<user>/.../stepmod/data/modules/assembly_structure/arm.exp`.

• execute eengine command:

---

[1] If <mode> is **arm_longform** or **mim_longform** then each schema file shall be a fully self contained Long Form schema) and there will be no need of the *-stepmod* option as there will be no Interface clauses to resolve.

[2] In the case that one schema is an extension of a basis schema (e.g., ISO 10303-409 is an extension of ISO 10303-442), the compare function may provide useful information when the user first aligns the schema names in the two files by selecting one or the other schema name as the schema name in each file.

```
The reference stepmod has a root directory at -reference_stepmod.
The trial stepmod has a root directory at -trial_stepmod.
eengine -compare \
  -reference_stepmod /usr/<user>/.../SMRLv6 \
  -reference_schema \
   /usr/<user>/.../SMRLv6/data/modules/assembly_structure/arm.exp \
  -trial_stepmod /usr/<user>/.../stepmod \
  -trial_schema \
   /usr/<user>/.../stepmod/data/modules/assembly_structure/arm.exp \
  -mode arm_shortform
```

• post-condition: EXPRESS_arm_longform_comparison_results.txt file in the local directory.

As another example, let us compare the AM ISO/TS 10303-1026 Assembly structure arm.exp file to a prior released version eengine but use the concatenated file for the interfaced schemata of the reference schemata and use stepmod for the interfaced schemata of the trial schema.

• pre-condition: The local directory is `.../stepmod/data/modules/assembly_structure`. The file `. . . / modules / ap209_multidisciplinary_analysis_and_design / dvlp / concatenated_arm.exp` with rcs file revision value of 1.10 is the version we will use as a reference because that version is tagged with the latest published edition of the STEP module and resource library, SMRLv6.[1]. The file arm.exp is at the version we wish to use for the trial schema.[2]

• execute eengine command:

```
eengine -compare \
  -reference_schema concatenated_arm.exp \
  -trial_schema arm.exp \
  -mode arm_shortform \
  -stepmod <path to stepmod> \
  -schema_name Assembly_structure_arm
```

• post-condition: output-arm.txt and output.txt files in the local directory.[3]

[1] It may be necessary to execute a CVS command to restore the required file to the local repository.

[2] Typically this would be the latest Head version.

[3] We need -schema_name to disambiguate the schema in the concatenated_arm.exp schema file.

As another example, let us compare two arm.exp files that are in development (i.e., the -stepmod option is valid for both file versions.).

• pre-condition: The local directory is `.../stepmod/data/modules/assembly_structure`. The file arm.exp revision 1.33 is the version we will use as a reference.[1]. The file arm.exp is at the version we wish to use for the trial schema.[2]

• execute eengine command:

```
eengine -compare \
  -reference_schema arm_1.33.exp \
  -trial_schema arm.exp \
  -mode arm_shortform \
  -stepmod <path to stepmod>[3]
```

• post-condition: output-arm.txt and output.txt files in the local directory.

[1] It will be necessary to execute a CVS command to restore the required file to the local repository and then change the file name to e.g., arm_1.33.exp.

[2] Typically this would be the latest Head version.

[3] The -schema_name option is unnecessary because both files have the same schema name declaration.

7 Concatenate EXPRESS Schema

This function takes one or more shortform schemata and a stepmod directory and creates a concatenated file. The value assigned to -mode is either arm_shortform or mim_shortform.

Call with one schema

```
eengine --concat_schema \
         -schema <schema> \
         -mode <shortform> \
         -stepmod <stepmod/dir> \
         -stepmod_vcs online
```

In the case of a single schema, it reads enough of each schema to identify any interfaced schemata. It then repeats the process for each of the interfaced schemata until is has identified all the schemata that need to be concatenated. It then copies each schema line by line into the concatentated schema file.

It will also output and VCS information about each of the schemata that have been read. To control this you can provide the

-stepmod_vcs

option with a value of either **online**, **offline**, or **off**. If the value is **online** (the default) and you are on Windows, you will also need to specify the

-vcs_exe <path/to/cvs.exe>

option to specify where the `cvs.exe` is located. For more information, see the `-vcs_exe` option in Appendix A [Command Line Arguments], page 50.

Call with multiple schemata

```
eengine --concat_schema \
        -schema <schema1,schema2,schema3> \
        -mode <shortform> \
        -stepmod <stepmod/dir>
```

In the case of a list of schemata, it loads all of the interfaced schemata called out in each shortform schema in the list and each of the schemata that each schema in the list interfaces.

It then writes out a file containing each schema in the list and each of the schemata that were interfaced directly or indirectly by each schema in the list.

The header of the concatenated file will have a comment that contains the schema version string and VCS info for each of the inclcuded schemata.

Any CVS $Id ...$ records will be adjusted so that CVS will not think that it should modify them. This will probably be done by removing the dollar signs ($) at the beginning and end of the record and just enclose them in double quotes (").

This means that $Id: ...$ becomes "Id: ...". This way we keep the content of the CVS ID string, but prevent CVS from changing it again.

7.1 Example Output from –concat_schema

[Example to be added]

NOTE: Lines have been manually folded and the schema name has been shortened for readability of the example.

Note that the output includes the file write date and header comments from the input file.

8 Generate dot graph

```
eengine --dot
        -graph <graph>
        -depth <depth>
        -entity <entity_name sequence>
        -stepmod <stepmod/path>
        -schema <schema.exp>
        -schema_name <schema_name>
        -prune <schema_names>
        -leaf <schema_names>
        -cluster <cluster>
        -uf_color <color>
        -rf_color <color>
        -sub_color <color>
        -super_color <color>
        -attribute_color <color>
        -select_color <color>
```

This function generates a graph based on the input specified. Options are provided to enable the user to tailor the graph for the specific presentation needs whilst using only one set of source schemata. The graph may be of schemata or may be of a single path through one or more schemata based on identified ENTITY and SELECT types.

<*graph*> This indicates what type of graph will be produced.

interface Generates a graph of the USE FROM and REFERENCE FROM relationships between multiple shortform schemata.

The value of -graph is "interface".

This graph also uses the -schema -schema_name -stepmod -prune -leaf -cluster -rf_color -uf_color options.

path Generates a graph from one entity to another via a specified sequence of entity and select types. These elements must be referencially adjacent to each other since Express Engine cannot deduce long arbitrary paths through the schemata.

The value of -graph is "path".

This graph uses the -schema -schema_name -stepmod -entity -sub_color -super_color -attribute_color -select_color options.

The parameter -entity is required.

<*depth*> An integer that identifies how deep the graph can get.

<*entity_name sequence*>
This is a sequence of entity_names each separated by a comma with no spaces (i.e. name1,name2,name3).

<*schema.exp*>
This is the name of the file that contains the starting schema for the interface option or the starting ENTITY TYPE for the path option.

<*schema_name*>
: This will be the name of the schema to be used as the starting point. This is needed if <*schema.exp*> is a concatenated schema file, otherwise not.

<*schema_names*>
: This is a set of schema names each separated by a comma with no spaces.

<*cluster*>
: **NOTE::** This is the only option that may be specified multiple times.

  This is a set of schema names where the first schema it the one that should be shown in the diagram. Any references to or from the other schemata will go to or from the first named schema. This basically causes the first schema to replace the other schemata in the diagram.

<*color*>
: This is the name of the color to be used for: uf_color, rf_color, sub_color, super_color, attribute_color, and select_color. Allowed values are black, red, blue, green, yellow, cyan, magenta, and orange.

8.1 Generate dot Interface Graph

Generates a graph of the interface relationship between schemata. Each node represents a SCHEMA. Each edge represents an EXPRESS **interface specification**. The edges representing the interfaces are drawn with a normal arrowhead when the interface has no resources. The edge representing the interface is draw with a curve arrowhead when the interface does have resources.

Interface mode is enabled by the "interface" value for the -graph parameter.

Interface may use the following additional data:

- -prune, -cluster, -leaf <schema-list>;
- -depth <depth>;
- -iface <iface>;
- -rf_color, -uf_color <color>;
- <schema-list>;
- <depth>;
- <iface>.

Reference the individual data descriptions for applicability.

<*schema-list*> is used by -prune, -leaf, and -cluster options. This is a comma separated list of schema names that will be represented by nodes in the graph. **DO NOT INCLUDE ANY SPACES**

For the -prune option, <schema-list> is the list of SCHEMA nodes that will be omitted from the graph. No edges pointing to or from the prune nodes will be drawn. More than one -prune option may be provided for a session of engine.

> You may wish to prune the **support_resources_schema** and **management_resources_schema** to reduce clutter due to edges entering and leaving those nodes.

For the -leaf option, the list of SCHEMA nodes that are to be considered leaf nodes in the graph. No edges pointing out of the leaf nodes will be drawn. Leaf nodes are at the bottom of the graph when vertical organization is selected. Leaf nodes are visually highlighted by their usual position on the bottom level of the diagram. Some layout engines may not place all leaf nodes on the lowest level. No more than one -leaf option shall be provided for a session of engine.

> You may wish to make the **assembly_shape_mim** a leaf node to hide the complex structure below it.

For the -cluster option, the list of SCHEMA nodes that are to be subsumed into one node in the graph. Usually the SCHEMAta are closely related in the context of the intended presentation. The first node in the list is retained to represent the cluster. Each edge pointing into the list is redeclared to point to the node representing the cluster. Each edge pointing out of the list is redeclared to point from the node representing the cluster. Self-referential edges are omitted from the graph. More than one -cluster option may be provided for a session of engine.

> You may wish to cluster nodes **Value_with_unit**, **Value_with_unit_extension**, **measure_schema**, **qualified_measure_schema**, and **representation_schema** to elide the complexity of the relationships among those nodes.

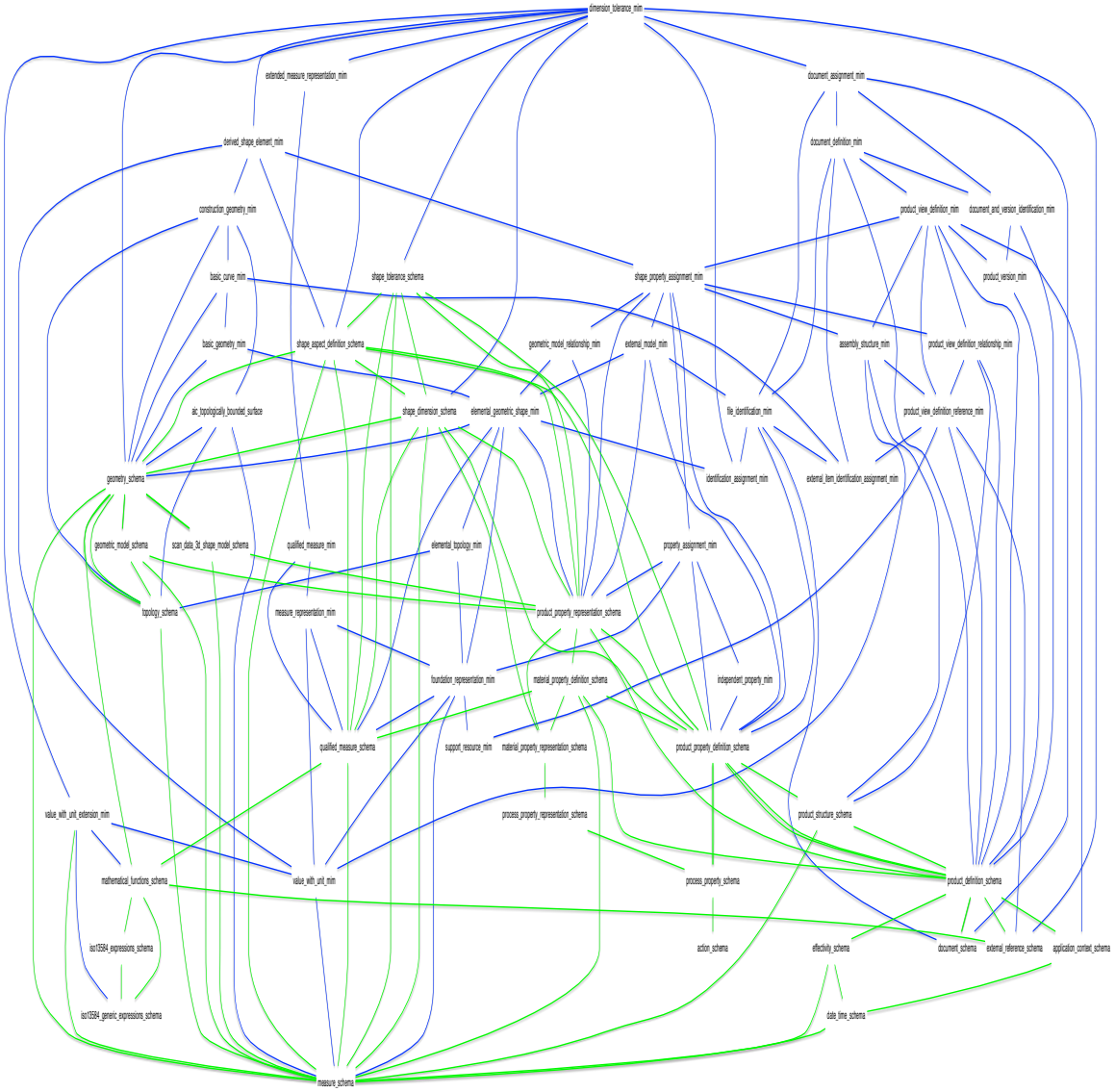
<*depth*> is the depth of the diagram. The root schema is depth 0 so a depth of 1 will draw a graph with the root schema and only the schemata that it interfaces. Depth 2 will draw a graph that contains the root schema, all of the schemata that it interfaces, and all of the schemata the those schemata interface.

Note that when the Dot graphical auto layout tool lays out the diagram it may create a graph with more levels than requested because auto layout is optimizing the overall layout.

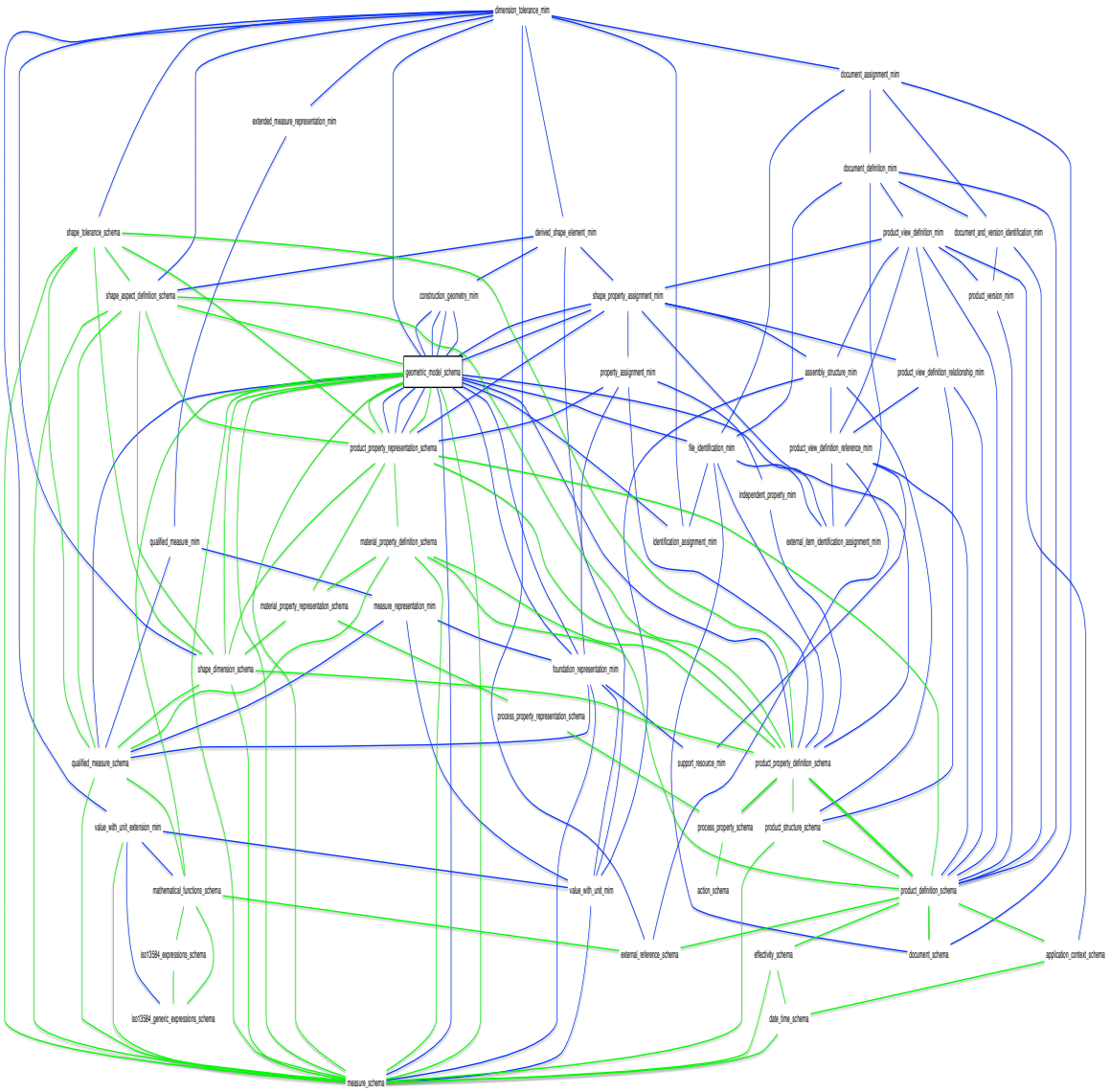
<*iface*> is the type of interface(s) that should be drawn in the graph.

The value can be **use**, **ref**, or **both**.

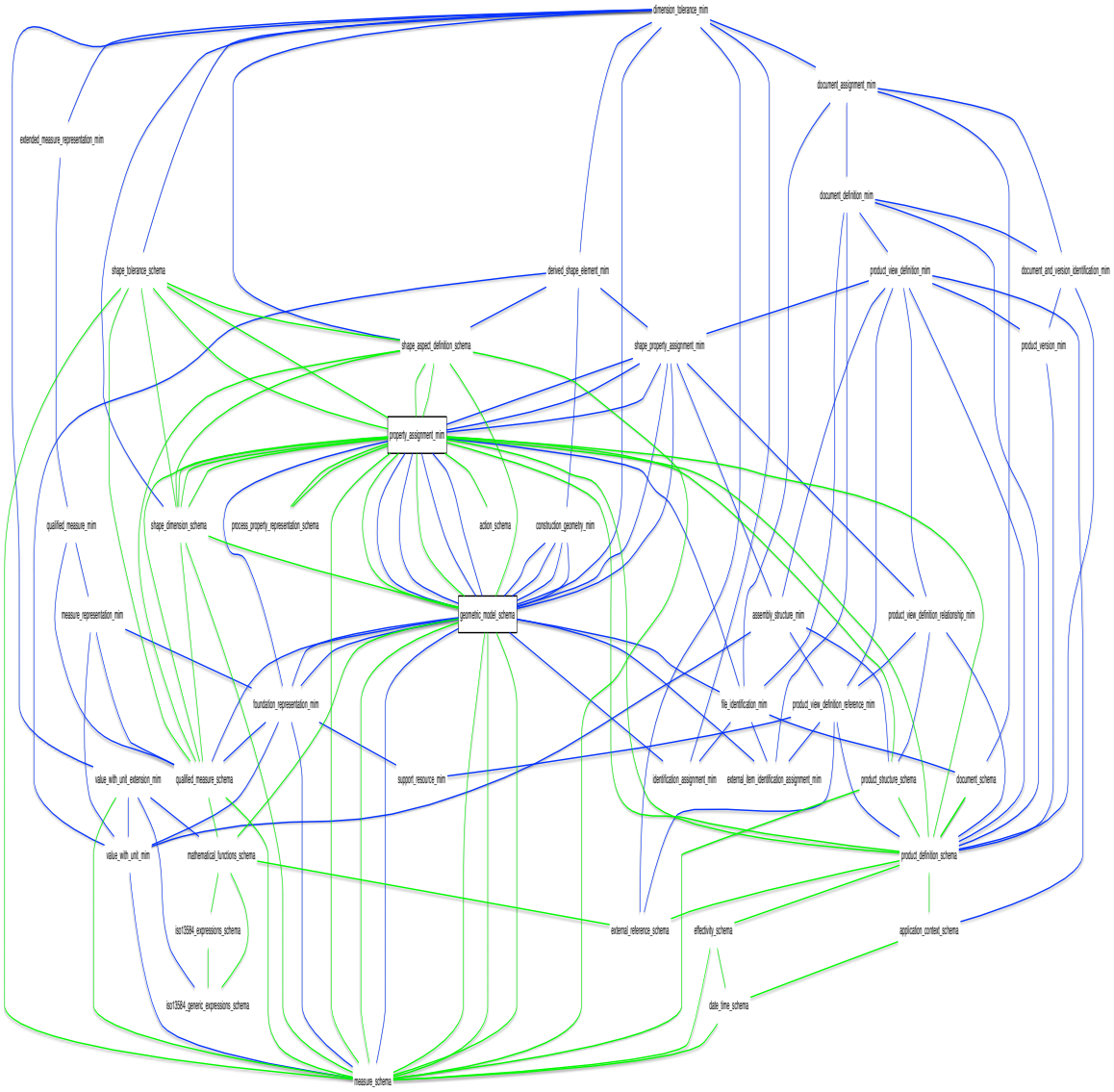
If this option is not specified then it is the same as specifying **both**.



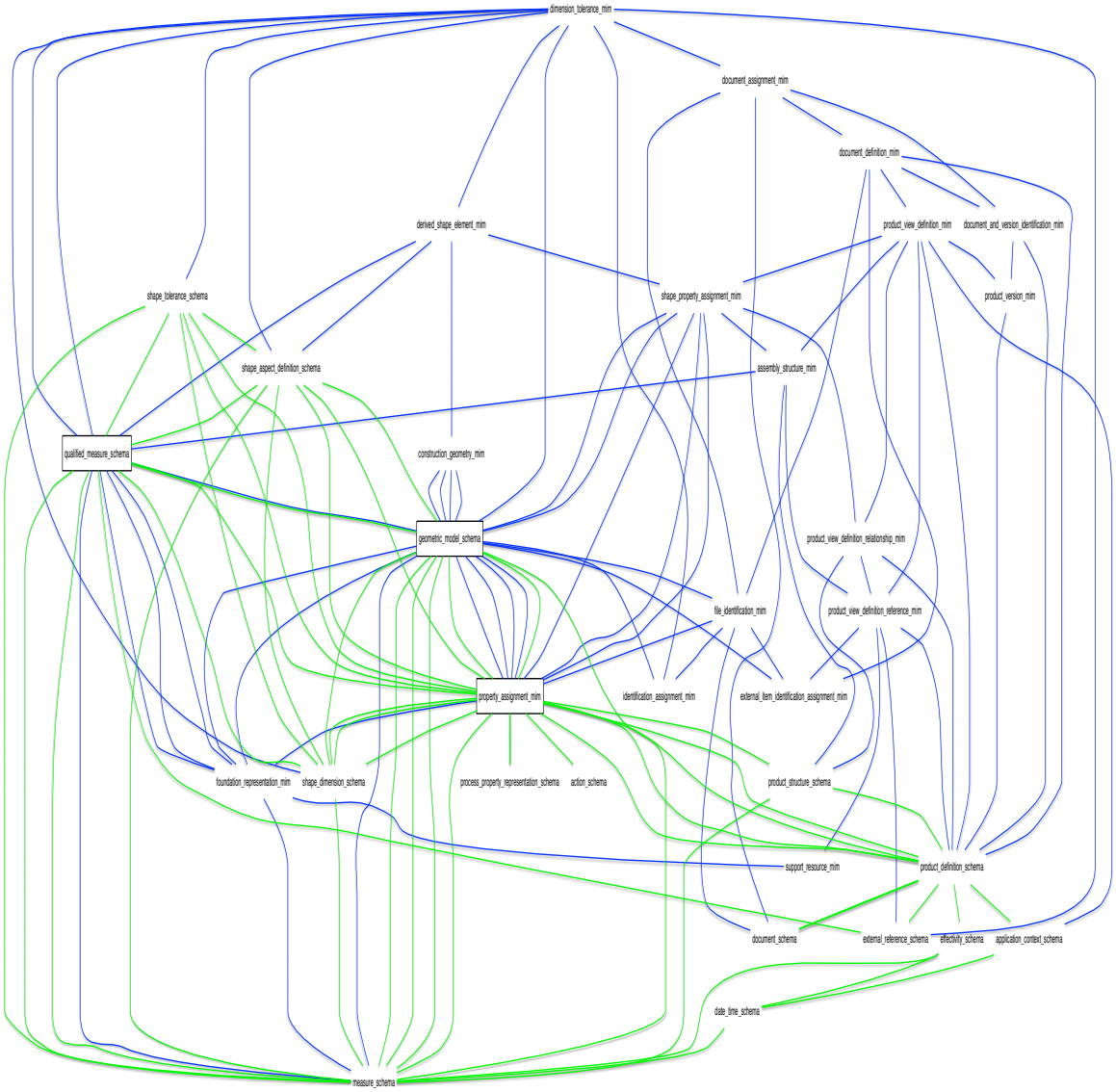
An initial graph. USE FROM edges are blue; REFERENCE FROM edges are green.



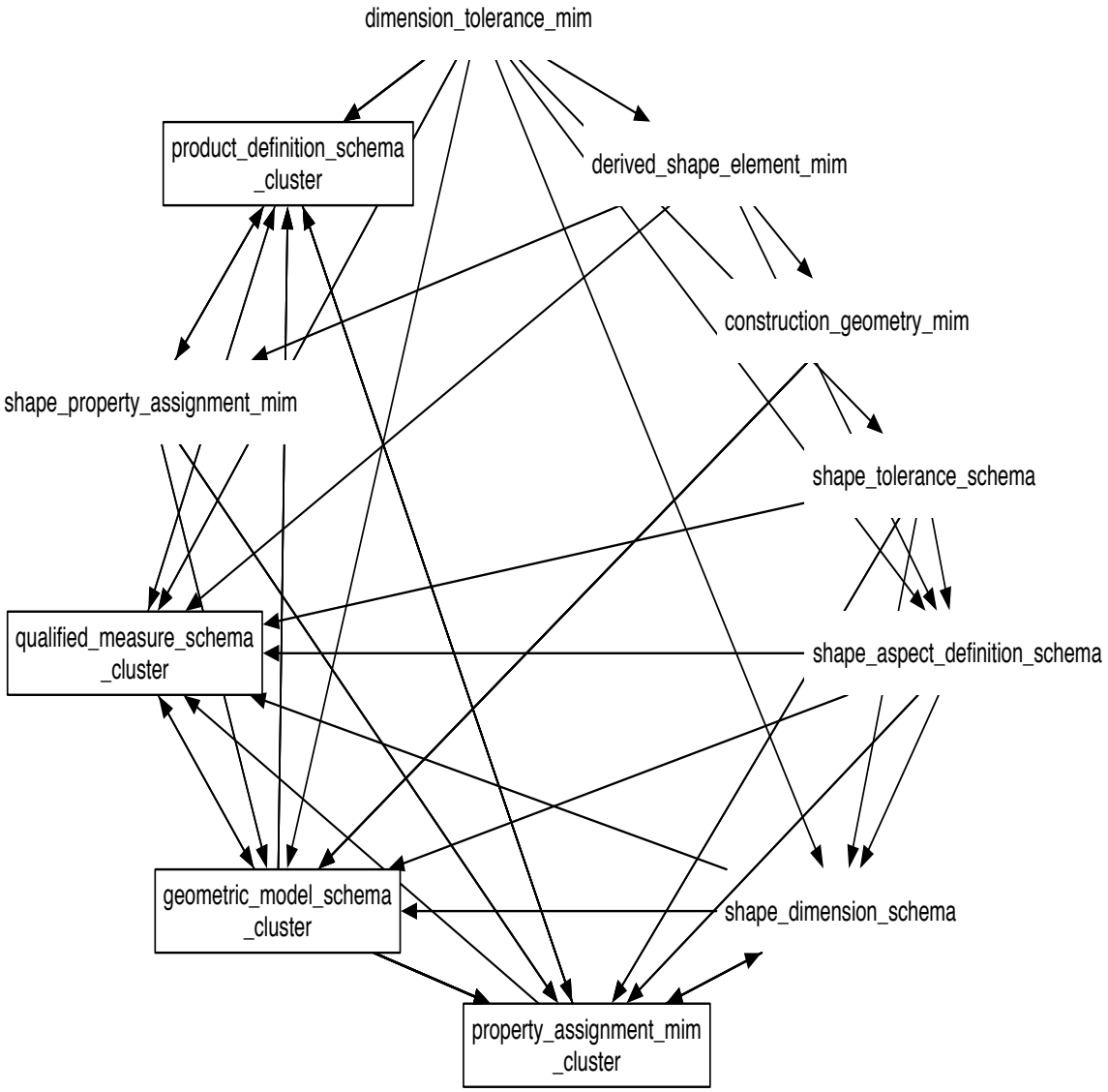
First tailoring of the graph for presentation focus. There is a cluster founded on the geometric_model_schema.



Second tailoring of the graph for presentation focus. A cluster was added founded on the property_assignment_mim schema.



Third tailoring of the graph for presentation focus. A cluster was added founded on the qualified_measure_schema.



Final tailoring of the graph for presentation focus. This graph had lines straightened, four clusters (rectangles with solid edges), pruning and leaf declarations.

NOTE: If desired, the layout engine can change the edge style of the rectangles but that was not done in this set of examples to illustrate the default behavior.

`The complete command for Eengine is this bash script:`

```
#!/bin/sh
eengine
--dot -graph interface
-schema /Users/tom/workspace_422/stepmod/data/modules/\
dimension_tolerance/mim.exp
```

```
-mode mim_shortform
-stepmod /Users/tom/workspace_422/stepmod
-prune foundation_representation_mim,\
measure_schema,\
design_product_data_management_mim,\
product_identification_mim,\
foundation_representation,\
support_resource_schema,\
support_resource_schema,\
basic_attribute_schema,\
management_resources_schema,\
representation_schema
 -cluster geometric_model_schema,\
geometry_schema,\
topology_schema,\
aic_topologically_bounded_surface,\
basic_curve_mim,\
basic_geometry_mim,\
elemental_geometric_shape_mim,\
elemental_topology_mim,\
geometric_model_relationship_mim,\
external_model_mim,\
scan_data_3d_shape_model_schema
 -cluster property_assignment_mim,\
process_property_representation_schema,\
action_schema,\
product_property_representation_schema,\
product_property_definition_schema,\
independent_property_mim,\
material_property_definition_schema,\
material_property_representation_schema,\
process_property_schema
 -cluster qualified_measure_schema,\
value_with_unit_extension_mim,\
extended_measure_representation_mim,\
value_with_unit_mim,\
qualified_measure_mim,\
measure_representation_mim,\
mathematical_functions_schema,\
iso13584_generic_expressions_schema,\
iso13584_expressions_schema
 -cluster product_definition_schema,\
external_reference_schema,\
identification_assignment_mim,\
document_assignment_mim,\
external_item_identification_assignment_mim,\
support_resource_mim,\
```

```
product_view_definition_relationship_mim,\
assembly_structure_mim,\
product_structure_schema,\
product_view_definition_reference_mim,\
product_view_definition_mim,\
application_context_schema,\
product_version_mim,\
product_identification_mim,\
file_identification_mim,\
document_schema,\
document_definition_mim,\
document_and_version_identification_mim,\
effectivity_schema,\
date_time_schema
```

8.2 Generate dot Path Graph

Generates a graph of the path specified by a series of ENTITY types.

Path mode is enabled by the "path" value for the -graph parameter.

The parameter -entity is required.

Path may use the following additional data:

- -sub_color <color>
- -super_color <color>
- -attribute_color <color>
- -select_color <color>

The nodes of the resulting graph will be made up of the following object types:

- ENTITY TYPE;
- SELECT TYPE;
- an extension to a SELECT TYPE.

Nodes that represent ENTITY and TYPE declarations from a single schema are enclosed by a rectangle which has the name of that schema as its label.

> **NOTE:** A single path is useful for educational and review purposes. The path option does not provide many options in order to retain simplicity and ease of use.

The label for an edge is the attribute name in the case of an ENTITY->ENTITY edge. The label is not provided in the case of a SELECT -> ENTITY edge or in the case of a SELECT -> SELECT edge. In the case that multiple attributes of an ENTITY specify the same target EXPRESS element, only one edge will be populated in the graph.

The label for an edge from a SUBTYPE to a SUPERTYPE is "SUPER".

The label for an edge from a SUPERTYPE to a SUBTYPE is "SUB".

The label for an edge from an extension SELECT TYPE to the EXTENSIBLE SELECT TYPE being extended is "based_on"

An ENTITY TYPE is represented by a rectangle.

A SELECT TYPE is represented by an oval.

A schema is represented by a rounded corner rectangle that encloses one or more nodes.

> **NOTE:** SELECT TYPES are rendered as ovals to differentiate them from ENTITY TYPES because some post-processor layout tools don't support dotted line types.
>
> **NOTE:** There may be cases where local (WHERE) RULES or (global) RULES constrain allowed path elements. For the initial implementation those constraints will be ignored.

<*entity_name sequence*>

> The <entity_name sequence> follows the -entity parameter on the command line and provides a comma separated list of EXPRESS declarations. The <entity_name sequence> is the list of EXPRESS declarations to be traversed to generate the sub graph. Only ENTITY and SELECT types are included for now, as enumerations are not deemed that much of interest. Although there may be multiple sub graphs due to SELECT TYPE trees, there must be only one starting ENTITY TYPE and one ending ENTITY TYPE.
>
> > **note:** Engine does not support parallel paths with this option.
>
> **DO NOT INCLUDE ANY SPACES**
>
> Each comma in the list represents a directed, structural, relationship specification between the element before the comma and the element after the comma. That is, there shall be a valid relationship in the relevant EXPRESS schemata between the two elements separated by the comma. The relationship may be a SUB/SUPER TYPE relationship, may be an attribute relationship, may be a (SELECT reference/SELECT element) relationship, or may be a collection of paths through a SELECT TYPE tree. The order of the elements in the graph is derived from the sequence of commas in the list.
>
> > **note:** The relationship will often traverse SCHEMA interface declarations.
>
> The list may consist of forward declarations and/or reverse declarations. A forward declaration specifies that the current ENTITY TYPE or SELECT TYPE references the element after the comma. A reverse declaration specifies that the current element is referenced by the ENTITY or SELECT TYPE after the comma.
>
> In the case that SELECT TYPES are not provided in the list, engine will provide a sub graph showing the possibly multiple paths through the SELECT TYPES implied by the relationship specified by the comma.
>
> > **note:** It is possible that more than one SELECT TYPE may exist between two ENTITY types that are related by a comma.
> >
> > **note:** It is possible that more than one path may exist between two ENTITY types that are related by a comma. This would be the case where multiple attributes of an ENTITY referenced the same (other) ENTITY. In that case, only one edge would be output by engine.

note: Engine ignores references to SELF in an ENTITY declaration.

note: It is possible that a SELECT extension may exist between two ENTITY types that are related by a comma.

note: In order to reduce the path generation algorithmic complexity, currently only SELECT TYPES may be omitted from the list.

example: For a case of a SUB/SUPER TYPE relationship and an attribute relationship, both edges will be emitted by engine.

note: It is quite often the case that a path traverses a SUB/SUPER relationship from one subtype to the supertype, then to another subtype.

note: Express engine calculates and displays alternate complete paths for the graph. This is useful for users who are not familiar with the SCHEMA structure, or who are not aware of recent additions to the schemata. It is also useful for users who do not know which schema contains the declaration for a particular ENTITY or SELECT TYPE.

In the following example, an incomplete list is given (i.e., SELECT TYPES are omitted); both forward and reverse declarations are included; SUB/SUPER relationships are included. The example illustrates the intermediate and final results of the path option.

note: The backslash character indicates there is no line break in the actual input data. There are spaces before the ENTITY names in the rendered presentation of the example in this document. There are no spaces in the actual input data list.

```
-entity product,\
product_definition_formation,\
product_definition,\
property_definition,\
property_definition_representation,\
representation,\
representation_item,\
measure_representation_item,\
measure_with_unit,\
length_measure_with_unit
```

The SELECT TYPES must be determined by engine so that the emitted graph is valid. The SELECT TYPES are determined by querying the EXPRESS schemata for valid paths from the ENTITY TYPE before the comma to the ENTITY TYPE after the comma. The function executed is similar to the EXPRESS USEDIN function except that the specific attribute is not specified in the list of entities, and USEDIN operates on data sets.

Specifically these SELECT TYPES are added by engine:

```
1-
represented_definition
(between property_definition_representation and
 property_definition)

2-
characterized_definition and
characterized_product_definition
(between property_definition and product_definition)
```

Engine had to traverse SUBTYPE relations between measure_representation_item and measure_with_unit and between measure_with_unit and length_measure_with_unit.

Engine had to traverse the following schemata in order to build the path:

```
product_definition_schema, \
product_property_definition_schema, \
product_property_representation_schema, \
representation_schema, \
qualified_measure_schema, \
measure_schema.
```

The resulting path (using the syntax of ISO 10303 mapping specification) constructed by engine prior to emitting a dot file is:

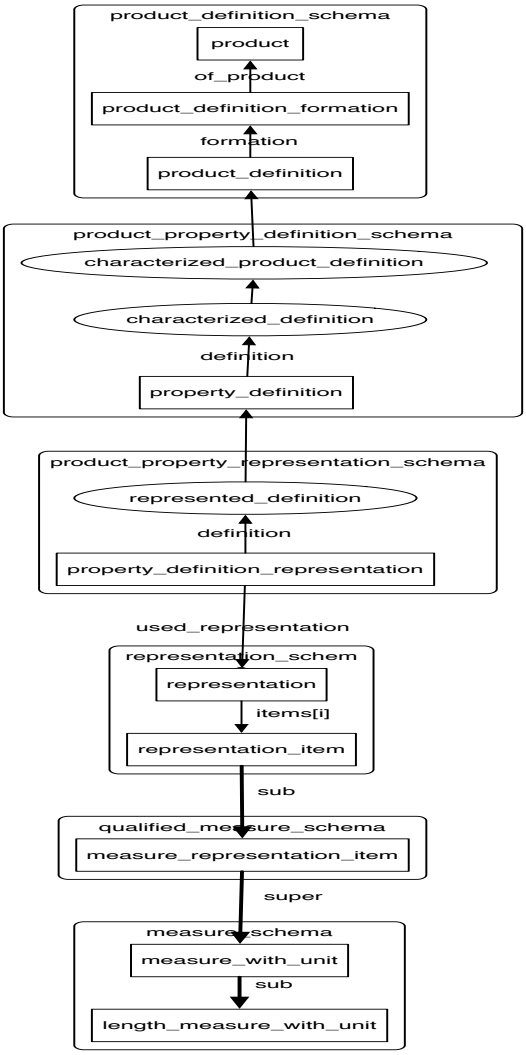
product <-\
product_definition_formation.of_product\
product_definition_formation <-\
product_definition.formation
characterized_product_definition = product_definition\
characterized_definition = characterized_product_definition\
characterized_definition <-\
property_definition.definition\
represented_definition = property_definition\
represented_definition <-\
property_definition_representation.definition\
property_definition_representation.used_representation ->\
representation.items[i] ->\
representation_item =>\
measure_representation_item <=\
measure_with_unit =>\
length_measure_with_unit

The graph in the dot format might look like:

```
digraph sample_path

  subgraph cluster0
    label = "product_definition_schema";

    product [shape=box];
    product_definition_formation [shape=box];
    product_definition [shape=box];
```

```
subgraph cluster1
  label = "product_property_definition_schema";
  shape=box;
  style=rounded;

  characterized_product_definition [shape=oval];
  characterized_definition [shape=oval];
  property_definition [shape=box];


subgraph cluster2
  label = "product_property_representation_schema";
  shape=box;
  style=rounded;

  represented_definition [shape=oval];
  property_definition_representation [shape=box];


subgraph cluster3
  label = "representation_schema";
  shape=box;
  style=rounded;

  representation [shape=box];
  representation_item [shape=box];


subgraph cluster4
  label = "qualified_measure_schema";
  shape=box;
  style=rounded;

  measure_representation_item [shape=box];


subgraph cluster5
  label = "measure_schema";
  shape=box;
  style=rounded;

  measure_with_unit [shape=box];
  length_measure_with_unit [shape=box];

product_definition_formation -> product [label="of_product"];
product_definition -> product_definition_formation [label="formation"];
characterized_product_definition -> product_definition;
characterized_definition -> characterized_product_definition;
property_definition -> characterized_definition [label="definition"];
represented_definition -> property_definition;
property_definition_representation -> represented_definition [label="definition"];
property_definition_representation -> representation [label="used_representation"];
representation -> representation_item [label="items[i]"];
representation_item -> measure_representation_item [label="sub",penwidth=3];
measure_representation_item -> measure_with_unit [label="super",penwidth=3];
measure_with_unit -> length_measure_with_unit [label="sub",penwidth=3];
```

note: In the above example backslash character "\" is inserted to avoid long lines in this document.



Post-processed path output. The layout tool executed a radial auto-layout to generate this layout.

note: The document processing engine introduced the distortion present in the text font in the figure.

9 Flatten EXPRESS Schema

```
eengine --flat -mode <mode> -schema mim_lf.exp -typeof noschema
```

-mode This specifies the kind of EXPRESS file being processed.

-schema This is the schema to be processed.

-typeof This specifies whether the TYPEOF comment preceding each ENTITY is written and what is included. Allowed values are **no**, **schema**, or **noschema**.

Possible values for *-typeof* option:

schema Means that the schema name is included in the TYPEOF string like this 'SCHEMA_NAME.ENTITY_NAME'.

noschema Means that the schema name is not included in the TYPEOF string like this 'ENTITY_NAME'.

no Means that the TYPEOF comment is not included at all. Instead a comment stating that no TYPEOF will be given.

This function writes out a flattened version of an EXPRESS Schema. The flattened EXPRESS schema has inherited attributes, inherited DERIVE, inherited INVERSE, inherited UNIQUE, and inherited WHERE shown as comments. The set of explicit attributes (including the inherited ones as comments) is written such that the order is correct for populating an instance in a Part 21 file. This was the original purpose for creating this *Express Engine* function.

Also SELECT TYPEs which extend another SELECT TYPE have the original values shown as comments.

ENUMERATION TYPEs which extend another ENUMERATION TYPE have the original values shown as comments.

Depending on the value of the `-typeof` option, there may be a comment just above each ENTITY declaration showing the TYPEOF value for that ENTITY.

> **NOTE:** At some future date, flatten will provide an option for including the description of each EXPRESS construct inherited. The description will be extracted from the xxx_descriptions.xml files on STEPmod.

```
Example 1

-- eengine --flat -schema schema.exp -typeof schema

-- TYPEOF => ('MIM.ACTION_ITEMS',
--       'MIM.ANGLE_DIRECTION_REFERENCE_WITH_A2P3D_SELECT',
--       'MIM.ATTRIBUTE_CLASSIFICATION_ITEM',
--       'MIM.ATTRIBUTE_LANGUAGE_ITEM',
--       'MIM.AXIS2_PLACEMENT',
--       'MIM.AXIS2_PLACEMENT_3D',
--       'MIM.CHANGE_MANAGEMENT_OBJECT',
--       'MIM.CLASSIFICATION_ITEM',
--       'MIM.DOCUMENT_REFERENCE_ITEM',
--       'MIM.DRAUGHTING_MODEL_ITEM_SELECT',
--       'MIM.DRAUGHTING_SUBFIGURE_REPRESENTATION_ITEM',
--       'MIM.DRAUGHTING_SYMBOL_REPRESENTATION_ITEM',
--       'MIM.FOUNDED_ITEM_SELECT',
--       'MIM.GEOMETRIC_MODEL_ITEM',
--       'MIM.GEOMETRIC_REPRESENTATION_ITEM',
--       'MIM.GROUPABLE_ITEM',
--       'MIM.INSPECTED_ELEMENT_SELECT',
--       'MIM.ITEM_IDENTIFIED_REPRESENTATION_USAGE_SELECT',
--       'MIM.KINEMATIC_LINK_REPRESENTATION_ITEMS',
--       'MIM.LAYERED_ITEM',
--       'MIM.MECHANICAL_DESIGN_GEOMETRIC_PRESENTATION_AREA_ITEMS',
--       'MIM.MECHANICAL_DESIGN_SHADED_PRESENTATION_AREA_ITEMS',
--       'MIM.MULTI_LANGUAGE_ATTRIBUTE_ITEM',
--       'MIM.ORIENTATION_BASIS_SELECT',
--       'MIM.PLACEMENT',
--       'MIM.POINT_PLACEMENT_SHAPE_REPRESENTATION_ITEM',
--       'MIM.REPRESENTATION_ITEM',
--       'MIM.RIGID_PLACEMENT',
--       'MIM.SHAPE_DIMENSION_REPRESENTATION_ITEM',
--       'MIM.SHAPE_REPRESENTATION_ITEM',
--       'MIM.SHAPE_REPRESENTATION_WITH_PARAMETERS_ITEMS',
--       'MIM.STYLED_ITEM_TARGET',
--       'MIM.STYLE_CONTEXT_SELECT',
--       'MIM.TEXT_STRING_REPRESENTATION_ITEM')
```

```
Example 1 continued

ENTITY axis2_placement_3d
    SUBTYPE OF (placement);
   -- representation_item.name : label;
   -- placement.location : cartesian_point;
   axis : OPTIONAL direction;
   ref_direction : OPTIONAL direction;
 DERIVE
   -- geometric_representation_item.dim : dimension_count :=
         dimension_of(SELF);
   p : LIST [3:3] OF direction := build_axes(axis,ref_direction);
 WHERE
   -- representation_item.WR1 : SIZEOF(using_representations(SELF)) > 0;
   -- geometric_representation_item.WR1 : SIZEOF(QUERY(using_rep <*
   using_representations(SELF) | NOT
   ('AP210_MIM_LF.GEOMETRIC_REPRESENTATION_CONTEXT' IN
     TYPEOF(using_rep.context_of_items)))) = 0;
   WR1 : SELF\placement.location.dim = 3;
   WR2 : (NOT (EXISTS(axis))) OR (axis.dim = 3);
   WR3 : (NOT (EXISTS(ref_direction))) OR (ref_direction.dim = 3);
   WR4 : (NOT (EXISTS(axis))) OR (NOT (EXISTS(ref_direction))) OR
          (cross_product(axis,ref_direction).magnitude > 0.0);
END_ENTITY; -- axis2_placement_3d
```

Example 2

```
-- eengine --flat -schema schema.exp -typeof noschema

-- TYPEOF => ('ACTION_ITEMS',
--            'ATTRIBUTE_CLASSIFICATION_ITEM'
--            'AXIS2_PLACEMENT_3D'
--            'DOCUMENT_REFERENCE_ITEM'
--            'DRAUGHTING_SUBFIGURE_REPRESENTATION_ITEM'
--            'FOUNDED_ITEM_SELECT'
--            'GROUPABLE_ITEM'
--            'ITEM_IDENTIFIED_REPRESENTATION_USAGE_SELECT'
--            'LAYERED_ITEM'
--            'MECHANICAL_DESIGN_SHADED_PRESENTATION_AREA_ITEMS',
--            'MULTI_LANGUAGE_ATTRIBUTE_ITEM'
--            'POINT_PLACEMENT_SHAPE_REPRESENTATION_ITEM'
--            'SHAPE_DIMENSION_REPRESENTATION_ITEM'
--            'SHAPE_REPRESENTATION_WITH_PARAMETERS_ITEMS'
--            'STYLE_CONTEXT_SELECT','TEXT_STRING_REPRESENTATION_ITEM')
ENTITY axis2_placement_3d
    SUBTYPE OF (placement);
   -- representation_item.name : label;
   -- placement.location : cartesian_point;
   axis : OPTIONAL direction;
   ref_direction : OPTIONAL direction;
 DERIVE
   -- geometric_representation_item.dim : dimension_count :=
        dimension_of(SELF);
   p : LIST [3:3] OF direction := build_axes(axis,ref_direction);
 WHERE
   -- representation_item.WR1 : SIZEOF(using_representations(SELF)) > 0;
   -- geometric_representation_item.WR1 :
      SIZEOF(QUERY(using_rep <* using_representations(SELF) | NOT
      ('AP210_MIM_LF.GEOMETRIC_REPRESENTATION_CONTEXT' IN
      TYPEOF(using_rep.context_of_items)))) = 0;
   WR1 : SELF\placement.location.dim = 3;
   WR2 : (NOT (EXISTS(axis))) OR (axis.dim = 3);
   WR3 : (NOT (EXISTS(ref_direction))) OR (ref_direction.dim = 3);
END_ENTITY; -- axis2_placement_3d
```

```
Example 3

-- eengine --flat -schema schema.exp -typeof no

-- -typeof option set to 'no'
ENTITY axis2_placement_3d
    SUBTYPE OF (placement);
   -- representation_item.name : label;
   -- placement.location : cartesian_point;
   axis : OPTIONAL direction;
   ref_direction : OPTIONAL direction;
 DERIVE
   -- geometric_representation_item.dim : dimension_count :=
       dimension_of(SELF);
   p : LIST [3:3] OF direction := build_axes(axis,ref_direction);
 WHERE
   -- representation_item.WR1 : SIZEOF(using_representations(SELF)) > 0;
   -- geometric_representation_item.WR1 :
     SIZEOF(QUERY(using_rep <* using_representations(SELF) | NOT
      ('AP210_MIM_LF.GEOMETRIC_REPRESENTATION_CONTEXT' IN
      TYPEOF(using_rep.context_of_items)))) = 0;
   WR1 : SELF\placement.location.dim = 3;
   WR2 : (NOT (EXISTS(axis))) OR (axis.dim = 3);
   WR3 : (NOT (EXISTS(ref_direction))) OR (ref_direction.dim = 3);
END_ENTITY; -- axis2_placement_3d
```

The schema name has been abbreviated within strings to help keep the strings shorter. Other edits have been made to make the document more readable. The actual EXPRESS in the examples is not necessarily the EXPRESS in the standard in order to fit the EXPRESS in the margins of the document.

10 Lisp EXPRESS File

This function takse an EXPRESS Schema, and writes it out to a file as a Lispified EXPRESS file.

It is currently still in development so this section of the manual is really just a placeholder for the real info that will be added later.

11 List EXPRESS Schema

This function takes an EXPRESS Schema, sorts the declarations, and writes out a list of the declarations found in the schema. Its output would look something like this:

```
*** Written by Express Engine v4.0.0
*** File: /home/craig/Desktop/exp-step/doc-examples/mim_lf.exp
*** 1 Schema: AP210_MIM_LF

TYPE AP210_MIM_LF.ABSORBED_DOSE_MEASURE
TYPE AP210_MIM_LF.ACCELERATION_MEASURE
TYPE AP210_MIM_LF.ACTION_ITEMS
...
ENTITY AP210_MIM_LF.ABRUPT_CHANGE_OF_SURFACE_NORMAL
ENTITY AP210_MIM_LF.ABSORBED_DOSE_MEASURE_WITH_UNIT
ENTITY AP210_MIM_LF.ABSORBED_DOSE_UNIT
...
FUNCTION AP210_MIM_LF.ABOVE_PLANE
FUNCTION AP210_MIM_LF.ACYCLIC
FUNCTION AP210_MIM_LF.ACYCLIC_COMPOSITE_TEXT
...
RULE AP210_MIM_LF.ALTERNATIVE_SOLUTION_REQUIRES_SOLUTION_DEFINITION
RULE AP210_MIM_LF.AP210_APPLICATION_PROTOCOL_DEFINITION_REQUIRED
RULE AP210_MIM_LF.AREA_COMPONENT_SHAPE_CONSTRAINT
...
```

Note that for this example the schema name has been shortened so that the output would fit within the margins of this document.

12 Pretty Print EXPRESS Schema

This function reads an EXPRESS Schema, sorts it and outputs it in the format specified by the *SC4 Supplementary Directives*. Unfortunately, those directives do not specify the format for all of the EXPRESS constructs that show up in a specific schema. In those cases we endeavour to use formats that are consistent with the specifications that are given.

Declarations are sorted into the order, `TYPE`, `ENTITY`, `SUBTYPE_CONSTRAINT`, `FUNCTION`, `RULE`, `PROCEDURE` as specified in the *SC4 Supplementary Directives*. Within each type grouping, the objects are sorted alphabetically by name.

For some EXPRESS schema this function can significantly enhance the readability of complex declarations by imposing alignment rules on the output file.

```
source EXPRESS:
ENTITY compound_feature
  SUBTYPE OF (feature_definition);
     WHERE
    WR1:  SIZEOF( QUERY( pds <* USEDIN( SELF,
          'AP242_MIM_LF.PROPERTY_DEFINITION.DEFINITION') |
          ('AP242_MIM_LF.PRODUCT_DEFINITION_SHAPE'
          IN TYPEOF(pds))  AND
         (SIZEOF( QUERY( csa <* USEDIN( pds,
         'AP242_MIM_LF.SHAPE_ASPECT.OF_SHAPE') |
         ((csa.name='compound feature in solid') AND
         ('AP242_MIM_LF.COMPOSITE_SHAPE_ASPECT'
         IN TYPEOF(csa))) )) = 1)  )) = 1;

As formatted by pretty function:
ENTITY compound_feature
  SUBTYPE OF (feature_definition);
WHERE
  WR1: SIZEOF(
       QUERY(pds <*
             USEDIN(SELF, 'AP242_MIM_LF.PROP.DEFINITION') |
             ('AP242_MIM_LF.PDS' IN TYPEOF(pds)) AND
             (SIZEOF(
              QUERY(csa <*
                    USEDIN(pds, 'AP242_MIM_LF.SA.OF_SHAPE') |
                    ((csa.name = 'compound feature in solid') AND
                     ('AP242_MIM_LF.CSA' IN TYPEOF(csa))))) =
              1))) =
       1;
note: The names have been truncated in this example to avoid
margin problems in this manual.
```

Another example output is:

```
(*
     $Id: mim_lf.exp,v 1.73 2015/11/13 14:12:24 kevletu Exp $
```

```
     ISO TC184/SC4/WG3 N8588 - ISO/TS 10303-410
      AP210 electronic assembly interconnect and packaging design -
      EXPRESS MIM Long form
     Supersedes ISO TC184/SC4/WG3 N8232
  Patched for:
  4665
  5056
  4992
  5057
  5004
  5690
  *)
(* ======================================================
============================== *)
(* Long form schema generated by The EXPRESS Data Manager
compiler version 9.8.9B 20130507*)
(* Wed Sep 30 14:27:01 2015                          *)
(* The schema is converted from ISO10303 P11-2003 to
ISO10303 P11-1994                    *)
(* ======================================================
============================== *)

(* Pretty Printed by Express Engine v4.0.0
   GIT ID: "568f5e5"
   Commandline:
     eengine --pretty
       -schema mim_lf.exp
       -mode mim_longform
*)

(* File: /stepmod/data/modules/\
ap210_electronic_assembly_interconnect_and_packaging_design/\
mim_lf.exp *)
(* backslash indicates a newline character inserted to avoid
overrunning margin in
this document.
That character does not exist in the actual schema file.
not all line breaks so introduced are documented by a backslash. *)
(* 1 Schema:
AP210_ELECTRONIC_ASSEMBLY_INTERCONNECT_AND_PACKAGING_DESIGN_MIM_LF
 *)

(*  Constants:                26 *)
(*  Entities:              2,205 *)
(*  Functions:               270 *)
(*  Procedures:                0 *)
(*  Rules:                    61 *)
```

```
(*  Subtype_Constraints:        0 *)
(*  Types:                    381 *)

SCHEMA \
  ap210_electronic_assembly_interconnect_and_packaging_design_mim_lf;

CONSTANT
  dummy_gri     : geometric_representation_item   :=
           representation_item('') || geometric_representation_item();
  dummy_tri     : topological_representation_item :=
          representation_item('') || topological_representation_item();
  schema_prefix : STRING                          := 'AP210_MIM_LF.';
...
  the_booleans  : elementary_space                :=
             make_elementary_space(es_booleans);
  ...
END_CONSTANT;

TYPE absorbed_dose_measure = REAL;
END_TYPE; -- absorbed_dose_measure

TYPE acceleration_measure = REAL;
END_TYPE; -- acceleration_measure

TYPE action_items = SELECT
  (action,
   action_directive,
   action_method,
   action_property,
   action_relationship,
   action_request_solution,
   alternate_product_relationship,
   applied_action_assignment,
   applied_classification_assignment,
   applied_external_identification_assignment,
   applied_person_and_organization_assignment,
   approval_status,
   ...
   versioned_action_request);
END_TYPE; -- action_items

TYPE actuated_direction = ENUMERATION OF
  (bidirectional,
   positive_only,
   negative_only,
   not_actuated);
END_TYPE; -- actuated_direction
```

```
ENTITY abrupt_change_of_surface_normal
  SUBTYPE OF (geometry_with_local_irregularity);
  SELF\shape_data_quality_criterion.assessment_specification :
              shape_data_quality_assessment_by_logical_test;
  small_vector_tolerance                 : length_measure;
  test_point_distance_tolerance          : length_measure;
WHERE
  WR1: validate_measured_data_type(SELF,
       'AP210_MIM_LF.BOOLEAN_VALUE');
  WR2: validate_inspected_elements_type(SELF,
       ['AP210_MIM_LF.SURFACE']);
  WR3: validate_locations_of_extreme_value_type(SELF,
       ['AP210_MIM_LF.POINT_ON_SURFACE',
        'AP210_MIM_LF.POINT_ON_SURFACE']);
  WR4: validate_accuracy_types(SELF, []);
END_ENTITY; -- abrupt_change_of_surface_normal

FUNCTION above_plane(p1 : cartesian_point;
                     p2 : cartesian_point;
                     p3 : cartesian_point;
                     p4 : cartesian_point) : REAL;
LOCAL
  dir2 : direction := dummy_gri || direction([1.0,0.0,0.0]);
  dir3 : direction := dummy_gri || direction([1.0,0.0,0.0]);
  dir4 : direction := dummy_gri || direction([1.0,0.0,0.0]);
  val  : REAL;
  mag  : REAL;
END_LOCAL;
  IF (p1.dim <> 3)
  THEN
    RETURN(?);
  END_IF;
  REPEAT i := 1 TO 3;
    dir2.direction_ratios[i] := p2.coordinates[i] - p1.coordinates[i];
    dir3.direction_ratios[i] := p3.coordinates[i] - p1.coordinates[i];
    dir4.direction_ratios[i] := p4.coordinates[i] - p1.coordinates[i];
    mag := dir4.direction_ratios[i] * dir4.direction_ratios[i];
  END_REPEAT;
  mag := SQRT(mag);
  val := mag * dot_product(dir4, cross_product(dir2, dir3).orientation);
  RETURN(val);
END_FUNCTION; -- above_plane

RULE alternative_solution_requires_solution_definition FOR
         (product_definition_formation);
LOCAL
```

```
  solution_versions : SET OF product_definition_formation := [];
END_LOCAL;
  solution_versions := QUERY(pdf <*
                       product_definition_formation |
                       SIZEOF(
                       QUERY(prpc <*
                       USEDIN(pdf.of_product,
   'AP210_MIM_LF.PRODUCT_RELATED_PRODUCT_CATEGORY.PRODUCTS') |
                       prpc.name = 'alternative solution')) = 1);
WHERE
  WR1: SIZEOF(
       QUERY(pdf <*
       solution_versions |
       SIZEOF(
       QUERY(pd <*
       USEDIN(pdf,'AP210_MIM_LF.PRODUCT_DEFINITION.FORMATION') |
       pd.frame_of_reference.name = 'alternative definition')) <>
       1)) = 0;
END_RULE; -- alternative_solution_requires_solution_definition

END_SCHEMA;
```

Note that the schema name was abbreviated to keep the strings shorter. Other textual modifications exist to fit in print margin.

13 Cross Reference STEP Data Population

This function takes a STEP Data Population and the EXPRESS Schema that it claims to conform to and outputs a list of what the values of the instance are by attribute and what other instances reference this instance.

Each instance is written as the pound symbol (#) followed by the number that is the id for the instance, followed by an equal sign (=) followed by the type of the instance. For an internal instance there is just a single ENTITY name. For an external instance the set of ENTITY names is written with plus signs (+) between adjacent names all enclosed within parentheses.

```
#42=ENTITY1
#42=(ENTITY2+ENTITY3+ENTITY4)
```

References to other instances are shown as a pound sign (#) followed by the number that is the instance id, followed by the type of the instance enclosed in angle brackets. Complex instances with have multiple ENTITY names separated by plus signs (+).

```
#50<ENTITY1>
#51<ENTITY2+ENTITY3+ENTITY4>
```

Now for a full example of the output of a STEP data population cross reference:

```
/* -*- Mode: Part21 -*- */

/* Written by Express Engine v4.0.0 */
/* Schema:
    /home/craig/Desktop/exp-step/doc-examples/mim_lf.exp */
/* Population:
    /home/craig/Desktop/exp-step/doc-examples/pic_3d_ap210.stp */

/* Schemata:
  AP210_ELECTRONIC_ASSEMBLY_INTERCONNECT_AND_PACKAGING_DESIGN_MIM_LF */

-- ================================================

------------------------------------------------
#73=(NAMED_UNIT+PLANE_ANGLE_UNIT+SI_UNIT)
    -- Named_Unit
    dimensions: *
    -- Plane_Angle_Unit
    -- Si_Unit
    prefix: $
    name: EU::RADIAN
  refs: #90<GEOMETRIC_REPRESENTATION_CONTEXT+
```

```
            GLOBAL_UNCERTAINTY_ASSIGNED_CONTEXT+
            GLOBAL_UNIT_ASSIGNED_CONTEXT+REPRESENTATION_CONTEXT>
...
-------------------------------------------------
#91=UNCERTAINTY_MEASURE_WITH_UNIT
    MEASURE_WITH_UNIT.value_component: length_measure(1.e-5)
 MEASURE_WITH_UNIT.unit_component: #77<LENGTH_UNIT+NAMED_UNIT+SI_UNIT>
    UNCERTAINTY_MEASURE_WITH_UNIT.name: 'distance_accuracy_value'
    UNCERTAINTY_MEASURE_WITH_UNIT.description: 'NONE'
  refs: #90<GEOMETRIC_REPRESENTATION_CONTEXT+
            GLOBAL_UNCERTAINTY_ASSIGNED_CONTEXT+
            GLOBAL_UNIT_ASSIGNED_CONTEXT+REPRESENTATION_CONTEXT>
-------------------------------------------------
#2000=COLOUR_RGB
    COLOUR_SPECIFICATION.name: 'BOARD'
    COLOUR_RGB.red: 1.0
    COLOUR_RGB.green: 0.0
    COLOUR_RGB.blue: 0.0
  refs: #2003<FILL_AREA_STYLE_COLOUR>,#2002<CURVE_STYLE>
-------------------------------------------------
#2053=CIRCLE
    REPRESENTATION_ITEM.name: ''
    CONIC.position: #2052<AXIS2_PLACEMENT_3D>
    CIRCLE.radius: 0.4
  refs: #2083<EDGE_CURVE>,#2080<EDGE_CURVE>
```

Some slight reformatting was done to keep the text within the margins of the document.

14 Test Compare

This function uses a STEPMod instance and a published Change Request to test the Compare Schema function by checking the XML output of Compare Schema.

```
[Removed output about loading schemata]

;;  ARM     MIM  Module
;; Failed  Failed  assembly_physical_requirement_allocation
;; Failed  Passed  derived_shape_element
;; Failed  Passed  feature_and_connection_zone
;; Failed  Passed  geometric_tolerance
;; Failed  Passed  kinematic_motion_representation
;; Failed  Passed  kinematic_state
;; Failed  Passed  kinematic_structure
;; Failed  Failed  non_feature_shape_element
;; Failed  Passed  part_external_reference
;; Failed  Passed  part_feature_function
;; Failed  Passed  part_shape
;; Failed  Passed  part_template
;; Passed  Passed  part_template_3d_shape
;; Failed  Passed  part_view_definition
;; Failed  Passed  product_and_manufacturing_annotation_presentation
;; Failed  Failed  product_and_manufacturing_information_with_nominal_3d_models
;; Failed  Passed  product_occurrence
;; Passed  Passed  product_placement
;; Failed  Failed  property_as_definition
```

The ARM column tells whether the arm_shortform schema *Passed* or *Failed* its test.

The MIM column tells wheter the mim_shortform schema *Passed* or *Failed* its test.

This function works by reading the published_index.xml file and identifying all of the included modules for the CR.

It then compares the arm.exp and mim.exp files with thier reference counter parts and then compares the resulting XML with the XML changes in the CR module.

15 Validate STEP Data Population

This function takes a STEP data population and the schema that it claims to conform to and runs all ENTITY and TYPE WHERE clauses, and RULE declarations. In addition, it checks instance values to make sure they conform to the attribute declarations within the ENTITY declarations.

15.1 Status of Validate Function

Unfortunately, Validate has fallen into disrepair. It will run, but is not guaranteed to work properly or completely. Eventually, we will spend the time necessary to make this function work properly.

Until then, this section will attempt to document what things are and are not working so that you can determine how useful the output is to you.

15.1.1 Working Validations

This section will contain a list of the validations that are known to work. This is a preliminary list. More effort will be expend later to make a more exhaustive list.

- Values in Instance match the TYPE of their respective attributes.
- Attributes that have been redeclared as DERIVE are filled with the value '*' in the instance.

15.1.2 Non-Working Validations

This section will contain a list of the validations that are known to not work.

- The population is not checked to make sure that it meets the expectations of the RULE declarations within the schema.
- INVERSE attributes are not checked.
- DERIVE attributes are not checked other than to make sure that any attribute that has been redeclared to be a DERIVE shows up in the instance with the value '*'.

Appendix A Command Line Arguments

This chapter will explain each of the arguments that can be passed to the various Express Tool Functions.

Option	Functions used in	Description
–xml-output	compare	If present this option indicates that the XML file should be written.
-cluster	dot, interface	Specifies schemata that should be considered a single node in an Interface –dot diagram. The value is a comma separated list of schemata where the first schema is the one that will show up in the diagram to represent the cluster and is drawn with a box around it, the rest are not drawn, but are considered part of the cluster. What this means is that any edges that point to or from any members of the cluster will point to or from the first schema. Any edges that point to and from members of the cluster to other members of the cluster are not drawn at all.
-cr_name	stepmod_test	Name of the Change Request.
-cr_pub_dir	stepmod_test	Directory for the published Change Request
-depth	dot, interface	Specifies the number of levels that should be included in the –*dot* graph. This number represents the number of levels other than the root which is level 0.
-graph	dot	Identifies the type of graph to be generated by the –*dot* function. Possible values: **interface** Draws a graph of the USE FROM/REFERENCE FROM relationships starting with a specified schema. **path** Draws a graph of the ENTITY and SELECT TYPE relationshps starting with a specified ENTITY and ending with a specified ENTITY.

-iface	dot, interface	Identifies which interfaces should be included in the graph. Not specifying this option is the same as specifying it with the value **both**. **both** Include both USE FROM and REFERENCE FROM interfaces. **use** Include only the USE FROM interfaces. **ref** Include only the REFERENCE FROM interfaces.
-rf_color	dot, interface	Specifies the name of the color to use for drawing the edge between a schema and the schema that it REFERENCE's. Possible values are: red, blue, green, yellow, cyan, magenta, orange. Default is green.
-uf_color	dot, interface	Specifies the name of the color to use for drawing the edge between a schema and the schema that it USE's. Possible values are: red, blue, green, yellow, cyan, magenta, orange. Defalut is blue.
-sub_color	dot, path	Specifies the name of the color to use for drawing the edge between an ENTITY TYPE and a SUBTYPE of that ENTITY TYPE. Possible values are: black, red, blue, green, yellow, cyan, magenta, orange. Default is black.
-super_color	dot, path	Specifies the name of the color to use for drawing the edge between an ENTITY TYPE and a SUPERTYPE of that ENTITY TYPE. Possible values are: black, red, blue, green, yellow, cyan, magenta, orange. Default is blue.
-attribute2-_color	dot, path	Specifies the name of the color to use for drawing the edge between an ENTITY TYPE and an ENTITY TYPE or SELECT TYPE that is specified as the target of the attribute. Possible values are: black, red, blue, green, yellow, cyan, magenta, orange. Default is red.
-select_color	dot, path	Specifies the name of the color to use for drawing the edge between a SELECT TYPE and an ENTITY TYPE or SELECT TYPE that is specified as the target of the SELECT TYPE. Possible values are: black, red, blue, green, yellow, cyan, magenta, orange. Default is red.
-leaf	dot, interface	Specifies schemata that should be treated like leaf nodes in the diagram. The value is a comma separated list of schemata. This means that no edges are draw from these nodes. Edges drawn to these nodes will be drawn normally.

-line	pretty	Specify the maximum length for a line. This basically set the right margin for the pretty printing function.

-mode dot, compare, concat, pretty

Possible values are **arm_longform**, **mim_longform**, **arm_shortform**, **mim_shortform**, **arm_concatenated**, and **mim_concatenated**. This indicates what kind of file is being processed.

arm_longform
mim_longform

The schema file read contains a longform schema. This means that the *-stepmod* option is not needed since a longform schema contains no USE FROM or REFERENCE FROM clauses.

arm_shortform
mim_shortform

The schema file read may contain a shortform schema or may be a concatenated file containing all necessary schemata. In the case that the schema file only contains a shortform schema, the *-stepmod* option is needed since a shortform schema normally contains USE FROM and/or REFERENCE FROM clauses. In the case that the schema file is a concatenated file that includes the shortform schema and all necessary interface schemas the value of the *-stepmod* option does not apply to this schema file and is ignored for this file.

note: It is certainly the case that in a compare operation, one of the schema files may be a concatenated file while the other file is a shortform file because internally eengine maintains a separate high level container for each schema (and interfaced schemata) being compared.

note: For concat, it only makes sense for mode to be arm_shortform, mim_shortform.

arm_concatenated
mim_concatenated

The schema file read contains a shortform schema plus all the referenced schemata. This means that the *-stepmod* options is not needed since while the schema is a shortform schema, the file contains all of the need extra schemata to satisfy all of the USE FROM and REFERENCE FROM clauses. The value of *-stepmod* option will be ignored.

-population	validate, xref	This is the STEP data population file.
-prune	dot, interface	Specifies a list of schemata that should be left out of the diagram. The value is a comma separated list of schemata. No edges drawn to any of the specified schemata will be drawn. No edges drawn from any of the specified schemata will be drawn, either. This effectively "prunes" out any subtrees of the graph that are rooted with these schemata.
-reference-_schema	compare	This is the reference EXPRESS schema file for the compare operation.
-reference-_stepmod	compare	This is the stepmod directory that should be used to resolve interface clauses in the *-reference_schema*.
-reference-_stepmod_vcs	compare	This sets the mode for accessing the VCS within the stepmod directory used for *-reference_schema*. Its value is one of **online**, **offline**, or **off**.
-schema	all	This is the EXPRESS schema file. It can take multiple schema files if they are separated by a comma and have no spaces around the comma: `-schema schema1.exp,schema2.exp,schema3.exp` Note that not all functions will use multiple schema files. Those that don't will write a warning about finding multiple -schema values and use only the first value.
-schema-_name	flat, pretty	This is the name of the schema to be compared when working with **arm_concatenated** or **mim_concatenated** files.
-stepmod	compare, concat, pretty, dot, flat, shtolo...	This is a pathname that specifies where the stepmod project has been checked out. This is used to load EXPRESS schema files called out in the interface clauses of the schema.
-stepmod-_vcs	compare, concat, pretty, dot, flat	This sets the mode for accessing the VCS within the stepmod directory. Its value is one of **online**, **offline**, or **off**.
-trial-_schema	compare	This is the EXPRESS schema file that is being compared to the reference schema specified with the *–reference_schema* option.
-trial-_stepmod	compare	This is the stepmod directory that should be used to resolve interface clauses in the *-trial_schema*.

-trial-_stepmod_vcs	compare	This sets the mode for accessing the VCS within the stepmod directory used for *-trial_schema*. Its value is one of **online**, **offline**, or **off**.
-typeof	flat	This specifies whether a TYPEOF comment should precede ENTITY declarations in the output from the *–flat* function. Possible values are **no**, **schema**, or **noschema**.

no Means that no TYPEOF info will be written. Instead, a comment that says that the –typeof option was specified with a value of 'no'.

schema Means that the TYPEOF info will include the schema name in the strings that are written.

noschema Means the TYPEOF info will not include the schema name in the strings that are written.

-vcs_exe	all	This specifies the full pathname to the **cvs.exe** file on Windows.

Appendix B Obtaining Software

Express Engine is hosted at SourceForge:

`https://sourceforge.net/projects/exp-engine/`

The source may be retrieved either by downloading it from the Files available on SourceForge or by checking it out from GIT by executing:

```
git clone git://git.code.sf.net/p/exp-engine/engine
```

Express Engine is written in Common Lisp. In order to compile it you will need either SBCL (`http://sbcl.org/`) or Clozure CL (`http://ccl.clozure.com/`) for the command line version or LispWorks for the graphical version.

Pre-compiled command line binaries for 32-bit and 64-bit linux, 64-bit mac os x 10.11.x 64-bit and 32-bit windows (Win7) will be made available.