Exppp – An EXPRESS Pretty Printer

Don Libes

Factory Automation Systems Division
National Institute of Standards and Technology
Gaithersburg, MD 20899

Abstract

EXPRESS is a data modeling language. EXPRESS is relatively new having only been standardized in 1993. Today, few tools exist that automatically generate EXPRESS and correspondingly most EXPRESS is hand-written. In the future, we predict that all but a tiny fraction of EXPRESS will be computer generated or computer read. While perhaps only during debugging, much of it will be read by humans so it is important that it "appear" as easy to read as possible.

The Supplementary Directives for the Drafting and Presentation of ISO 10303 state rules and guidelines for layout of EXPRESS. These rules and guidelines are incomplete and inadequate for humans, no less for the purposes of writing software to perform such presentation. Instead they appear to be an attempt to codify how people write EXPRESS now, with an eye towards making this easy for humans.

We provide improvements on their rules. These new rules are aimed at what we believe will be the biggest producers of EXPRESS in the future - programs. We have encoded these rules into a software library that can be incorporated into other tools. It has been used successfully in tools produced at NIST.

Keywords: EXPRESS; presentation; pretty print; PDES; STEP

Background

The PDES (Product Data Exchange using STEP) activity is the United States' effort in support of the Standard for the Exchange of Product Model Data (STEP). STEP is an emerging international standard for the interchange of product data between various vendors' CAD/CAM systems and other manufacturing-related software [1]. A National PDES Testbed has been established at the National Institute of Standards and Technology (NIST) to provide testing and validation facilities for the emerging standard. The Testbed is funded by the Department of Defense Continuous Acquisition and Life-Cycle (CALS) Program.

As part of the testing effort, NIST is charged with providing software for manipulating STEP data. Provided in the form of tools and toolkits for building new tools, the software is research-oriented and evolving. This document is one of a set of reports [2-12] which describe various aspects of the software.

Introduction

Clause 2 of the Supplementary Directives for the Drafting and Presentation of ISO 10303 [13] define rules and guidelines for layout of EXPRESS [14].[1] Coverage is far from complete. While many situations are described, only the simplest cases are described. Rules for dealing with complex but realistic schemas are not provided.

Providing sketchy guidelines may actually be quite appropriate. However, the Supplementary Directives fail to justify why this is. To the contrary, the Supplementary Directives leave the impression that future versions will indeed provide more complete guidelines. We will describe exactly why these things are best left unstated – at least for now. While it is not the intent of this paper to critique the Supplementary Directives, we will comment further on it.

In addition, we have produced a complete set of rules for printing EXPRESS. We have encoded these rules into a software library that can be incorporated into other tools. We will describe the library. The library has been used successfully in tools produced at NIST. We will describe how these tools make use of the library.

EXPRESS – a low-level language

EXPRESS is a STEP data modeling language. Currently, many people literally write EXPRESS statements "by hand". For example, when STEP Application Protocol developers write EXPRESS entities, they literally use an editor, hold down the shift key , and press the "E" key, the "N" key, the "T" key, the "I" key, the "T" key, and the "Y" key. They then release the shift key, and press the space bar, followed by the entity name and its definition.

There are aids for this. It is possible to cut and paste from existing schemas. Some editors are language-sensitive and can, for example, automate indenting and block comments. But, in general, the process is time-consuming and error-prone. Alternatives exist such as EXPRESS-G and translators that generate EXPRESS from other formats such as NIAM and IDEF1X.

Correspondingly, reading EXPRESS is also laborious. Most EXPRESS definitions have many references to distant definitions. Examining this with text editors requires a great deal of keystrokes to locate other entities in other schemas. Again, alternatives exist such as EXPRESS-G tools and browsers such as DataProbe [15].

The ASCII form of EXPRESS is not directly used by any tool. Intended as a neutral format, it is translated into other forms before being manipulated. This is understandable. As an ASCII text file, the semantics of the EXPRESS are not directly accessible except by – at the very least scanning, parsing, and an expensive semantic interpretation process.

There is thus, little benefit to actually writing and/or reading EXPRESS, at least directly by humans. Perhaps the only remaining reason is that the software that reads/writes EXPRESS may need to be

1. In the remainder of this paper, the NIST EXPRESS Pretty Printing Toolkit is referred to as "*the Exppp Toolkit*". The EXPRESS Language Reference Manual is referred to as the "*LRM*". The Supplementary Directives for the Drafting and Presentation of ISO 10303 are referred to as "*the Supplementary Directives*". To appease grammatical consistency, the Supplementary Directives may be thought of as the multitude of the directives themselves.

written, or may break and need to be debugged. For this reason alone, one may need to actually look at the raw EXPRESS itself.

While reading and writing raw EXPRESS is still commonplace today, it is reasonable to suggest that in the future, all but a tiny fraction of EXPRESS will be computer generated and computer consumed. The only time EXPRESS will be read and written by humans is during debugging.

EXPRESS is a format-free language. In particular, whitespace is ignored except to separate tokens. It is possible to separate every token by a single space or twenty spaces. Choosing one or the other does not change the meaning. Nonetheless, the Supplementary Directives address this very topic as if it were of great importance. On the other hand, the EXPRESS specification itself exhibits a great variety in formatting its examples.

While a particular style of formatting may be helpful during debugging, it is ultimately of minimal value. We suggest a compromise that allows adherence to the spirit of the Supplementary Directives with a minimum of effort. At the same time, we note that the Supplementary Directives rules are incomplete. Our rules are complete. At the same time, our rules are also parameterized providing for some flexibility. Thus, it is possible for others to follow our rules when writing formatters for different purposes.

The Supplementary Directives

In clause 2, the Supplementary Directives gives “rules and guidelines” to produce an “attractive and consistent layout” and a “uniform appearance”. Clause 2.1 goes on to say that “a uniform appearance helps the reader to find and use the material of interest”. This section provides several examples of how the Supplementary Directives inadequately address its own goals.

Incomplete

Unfortunately, there are actually few rules and most of them are presented as incomplete guidelines. For example, clause 2.1.7 (Use and reference layout) shows the following example:

```
USE FROM s-name
     (e1 AS s1,
      e2 AS s2);
```

and then says “If aliases are present and space on the line permits, spaces shall be used to align the AS tokens in tabular fashion. There is no explanation of what to do when space does not permit. Indeed, most of the other rules do not even mention this possibility.

In many cases, it is almost always the case that space does not permit. For example, clause 2.1.10-11 describe algorithm layout and parameters. The most complicated case described is:

```
FUNCTION func_name(a, b, c : INTEGER;
                   d, e, g : REAL;
                   x, y, z : AGGREGATE OF point) : REAL;
```

In practice, real names and types almost always cause these lines to be longer than the page width. However, there is no explanation of how to deal with this.

While dealing with line length is a common problem, the Supplementary Directives overlook other difficulties. For example, clause 2.1.9 (Type layout) does not mention the possibility of WHERE clauses within type definitions. And no mention at all is made of expressions.

Oriented towards Ease of Human Writing

Some of the guidelines for EXPRESS formatting seem as if they are intended for simplicity in writing rather than reading. Indeed, reading such EXPRESS is difficult when it could be made easier. For example clause 2.1.6 says “The schema keyword is written flush with the left margin . . . All declarations within a schema begin at the same margin.” This wording is vague, but evidently what Supplementary Directives refers to are declarations visible throughout the scope. The Supplementary Directives presents the example:

```
SCHEMA  . . .;

TYPE . . .;

ENTITY  . . .;

END_SCHEMA  . . .;
```

However, when multiple schemas are present, this style fails to visually group schema contents. It is very easy to miss an END_SCHEMA/SCHEMA pair when quickly scanning. An obvious improvement is to use the indentation rules that are given for most of the other scopes, namely, to indent the contents of a scope. This is more work with a dumb editor, but much easier to read.

One may argue that this kind of rule isn’t necessary at the schema level, but in practice this rule does indeed help.

Similar defects apply to the remainder of the Supplementary Directives. For instance, clause 2.1 says that “The layout rules explain how to use white space to produce a schema that has a uniform appearance. A uniform appearance helps the reader to find and use the material of interest.” Yet, the Supplementary Directives includes many deviations from uniformity, with no apparent benefit except to make EXPRESS easier to write by hand.

Improved Guidelines

It is not the purpose of this paper to critique and make improvements to the Supplementary Directives guidelines. Nonetheless, the implementation described does improve on these guidelines. Some new guidelines are mentioned below. Some other specifics are noted in the implementation section later.

Consistent Scope Indenting

All contents of a scope are indented. For example, everything in a schema is indented. Similarly, everything in an entity is indented, including SUPERTYPE expressions, UNIQUE clauses, etc. Similarly, with functions, rules, and other scopes.

User-defined names are indented twice. This allows different elements of a scope to stand out very clearly. For example:

```
ENTITY w_both;
    c1 : text;
    c2 : b;
  DERIVE
```

```
    c3 : text := c1;
  INVERSE
    c4 : b FOR b1;
END_ENTITY; -- w_both
```

End Statements are Named

Every END_XXX statement which matches a named declaration, has a comment appended which repeats the name. This is also demonstrated in the previous fragment.

Unmentioned Miscellany

A large number of rules used by the software do not contradict the Supplementary Directives because the Supplementary Directives simply do not discuss them. Many of these are mentioned elsewhere in the paper. The remainder can be found in the source code.

A Library to Implement EXPRESS Pretty Printing

We have encoded these rules into a software library that can be incorporated into other tools. Called "exppp" for short (which stands for "EXPRESS Pretty Print"), the Exppp toolkit assumes the in-memory representation built by the NIST EXPRESS Toolkit. This in-memory representation is a representation of the deep structure of an EXPRESS model.

There are two primary entry points into the library. They are EXPRESSout and SCHEMAout.

```
void EXPRESSout(Express e);
char *SCHEMAout(Schema s);
```

SCHEMAout

Given a schema s, SCHEMAout creates a file representing the schema. The file is named after the schema. Usually, the extension ".exp" is appended. For example, the statement "SCHEMA FOO" causes the creation of a file called "foo.exp".

If a ".exp" already exists and was not written by the Exppp Toolkit, the extension ".exp.pp" is used to name the file. This avoids the likely possibility of overwriting the source file while at the same time preventing unbounded additions of ".exp" suffixes and files named, for example, "foo.exp.exp.exp".

The Exppp Toolkit intuits whether it wrote the file by examining the first comment. The comment clearly identifies its purpose and the files authorship. If it has been changed, the file will not be overwritten. The comment written by the Exppp Toolkit is as follows:

```
(* This file was generated by exppp (an EXPRESS Pretty Printer)
written at the National Institute of Standards and Technology
by Don Libes, February 19, 1993.

WARNING: If you modify this file and want to save the changes,
delete this comment block or else the file will be rewritten
the next time exppp processes this schema. *)
```

If SCHEMAout successfully writes a schema file, a pointer to static storage is returned containing the filename. This memory will be overwritten on subsequent calls to SCHEMAout. If SCHEMA was not able to write a file, 0 is returned.

EXPRESSout

EXPRESSout calls SCHEMAout repeatedly, once for each schema in the EXPRESS object. In addition, EXPRESSout generates a file called “rmpp”. When run, this file deletes all the files created by the most recent use of the Exppp Toolkit. This includes all the “.exp” and “.exp.pp” files as well as “rmpp” itself. Creation of rmpp may be suppressed by setting the variable exppp_rmpp to false. This is useful when using exppp to generate schemas from another application rather than the standalone exppp program.

```
exppp_rmpp = false;
```

SCHEMAout calls functions for each object in the schema. These functions generate printable representations of the object to the current output file. There is generally a one to one correspondence between object and functions. These other functions will not be discussed here except for two noteworthy utility functions:

```
wrap(char *fmt, . . .)
raw(char *fmt, . . .)
```

These functions perform formatted output operations, formatting in the style of printf. Output is directed to either a file or character buffer according to the current context established by other functions discussed here.

Output is performed on a token by token basis. Except in rare instances, tokens are identical to those used by the NIST EXPRESS Toolkit scanner. This allows formatting to occur with a fairly natural appearance. The remainder of this section gives more details on this process.

All lines are automatically indented according to the number of scopes within which they are nested. For example, the SCHEMA keyword always appears at column 0, while the ENTITY keyword appears at column 2 (unless it is further nested). For example:

```
SCHEMA a;
  TYPE en = ENUMERATION OF
    (a,
     b,
     c);
  END_TYPE; -- en
```

In this example, TYPE is indented, and the list of enumeration items are further indented. The formatting of the enumeration list itself is specific to enumerations and is not controlled by the indent mechanism described here.

The default nesting indent is 2. This can be modified by setting the variable exppp_nesting_indent.

```
exppp_nesting_indent = 3;
```

The Exppp toolkit maintains a cursor which describes the distance from the left margin in character units. If the current output operation would move the cursor off the right side of the page, the output is wrapped if possible by inserting a newline between output operations. The wrap function permits this wrapping. The raw function does not permit this wrapping. For example, adding a semicolon to the end of a line should never cause a wrap. (The Supplementary Directives note that a semicolon should never appear on a line by itself.) Thus, semicolons are always written using the raw function.

For the same reasons, matching (right) parentheses are added this way. In general, data output with raw should either be a very small number of characters or should be sufficiently far from the right margin that there is no chance for wrapping to occur. No actual check is made of this.

By default, the line length is set to be 75 characters. This permits several additional characters to be added (presumably by raw) before 80, a traditional line width for terminals, is reached. The default line length may be modified by setting the variable exppp_linelength. For example:

```
exppp_linelength = 40;
```

When a line wraps, a new line is started and an indent occurs before more symbols are printed. In this case, the indent includes the usual nesting indent plus an additional continuation indent. The default continuation indent is 4 spaces. This can be changed by modifying the variable exppp_continuation_indent. For example:

```
exppp_continuation_indent = 1;
```

The following example shows the continuation indent because the assignment statement is too long to fit on a line.

```
IF (SIZEOF(Z) > 0) THEN
ELSE
  Y := QUERY ( X <* USEDIN(ITEM,'') | ((`REPRESENTATION_SCHEMA.' +
      `GEOMETRIC_REPRESENTATION_ITEM') IN TYPEOF(X)) );
  IF (SIZEOF(Y) > 0) THEN
    RETURN(DIMENSION_OF(Y[1]));
  ELSE
  RETURN(0);
END_IF;
```

Sorting

SCHEMAout and EXPRESSout automatically group printed object by type. As described by the Supplementary Directives, all constants are grouped together, then all types, and so on. Within each grouping, objects appear in alphabetical order.

This sorting is unnecessary given appropriately intelligent editors. For example, one can imagine visiting the definition of an object merely by clicking in its name appearing in another object. Software to implement such an editor would not care whether the objects were alphabetized or grouped.

Since alphabetizing is potentially time-intensive, the toolkit allows it to be skipped. In this case objects are produced in whatever order they are encountered in the in-memory data structure. Setting the variable exppp_alphabetize to false disables alphabetizing. For example:

```
exppp_alphabetize = false;
```

Other "out" Functions

A number of functions are defined that can be called directly. These are particularly useful when debugging, as they can print out small objects from within the debugger. The functions are:

```
ENTITYout(Entity e);
```

ENTITYout prints the entity e.

```
EXPRout(Expression e);
```

EXPRout prints the expression e.

```
FUNCout(Function fn);
```

FUNCout prints the function fn.

```
PROCout(Procedure p);
```

PROCout prints the procedure p.

```
RULEout(Rule r);
```

RULEout prints the rule r.

```
SCHEMAref_out(Schema s);
```

SCHEMAref_out prints the inter-schema references.

```
STMTout(Statement s);
```

STMTout prints the statement s.

```
TYPEout(Type t);
```

TYPEout prints a type definition in the form TYPE x = y; TYPE_END;

```
TYPEhead_out(Type t);
```

TYPEhead_out prints out a type head. For example, "y" in the previous example is a type head.

```
TYPEbody_out(Type t);
```

TYPEbody_out prints a type body. In the earlier example, the type body of x would be the type of y.

```
WHEREout(Linked_List where);
```

WHEREout prints a list of where clauses.

There is no function to print a schema.

String Functions

A number of functions are provided which provide a string representation of an object. Each function returns a string pointer which should be freed after it is no longer in use. For example, to display an entity, an application might say:

```
text = ENTITYto_string(e);
pop_up_text_window(text);
free(text);
```

The functions are as follows. They have the same per-object descriptions as the "out" functions above.

```
char *ENTITYto_string(Entity e);
char *EXPRto_string(Expression e);
```

```
char *FUNCto_string(Function f);
char *PROCto_string(Procedure p);
char *RULEto_string(Rule r);
char *SCHEMAref_to_string(Schema s);
char *STMTto_string(Statement s);
char *TYPEto_string(Type *t);
char *TYPEhead_to_string(Type *t);
char *TYPEbody_to_string(Type *t);
char *WHEREto_string(Linked_List where);
```

Buffer Functions

Another set of functions are useful when you can supply an output buffer in advance. These functions are similar to the string functions, but they also take a buffer pointer and a length. The length should describe how much space is available in the buffer. Each function returns an integer describing the length of resulting string (with its terminating null). A -1 is returned upon error.

These functions will execute faster than the string functions. They have the same per-object descriptions as the "string" functions above. The functions are as follows:

```
int ENTITYto_buffer(Entity e,char *buffer,int length);
int EXPRto_buffer(Expression e,char *buffer,int length);
int FUNCto_buffer(Function e,char *buffer,int length);
int PROCto_buffer(Procedure e,char *buffer,int length);
int RULEto_buffer(Rule e,char *buffer,int length);
int SCHEMAref_to_buffer(Schema s,char *buffer,int length);
int STMTto_buffer(Statement s,char *buffer,int length);
int TYPEto_buffer(Type t,char *buffer,int length);
int TYPEhead_to_buffer(Type t,char *buffer,int length);
int TYPEbody_to_buffer(Type t,char *buffer,int length);
int WHEREto_buffer(Linked_List w,char *buffer,int length);
```

Diagnostics

By default, the Exppp toolkit sends messages to the standard output stream describing what it is doing. For example, it prints the name of any schema files created. This may be inappropriate for programs that use Exppp as a library. To disable these diagnostics, set the variable exppp_terse to true. For example:

```
exppp_terse = true;
```

Note that even when exppp_terse is true, true errors will still be sent to the standard error stream.

Uses

The Exppp Toolkit has been used successfully in several ways. For example, the shtolo program [16] converts short forms to annotated listings. The annotated listing is printed by calling EXPRESSout on the EXPRESS object generated by shtolo. The Exppp Toolkit has also been used to display smaller pieces of the various objects maintained by the NIST EXPRESS Toolkit. The Exppp Toolkit is used in this way by fedex++ (which translates EXPRESS to C++), and also during debugging of virtually any NIST EXPRESS Toolkit application. We are currently working on a Tk-based application which will support browsing of NIST EXPRESS Toolkit objects. The representations of the objects will be generated by the Exppp Toolkit.

Big and Sticky Issues

A number of issues remain unresolved or have been solved but in unsatisfying ways. We encourage others to focus on these issues.

Comments

The NIST EXPRESS Toolkit strips comments (or "remarks" as they are called by EXPRESS) while reading in schemas. We have not made any attempt to recover comments for printing by the Exppp Toolkit. There are a number of reasons why we have not done anything further with comments.

It is impossible to automatically associate comments to EXPRESS objects. For example, comments may fall between any language token. The comment may be intended to describe the following token, the previous token, all tokens up to the next blank line, all tokens remaining in the schema, etc. It is impossible to intuit this. As the LRM says, "A remark . . . is only significant as white space."

Unfortunately, most comments are meaningless out of context. So the next question is, what is the context of a comment? For example, if there was originally a comment between two entities, with which entity does the comment belong? If the Exppp Toolkit simply printed pre-existing EXPRESS, there would be no problem – both entities would always exist and the comment would appear between them. In programs such as shtolo, however, both, one, or even neither of the entities might appear in the newly created schema. Should the comment appear? Additionally, entities may not appear in the order in which they originally appeared. Indeed, they might be far apart from one another. Where should the comment appear then?

This same problem occurs in other situations. The shtolo program prunes unused pieces of the EXPRESS model. What happens when the context of a comment is pruned? Or when some but not all of the context is pruned?

The NIST EXPRESS Toolkit stores a semantic model of the original EXPRESS. As long as the deep structure of the model is correct, the NIST EXPRESS Toolkit is free to represent schemas in any way. For example, the NIST EXPRESS Toolkit fills in missing subtype references implied by supertypes. This improves efficiency during semantic analysis and does not invalidate the model. However, it can no longer be used to recreate the original EXPRESS representation. Again, because of changed context, a comment may no longer be meaningful.

To take a step back, the idea of comments in EXPRESS is questionable at the outset. The language intends to formally state information models, yet it obviously allows the user to step outside the statement. To who is this additional information in the form of comments addressed? It does not mean anything formal in EXPRESS. In some sense it has no value whatsoever.

In reality, there are plenty of reasons for using comments. EXPRESS may lack the power to state something, or the user may not know how to use the power of EXPRESS, or may simply want to document that they were the author of such a change and when it occurred. In all of these and other circumstances it could be argued that using comments in the EXPRESS schema is the wrong mechanism to do this. In the first case, EXPRESS should be improved. In the second case, the user's knowledge should be improved, and in the last case, a software development system might be much more appropriate. Ultimately, a database and development environment such as the Application Protocol Information Base and Application Protocol Development Environment [17] is a much better place for recording information about EXPRESS information such as how the schemas are used, relate to the world, and who created them.

Expressions

Expressions are typical of constructions that often extend for many lines. The original indentation and breakpoints are not retained by the NIST EXPRESS toolkit so they must be regenerated by the Exppp Toolkit. There are no guidelines as to the presentation of expressions. Presentations vary from person to person. For example, some people add space around operators, some do not. Some people write query expressions so that arguments are on separate lines, some do not. In some cases, the decision depends on context, such as whether the expression is "complicated enough" whatever that might mean for a particular person.

The Exppp Toolkit has a number of rules to guide the printing of expressions. In particular, expressions are generally packed as tightly as possible, except: Lines wrap before the end of line and are indented enough to make it clear that the remaining part is a continuation. Each operator has a predefined idea of whether it should have space around it or on one side. For example, all comparison operators have a space between them and their operators, whereas the group operator has no space. Each operator may also decide to add parentheses. Parentheses are added in some cases to preserve correct binding, and in some cases to add readability. Some people might argue that any unnecessary parentheses inhibit readability, however the majority of EXPRESS users do not memorize the EXPRESS precedence table. A number of other miscellaneous rules are used to render good-looking expressions.

Preserving Case and Case Sensitivity

In EXPRESS, the case of letters in identifier names are not significant. For convenience in string comparisons made internal to the software, all identifiers are referenced as lowercase. The original case is discarded.

On output, all user identifiers are printed in lower case. Users may not appreciate this. It is possible to preserve case by using additional memory to store this information, however this is not currently seen as important (enough).

Future Enhancements

The most likely future enhancements are to add more flexibility to the software. It would be helpful to allow users more control over some formatting areas such as the amount of indentation in various types of situations. Currently, we have only added the control that is useful in our own development environment. To focus our effort, it would be helpful to know exactly what other people think is important.

A common request is to support the reproduction of comments in the original EXPRESS. Unfortunately however, we do not know an effective way to do this. This situation will probably not change in the near future.

Preserving case-sensitivity would be an easy modification. We would be interested to know how useful this is to people.

Obtaining the Exppp Toolkit

The Exppp Toolkit depends on the NIST EXPRESS Toolkit. The NIST EXPRESS Toolkit can be ftp'd from ftp.cme.nist.gov as pub/step/npttools/exptk.tar.Z. The associated installation documentation can be ftp'd as pub/step/nptdocs/exptk-obtaining-installing.ps.Z. The Exppp Toolkit can be ftp'd as pub/step/npttools/exppp.tar.Z. Installation is described in the README file.

Any of these files can be sent by e-mail. For example, to request e-mail delivery of the Exppp Toolkit, mail to "nptserver@cme.nist.gov". The contents of the message should be (no subject line), "send pub/step/npttools/exppp.tar.Z". Other files may be requested by modifying the message appropriately.

For Support or More Information

Contact the Factory Automation Systems Division – National PDES Testbed (1-301-975-3386 or **`npt-info@cme.nist.gov`**) for more information about the software in general, or other NIST projects at the National PDES Testbed.

This software is a research prototype, intended to spur development of commercial products. Most of the system is available in source form and you are encouraged to obtain and experiment with it. If you have questions and/or problems, you may send e-mail to the exppp@cme.nist.gov.

Acknowledgments

This work was funded by the NIST Scientific and Technical Research Services as part of the ARPA Persistent Object Base project, and the Department of Defense Department of Defense Continuous Acquisition and Life-Cycle (CALS) Program as part of the Application Protocol Development Environment project.

The author gratefully acknowledges Diane Allen (Northrop Corp), Allison Barnard Feeney, and the rest of the AP 202 Team who were very patient and understanding while trying to use early versions of the Exppp Toolkit.

Thanks to Allison Barnard Feeney and Diane Allen for significant improvements to the content and style of this paper.

Disclaimers

Trade names and company products are mentioned in the text in order to adequately specify experimental procedures and equipment used. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products are necessarily the best available for the purpose.

References

[1] Mason, H., ed., "Industrial Automation Systems – Product Data Representation and Exchange – Part 1: Overview and Fundamental Principles", Version 9, ISO TC184/SC4/WG PMAG Document N50, December 1991.

[2] Libes, Don, "The NIST EXPRESS Toolkit – Introduction and Overview", NISTIR 5242, National Institute of Standards and Technology, Gaithersburg, MD, September 1, 1993.

[3] Libes, Don, and Fowler, Jim, "The NIST EXPRESS Toolkit – Requirements", NISTIR 5212, National Institute of Standards and Technology, Gaithersburg, MD, June 9, 1993

[4] Libes, Don, "The NIST EXPRESS Toolkit – Design and Implementation", *Proceedings of the Seventh Annual ASME Engineering Database Symposium*, San Diego, CA, August 9-11, 1993.

[5] Libes, Don, and Clark, Steve, "The NIST EXPRESS Toolkit – Lessons Learned", *Proceedings of the 1992 EXPRESS Users' Group (EUG '92) Conference*, Dallas, Texas, October 17-18, 1992.

[6] Libes, Don, "The NIST EXPRESS Toolkit – Obtaining and Installing", NISTIR 5204, NISTIR National Institute of Standards and Technology, Gaithersburg, MD, September 1, 1993.

[7] Libes, Don, "The NIST EXPRESS Toolkit – Using Applications", NISTIR 5206, National Institute of Standards and Technology, Gaithersburg, MD, June 9, 1993

[8] Libes, Don, "The NIST EXPRESS Toolkit – Programmer's Reference", National Institute of Standards and Technology, Gaithersburg, MD, to appear.

[9] Libes, Don, "The NIST EXPRESS Toolkit – Creating Applications", National Institute of Standards and Technology, Gaithersburg, MD, to appear.

[10] Libes, Don, "The NIST EXPRESS Toolkit – Updating Existing Applications", NISTIR 5205, National Institute of Standards and Technology, Gaithersburg, MD, June 9, 1993.

[11] Clark, S.N., "The NIST Working Form for STEP", NISTIR 4351, National Institute of Standards and Technology, Gaithersburg, MD, November 1990.

[12] Clark, S.N., "NIST STEP Working Form Programmer's Reference", NISTIR 4353, National Institute of Standards and Technology, Gaithersburg, MD, November, 1990.

[13] Shaw, Nigel, "Supplementary directives for the drafting and presentation of ISO 10303 Version 1.0", ISO TC 184/SC 4 Editing N-22 Editing Committee, June 1993.

[14] Spiby, P., ed., "ISO 10303 Industrial Automation Systems – Product Data Representation and Exchange – Part 11: Description Methods: The EXPRESS Language Reference Manual", ISO DIS 10303-11:1992(E), July 15, 1992.

[15] Sauder, D., "Data Probe User's Guide", NISTIR 5141, National Institute of Standards and Technology, Gaithersburg, MD, March 1993.

[16] Libes, Don, "Shtolo: Converting STEP Short Listings to Annotated Listings", NISTIR 5291, Gaithersburg, MD, October 8, 1993.

[17] Clark, S.N., Feeney, A.B., and Fowler, J.E., “Specifications for an Application Protocol Development Environment, NISTIR 5248, National Institute of Standards and Tecnology, Gaithersburg, MD, August 19, 1993.