

A Language Enhancement Facility for COBOL – its Design and Implementation

J. M. TRIANCE AND P. J. LAYZELL*

Department of Computation, University of Manchester Institute of Science and Technology, P.O. Box 88, Manchester M60 1QD

A working party of the British Computer Society has designed a facility which allows users to configure their COBOL compilers to accept different dialects of COBOL. The facility, known as the COBOL Language Enhancement Facility (CLEF), was produced in response to users' requirements to enhance their versions of COBOL. This paper reviews the development of CLEF: looking at the users' requirements, identifying design criteria and outlining the design, implementation and evaluation of CLEF.

1. THE NEED FOR CLEF

There is a popular belief that COBOL users want a vigorously enforced standard language. This is a fallacy. Most COBOL users have a standard (ANS 74¹) COBOL compiler but are unhappy with the language it supports. A UK survey² revealed that 91% of the users questioned were dissatisfied with their COBOL compiler. The major reasons for dissatisfaction were:

- (a) omission of modern features (such as structured programming and data base);
- (b) the presence of features which are undesirable from the point of view of style or efficiency (such as the ALTER or GO TO statements);
- (c) differences from COBOL on other machines on which the programs are to run;
- (d) absence of applications-oriented features.

Symptoms of these deficiencies are:

- (a) the large number of pre-processors;
- (b) the prevalence of installation standards which confine programmers to a subset of the COBOL dialect supported by the compiler;
- (c) libraries of subroutines to provide functions which are not directly supported by the language.

Improvement to the standard could reduce the need for user variability, but the specific requirements of the users in different application areas will always demand different dialects of COBOL. It makes no more sense to limit all COBOL programmers to one common dialect than it would, for example, to demand that doctors and civil engineers use the same terminology and working practice.

This need for individual users to build on standard COBOL was recognised by a working party of the British Computer Society's COBOL Specialist Group. It designed the COBOL Language Enhancement Facility (CLEF) to satisfy this need.³

CLEF allows users to take an existing dialect of COBOL (the base language) and add new features, delete features and alter the run-time behaviour of features.

The remainder of this paper describes the design of CLEF by the BCS CLEF Working Party and its implementation and evaluation at UMIST in a project funded by the UK Science and Engineering Research Council. Since this paper covers the whole project,

detailed descriptions of each stage are not possible. For these, the reader is referred to refs. 3, 12 and 5 for the design, 13 and 11 for the implementation and 14 for the evaluation.

2. CLEF DESIGN CRITERIA

The design of CLEF is based on the following criteria.

(1) *Portability.* There is a need for portability of applications programs: CLEF must have no adverse effect on this.

(2) *Ease of learning.* It should be easy to learn how to specify Enhancements in CLEF.

(3) *Independent implementation of enhancements.* To keep implementations as straightforward as possible separate enhancements should be capable of independent implementation. Thus it should be possible, for example, to implement IF...END-IF construct without any danger of interfering with the implementation of the INITIALIZE statement.

(4) *Support for COBOL-like enhancements.* CLEF must permit enhancements which are COBOL-like.

(5) *Full support for enhancements.* The enhancements must be supported to the same level as the base language. New features must be fully validated and any errors reported in terms of the original source text.

(6) *High-level means of specifying enhancements.* The enhancements should be specified in a suitable high-level language.

3. DESIGN OF CLEF

CLEF is designed to permit users to add extra features to COBOL, delete existing features and alter the semantics, subjects to the design criteria stated in the previous section. This section demonstrates how CLEF was derived from these criteria.

3.1 Portability

To assist portability enhancements are supported by converting them into standard COBOL. For example, Fig. 1 shows the translation of a PERFORM...WITH TEST AFTER statement (as permitted in draft ANS 83 COBOL⁴) into standard COBOL. In Fig. 1, p-n represents a procedure-name and c represents a condition. The COBOL produced by the translation process is compiled by an existing compiler.

* To whom correspondence should be addressed.

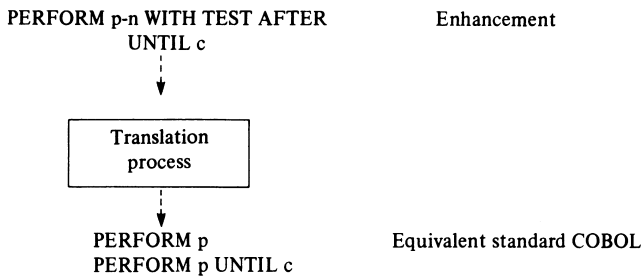


Figure 1. The PERFORM...WITH TEST AFTER enhancement

This design decision restricts the enhancements that can be supported by CLEF to those which are expressible in existing COBOL. However COBOL is a rich language, so in practice this is not a major constraint. It does not for example rule out any enhancement that can be achieved by pre-processors, called subroutines or installation standards. Furthermore CLEF does not prevent the user from generating CALLs to assembler subroutines; though a user who does this will be sacrificing portability. To maximise the portability of CLEF, COBOL is used for coding the translation process.

3.2 Ease of learning

It should be easy for the programmer responsible for implementing the enhancements to learn how to write the translation process. The choice of COBOL as the language for implementing the translation process assists in this: the programmer must already know COBOL to decide on the output from the translation process. In many respects the translation process is a typical data-processing program with input, output and processing. Where specialised facilities are required they have however been added to COBOL.

3.3 Independence

The translation processes for separate enhancements should be independent. For example, when writing the translation process for COBOL's new EVALUATE statement,⁴ the programmer should need to have no knowledge of how other new constructs, such as PERFORM...WITH TEST AFTER, are implemented and it should be impossible to corrupt the translation process for these other enhancements.

Independent routines for translating source text such as this are known as macros. They are invoked by a macro processor which scans the source text looking for enhancements (normally referred to as macro calls). The text output by the macro is referred to as generated text.

The action of a macro is summarised in Fig. 2. Within the macro there are three major components. They are as follows.

(1) Specification of the format of the macro call (known as the template) which is used by the macro processor to recognise macro calls within the source text and is used by the rest of the macro as a means of reference to parts of the macro call.

(2) A description of the generated text which is to be output.

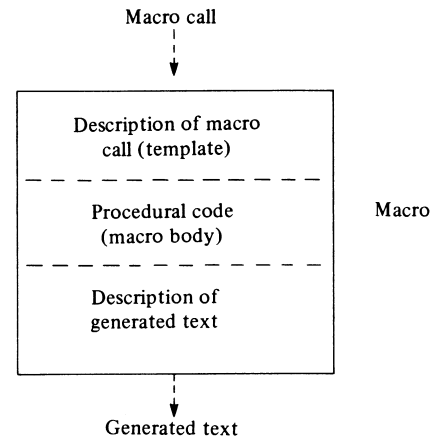


Figure 2. Overview of a macro

(3) Procedural code (or macro body) which examines the macro call, determines what text should replace it and then generates the text.

CLEF differs from existing macro processors in that it has a number of high level COBOL oriented features. The term 'CLEF processor' is used to refer to the CLEF macro processor and a 'CLEF program' is a CLEF macro. Each CLEF program is a separate COBOL program which is called by the CLEF processor wherever a relevant macro call is encountered. By the very nature of COBOL programs a great degree of independence of macros (CLEF programs) is assured.

3.4 Support for COBOL-like enhancements

CLEF permits any COBOL-like enhancements to be specified as a macro call. In other words anything that can be expressed in the COBOL metalanguage can be a macro call. Fig. 3 represents a sample of a macro call. The format may contain key words (underlined upper-case words in Fig. 3), optional words (other upper-case words), syntactic classes (lower-case words), options (enclosed by [and], alternatives (enclosed by { and }) and repetition (indicated by ...).

```
INITIALIZE identifier-1...
[REPLACING NUMERIC DATA BY {identifier-2}
                                literal }
```

Figure 3. Format of a macro call

The macro calls do not have to be preceded by any special symbol (or herald) and they obey the standard COBOL rules for layout and continuation across lines.

Macro calls thus look just like any COBOL statement – in fact the programmer may well be unaware of which statements are supported by CLEF and which ones are supported directly by the compiler.

3.5 Full support for enhancements

In the previous section it was established that the macro calls are indistinguishable in appearance from the base language COBOL statements. Since they are to become part of the COBOL programmer's language, they should also be indistinguishable in the support the CLEF processor provides for them. The enhancements should

be fully checked for errors, and any errors encountered should be reported in terms of the source code (the macro call). It is not acceptable for any errors in the macro call to be transmitted to the generated text because the resulting error messages, issued by the compiler, would refer to code that the programmer did not write and does not therefore understand. In addition to detecting any error within the macro call the CLEF processor should also reject the macro call if it appears in the wrong context – if for example a new Procedure Division statement is specified in the Data Division.

3.6 High-level means of specifying enhancements

The aim is to provide the macro writer with a high-level programming language – at least as high-level as COBOL is for normal data-processing operations.

In addition to normal data-processing operations the CLEF program must be able to do the following:

- (a) describe the format of the macro call;
- (b) specify the context in which the macro call may appear;
- (c) indicate the destination of the generated text (the 'current position' in the program being processed, the 'working-storage section', etc.);
- (d) discover information about the rest of the program being processed (the attributes of data items, etc.).

With each of these functions the emphasis has been on shifting from the CLEF programmer on to the CLEF processor the burden of validating macro calls and of ensuring the integrity of the generated text.

The CLEF programmer specifies the format of the macro call in a form equivalent to that of the COBOL metalanguage (see Fig. 3) in a special format section. Fig. 4 shows a format section equivalent to the format in Fig. 3.

```

FORMAT SECTION.
1  INITIALISE-STATEMENT.
2  FILLER          KEY-WORD
                     << INITIALIZE >>.
2  FILLER          ARGUMENT IDENTIFIER >
                     OCCURS MINIMUM 1 TIMES
                     DEPENDING ON ID-COUNT.

2  REPLACING-
   PHRASE          OPTIONAL.
3  FILLER          KEYWORDS << REPLACING
                     NUMERIC >>.
3  FILLER          NOISE-WORDS << DATA >>.
3  FILLER          KEY-WORD << BY >>.
3  FILLER ALTERNATIVES.
   5 FILLER        ARGUMENT IDENTIFIER.
   5 FILLER        ARGUMENT LITERAL.

```

Figure 4. A CLEF format section

The Format Section is extracted from the CLEF program in a special CLEF library run and is used by the CLEF processor to detect and fully validate macro calls. This validation ensures that the reserved words appear as specified and that the syntactic classes (identifier and literal in this example) are also valid.

The context of the macro call is specified by a special clause in the Program-Id paragraph of the CLEF program. The CLEF processor is responsible for ensuring

that the macro is only matched in the specified context. The context may be any syntactic class (imperative-statement, data description clause, etc.). Thus a macro of context imperative-statement may appear anywhere in the source program that an imperative-statement may appear, and nowhere else.

A special Generated-Text Section is used to specify text which is to be output by the CLEF program. Within this section each piece of text is labelled according to its destination within the source program (working-storage section, file section, current position, etc.). When the text is output from the CLEF program the CLEF processor will ensure that it is stored in the correct position in the source program.

Some macros will require access to the data attributes to determine what code should be generated. For example the default action of the INITIALIZE statement in draft ANS COBOL⁴ is to move zeros to numeric fields and spaces to alphanumeric fields. Thus the data definition of any identifier specified in the INITIALIZE statement must be checked to see whether it is numeric or alphanumeric. The CLEF processor stores all the information about the data items, files and other entities in a property table which is made available to CLEF programs.

3.7 Summary of design

This section has described the major aspects of the design of CLEF and related them to the design criteria. A full description of CLEF appears elsewhere,³ as does a fuller justification of the design decisions.^{5, 12}

4. IMPLEMENTATION OF CLEF

An implementation of CLEF has been developed at UMIST^{11, 13} and is described in this section.

4.1 Implementation options

Two strategies exist for the implementation of CLEF: a pre-processor implementation and a compiler-integrated implementation.

A pre-processor has the following advantages.

(1) The implementation is compiler-independent and so can be portable between compilers.

(2) The overall size of an implementation can be minimised by keeping the CLEF facility separate from the compiler.

A compiler-integrated CLEF implementation has the following advantages.

(1) There is no duplication of the lexical and syntax analysers. In a pre-processor implementation, the input source must be fully validated and thus a full syntax analysis is necessary. This can only be achieved by full lexical and syntax analysis by the pre-processor (as well as the compiler).

(2) The use of one lexical analyser and one syntax analyser, together with a single error processor, helps to provide a uniform level of support for both the base language and enhancements.

(3) Problems of interfacing with symbolic run-time debugging packages are overcome with a compiler-integrated approach.

These two approaches are not as different as they might appear. Because the pre-processor must do full syntax checking it is very similar to the front-end of a compiler. The only difference is that the pre-processor will generate COBOL object code whereas compilers normally generate some lower-level code.

The UMIST implementation is a pre-processor and thus demonstrates the feasibility of both approaches.

4.2 Overview of the compiler

The COBOL compiler with an integrated CLEF facility developed at UMIST has the following characteristics:

- (a) the compiler is one-pass;
- (b) the compiler is based upon a syntax-driven design^{7,8} with a single set of syntax diagrams that define both the base COBOL language and the CLEF extensions that are acceptable to the compiler;
- (c) the syntax diagrams that drive the compiler are constructed so as to minimize the need for backtracking during the parse of an input source;
- (d) the compiler can return to the stage at which it processed any previous part of the program so as to compile generated text that may be produced after a macro call;
- (e) the compiler is controlled by a scheduling process that detects the production of generated text and can, if necessary, suspend compilation of the current part of the COBOL program and compile the generated text first.

4.2.1 Configuring the compiler

With most implementations of extensible language processors, some configuration stage^{6,9} is necessary before the processor can be used. In CLEF, this configuration stage is known as the CLEF library run. Fig. 5 shows the production of a CLEF library by means of the CLEF library run and the subsequent use of the library in the compilation of COBOL applications programs.

The purpose of the CLEF library run is to convert the CLEF programs written by a user into a form suitable for use by the compiler. This process involves:

- (a) lexical analysis, syntax analysis and semantic checking of each CLEF program;
- (b) encoding of each CLEF program format section in the form of syntax diagrams (for subsequent use by the compiler's syntax analyser when checking for macro calls);
- (c) identification of new reserved words;
- (d) conversion of the CLEF program into executable code (for execution whenever the compiler matches the source text against the associated format section);
- (e) updating the CLEF library with the new syntax diagrams, reserved words and CLEF object programs.

Before the library is used by the compiler, some optimization is performed by the UMIST implementation.

(1) The syntax diagrams that represent the format sections are combined with the syntax diagrams that

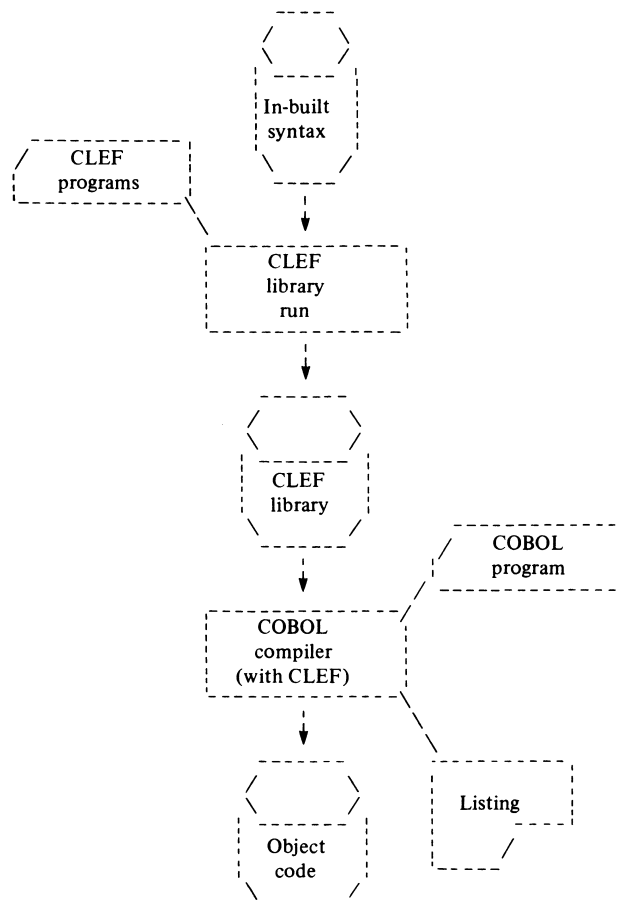


Figure 5. Overall implementation strategy

represent the base language. By adding the syntax diagrams of the enhancements in the correct position of the syntax diagrams of the base language, the context of the macro calls is automatically validated by the syntax analyser.

(2) The syntax diagrams are 'factorized' to minimise the need for backtracking during syntax analysis.

Once all the CLEF programs to support the desired dialect of COBOL have been added to the CLEF library, the compilation of COBOL applications programs is possible. These COBOL programs may contain any enhancements supported by CLEF programs contained in the CLEF library currently available to the compiler.

4.2.2 Running the compiler

When the desired macro call formats have been added to the rest of the language, the compiler is ready for use. A user submits his COBOL programs, possibly containing CLEF-supported language enhancements, in the normal way. The only distinction made between the base COBOL and the enhancements supported by CLEF is when the syntax has been validated. The syntax analyser passes control to the semantic analyser which either generates object code, in the case of a base language feature, or calls the associated CLEF program, in the case of a CLEF-supported feature.

If the syntax analyser cannot match the input source against either the base language or CLEF-supported

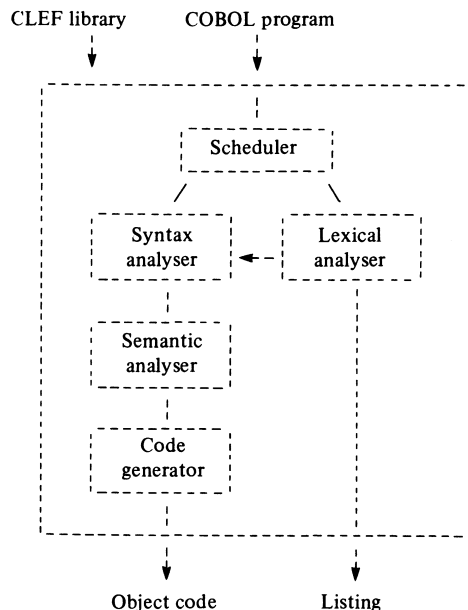


Figure 6. Structure of the CLEF compiler

enhancements, an error processor is called that will report the error and proceed with error recovery, irrespective of whether the error has occurred in the base language or macro call.

The error processor reports the error by writing the appropriate message to an error listing. CLEF programs can also direct messages to this listing. Thus the programmer receives a single error listing containing both CLEF program-generated error messages and base language error messages, presented in a consistent format. The error processor then attempts a local error recovery. If this fails a global error recovery¹⁰ is performed, whereby the error processor skips through the source text until a period, another recognisable symbol or end of source is encountered. Processing is resumed from this point.

The structure of the compiler is shown in Fig. 6. Its construction is similar to existing compilers with the exception of the scheduler, whose function is to direct the compiler to process any generated text produced after the execution of a CLEF program.

During the validation of data description entries, the semantic analyser constructs a property table (or symbol table) for use by the compiler. The property table is constructed in such a way that CLEF programs can easily access the information stored. This enables the CLEF

programs to perform a full semantic analysis of macro calls.

5. EVALUATION OF THE CLEF FACILITY

The final stage of the CLEF project at UMIST was an evaluation to ensure that CLEF could support a sufficiently high number of enhancements to COBOL required by users.

The evaluation followed the following steps:

- an extensive set of COBOL enhancements were collected (340 in all) by surveying COBOL users² and using the changes between the ANS 74 COBOL standard and the draft ANS 83 COBOL standard;
- each enhancement was categorised on a number of criteria;
- a cross-section of enhancements was identified which between them displayed all the characteristics identified by the categorisation;
- CLEF programs were designed and implemented for the selected enhancements;
- the implemented CLEF programs were tested.

The evaluation of CLEF showed that between 70% and 80% of the enhancements could be implemented by CLEF, thus demonstrating its viability.

6. THE FUTURE

The feasibility of the CLEF facility has been demonstrated and its potential has been recognised by the UK Science and Engineering Research Council and ICL. These two organisations are sponsors of a project at UMIST to develop a CLEF facility for ICL machines. The aim is to bring the benefits of the University research to commercial programmers. In the longer term it is the aim of the CLEF Working Party to incorporate CLEF into standard COBOL, thereby offering the benefits of CLEF to all COBOL users. It is hoped that in due course each installation will be able to choose the dialect of COBOL that suits it in the same way as it can currently choose its own hardware configuration.

Acknowledgement

The specifications of CLEF were produced by the BCS CLEF Working Party. The work at UMIST was sponsored by the UK Science and Engineering Research Council.

REFERENCES

1. Ansi. *American National Standard Programming Language COBOL, X3.23* (1974). [Official COBOL specification.]
2. P. J. Layzell, *A summary of the results of the BCS COBOL users' questionnaire*, Computation Department Report 257, UMIST, Manchester, England (1980). [Summary of results of a questionnaire to COBOL users on the suitability of their versions of COBOL.]
3. P. J. Layzell, *CLEF Journal of Development*, Computation Department Report 258, UMIST, Manchester, England (1981). [Official specification of CLEF.]
4. ANSI, *Revised X3.23 American National Standard Programming Language COBOL* (1981). [Draft of the proposed official COBOL specification to replace ANS 74 COBOL.]
5. J. M. Triance, *The design and evaluation of a language enhancement facility for COBOL*, M.Sc. thesis, Computation Department, UMIST, Manchester, England (1981). [Describes the design principles behind CLEF.]
6. J. M. Triance & J. F. S. Yow, MCOBOL – a prototype macro facility for COBOL, *Comm. ACM.* **23**, 8, 432–439. [Introduction to MCOBOL, a COBOL macro facility and forerunner of CLEF.]

7. M. E. Conway, Design of a separable transition-diagram compiler, *Comm. ACM.* 6, 7, 396-408. [Describes a syntax-driven COBOL compiler.]
8. P. J. Layzell & J. M. Triance, *Syntax-driven COBOL Compilers*, Computation Department Report 275, UMIST, Manchester, England (1982). [An introduction to syntax-driven compilers.]
9. Cobra Systems and Programming, *Cobra Users Manual*, 21 Green Hill Road, Camberley, Surrey, England (1982). [An introduction to the Cobra macro processor.]
10. A. B. Pai & R. B. Kieburtz, Global context recovery: a new strategy for syntactic error recovery by table-driven parsers, *ACM Transactions on programmed language systems* 2, 1, 18-41 (1980). [Describes a global error recovery technique for syntax driven compilers.]
11. P. J. Layzell, The implementation of a COBOL language enhancement facility, Ph.D. thesis, UMIST, Manchester, England (1982). [Describes an implementation of CLEF.]
12. J. M. Triance & P. J. Layzell, *CLEF-A COBOL Language Enhancement Facility*, Computation Department Report 273, Manchester, England (1983). [Outlines the basic concepts of the CLEF facility.]
13. P. J. Layzell & J. M. Triance, *Implementation of a COBOL Language Enhancement Facility*, Computation Department Report 274, Manchester, England (1982). [Describes an implementation of CLEF.]
14. O-C. C. Lin, The Evaluation of the COBOL Language Enhancement Facility, M.Sc. Dissertation, Computation Department, UMIST, Manchester, England (1982). [Describes an evaluation of the CLEF facility.]