A Proposal for an Automated Documentation System in FORTRAN 90

Document No.: PL-LFI-OAT-TN-12.3

Issue/Rev. No.: 0.0

Date: 2000-May-31

Prepared by: Michele Maris

Authorized by: Fabio Pasian

Abstract

A macro processor is able to scan a F90 code, to extract automatically comments and to convert them into a L_AT_EX document. Joining this possibility with a mark-up language interspersed into the F90 code it is possible to use the source code itself as the source for documentation and Man Pages, in a way analogous to PERL 5.0 PODs. A simplified mark-up language is introduced here. The SGML syntax (i.e. marks with the form <*mark-name*> … </*mark-name*>) is used to allow future expandability toward XML/SGML interpreters. In order to avoid interfering with the F90 compiler marks are added to comments so that the compiler will skip it.

Document Approval

Name	Organization	Signature	Date
F.Pasian LFI DPC Manager	OAT		2000 May 25

Document Status Sheet

1. Document Title: An LFI data processor to create a Quick-Look Source Catalogue			
2. Issue	**3. Revision**	**4. Date**	**5. Reason for change**
N	0	2000-May-25	Initial version

Table of Contents

1. Introduction

In order to assess a proper documentation, program manuals and remarks would be keep constantly synchronised release after release. As it is know this is a tedious and time-consuming process which produces frequent errors, and miss-understandings. A code preprocessor would be able to scan a F90 code, to extract automatically comments and to convert them into a LaTeX document. While this is straightforward, the next natural step: to produce a well-formatted document, i.e. a properly divided and paged one, is again long and time consuming, since it requires a full reediting of the automatically generated file. The best way to overcome such problems is to introduce some form of notation in the comments, allowing the preprocessor to interpret the remarks content and takes appropriate actions in order to obtain a well formatted document. The best way to perform this is to use a mark-up notation, based on a mark-up language, in order to classify the information interspersed in the F90 code. The validity of this method to proceed is testified by the success of PERL 5.0 PODs. However Perl PODs are PERL specific i.e. a full new mark-up notation has to be learned and introduced in order to apply it. A better choice for a syntax is the SGML syntax, where marks has the form <*mark-name*> … </*mark-name*>.

2. Requirements

Requirements for this mark-up language are:

Re.1 Marks has not to interfere with F90 compiler.

Re.2 Easy to learn, remember and use.

Re.3 It has not to reduce the F90 code readability.

Re.4 It has not to force the introduction of redundant information, i.e., informations which may be easily recovered from the code.

Re.5 Comments classification must be based on already defined comments scope.

3. Basic Rules

Previous requirements immediately suggests a set of basic rules:

BRu.1 Marks have to be introduced into comment lines.

BRu.2 Marks syntax musts resemble well know, diffused, existing conventions.

BRu.3 It must be case insensitive.

BRu.4 Few, short commands, i.e. not verbose commands.

BRu.5 Whenever possible the preprocessor must be able to extract informations from the code itself.

BRu.6 The preprocessor must be robust enough to allow operate even if syntax errors or lack of information occurs. In this case the preprocessor must warn the operator but it is not required to reconstruct the lacking information.

As a consequence of these rules and in particular of (**BRu.2**) we make the following choice:

CH.1 Tags syntax is SGML conformant.

Other reasons to prefer SGML to other well know scientific markup notations, such as TeX/LaTeX, are:

RS.1 It has a more simple syntax than TeX/LaTeX tags.

RS.2 It allows the definition of formulae and equations as well

RS.3 Its less verbose structure and its case insensitivity make it less prone to errors than TeX/LaTeX notation.

RS.4 The SGML notation is used to define very well know markup languages as: HTML and XML.

RS.5 SGML and its child are well recognized standard even outside the scientific community.

RS.6 Many of the already defined requirements and rules as (**BRu.3**, **BRu.6**) maps easily in the SGML/HTML/XML rules.

4. Code Model

The code model defines a model for the input code. We assume the following model:

CMD.1 A code is collection of lines separed by a \n character

CMD.2 Lines may be:

i. comment lines:

lines which contains only spaces and a comment

F90 regular expression: [^\s]!+~[]

ii. executable lines:

lines which do not conform to the definition of comment lines

CMD.3 Comment block

i. A comment block is an uninterrupted sequence of comment lines

ii. It begins with the begin of file or the first comment line after a sequence of code lines

iii. It ends with the end of file or a executable line

CMD.4 Marked Comment Lines

Comment lines belonging to a comment block, with marks inside

CMD.4 Marked Comment Block

i. An uninterrupted list of Marked Comment Lines

ii. or comment lines belonging to the scope of a mark.

The reason which leads to the implicit (**CMD.2**) exclusion of mixed lines i.e. lines of code + comment, as valid comment lines is to avoid complex situations as an "**!**" or a tag into a string which will be difficult to be handled by the preprocessor.

5. Contest

The concept of *contest* allows to complain to (**Re.4**), since through Contests the preprocessor is sensitive to the code which is scanning and so may extract recover some information from it. To prevent an excessive complexity, contest follows few, well-defined rules, and only few contests are recognized and handled.

5.1 Contest Rules

The rules for contests are listed below, the reason for (**CO.3.ii**) and (**CO.3.iii**) is to allow flexibility to the preprocessor in the case the end pattern is not recognized, since most of these patterns presently are not mandatory in FORTRAN90.

CO.1 A contest is defined by the block of executable lines where the Marked Comment Block is inserted.

CO.2 A FORTRAN 90 Contest Pattern must be a valid (compilable) FORTRAN 90 construct.

CO.3 The contest begins when a FORTRAN 90 Contest Pattern is recognized by the preprocessor.

CO.3 The contest ends when either:

i. A FORTRAN 90 Contest Pattern is recognized.

ii. A new Contest Begins.

iii. The File ends.

CO.4 Nidified contests are not allowed.

CO.5 The following contests are recognized: *External*, *Main*, *Subroutine*, *Function*, *Module*.

5.2 Contest Descriptions

These descriptions specifies: the name, the situation, when it is applied, the begin pattern, the end pattern, what data, extracted from the FORTRAN 90 code, are handled:

5.2.1 External

Situation: The Marked Comment Block is outside any contest

Applied: This is the default at the begin of each scan

Begin Pattern: NONE

End Pattern: NONE

Code Handled: NONE

5.2.2 Main

Situation: The Marked Comment Block belongs to a PROGRAM

Applied: Inside a PROGRAM

Begin Pattern: PROGRAM <<name>>

End Pattern: END PROGRAM <<name>>

Code Handled: Program <<name>>

5.2.3 Subroutine

Situation: The Marked Comment Block belongs to a SUBROUTINE

Applied: Inside a SUBROUTINE

Begin Pattern: SUBROUTINE <<name>>

End Pattern: END SUBROUTINE <<name>>

Code Handled: Subroutine Interface

5.2.4 Function

Situation: The Marked Comment Block belongs to a FUNCTION

Applied: Inside a Function

Begin Pattern: FUNCTION <<name>>

End Pattern: END FUNCTION <<name>>

Code Handled: Function Interface

5.2.5 Module

Situation: The Marked Comment Block belongs to a Module

Begin Pattern: MODULE <<name>>

End Pattern: END MODULE <<name>>

Code Handled: Module <<name>>

6. Document Content Model

The automated documentation must reflects a document content model. The document content model describes what the document contains, not how it is formatted. In accord with our model each document must report one or more of the following items (M = Mandatory, F = Facoltative):

DMCRu.1 M Author(s) information (including updates)

DMCRu.2 M Release information (release numbers and dates)

DMCRu.3 M Description (a description of the code)

DMCRu.6 M Synopsis

DMCRu.5 F References to other documents

DMCRu.7 F Bugs, Warnings, Hypothesis for use

DMCRu.8 F Validation informations and certifications

DMCRu.4 F History

DMCRu.5 F FAQs

7. Tag Classifications and Scopes

This section defines the kind of tags, and their scopes.

7.1 Tags Rules

TGRu.1 Each tag is characterized by its scope, i.e. the part of code affected/referred by it. We recognize ITEMS and HEADINGS.

TGRu.2 ITEMS are limited to the line in which they are declared

TGRu.3 HEADINGS are limited to the lines between a begin heading mark and a end heading mark

TGRu.5 Each Tag may carry optional attributes

TGRu.6 Each Tag must begin and end on a single comment line

Rule (**TGRu.6**) is a limitation, which simplifies heavily the preproprocessor.

7.2 Tags Types

Recognised types of tags are:

TY.1 Begin Heading [Attributes]

TY.2 End Heading

TY.3 Item [Attributes]

where [attributes] are the attributes to be added whenever required.

7.3 Scoping Rules

These rules defines how scopes are handled:

SCRu. 1 The scope of one ITEM is limited to the line in which the ITEM is inserted starting from the first character after the ITEM declaration.

SCRu. 2 ITEMS with empty scopes are allowed

SCRu. 3 The scope of one HEADING is limited by its Begin-Mark / End-Mark

SCRu. 4 If the End-Mark is missing the scope is limited to the last line of the Comment Block at which the HEADING belongs.

7.4 Recognised Scopes for Headings and Items

Recognised scopes for headings and items are:

SC.1 `Description`

The description of the code, or routine or subroutine

Must be located at the beginning of the executable block.

No more than one Description for block is allowed.

SC.2 `Assumption`

One hypothesis or assumption which is considered important for the program to operate

SC.3 `History`

Historical annotation, as release number, or some last minute change.

SC.4 `Validation`

A validation step, this includes also peer-reviewing.

SC.5 `Author`

The author or the authors of the code/routine/update.

SC.6 `Warning`

A warning the authors like to introduce in the documentation

SC.7 `Bug`

A warning about a possible bug in the code

SC.8 `BugFixed`

A warning to notice that a bug has been fixed.

SC.9 `Release`

The release number

SC.10 `Refer`

A citation or reference to another document/program

SC.9 `Requirement`

A requirement the code needs in order to be compiled or to operate

SC.10 `Synopsis`

A synopsis (short explanation) to the code use

SC.11 `Environment`

The environment where the code is embedded in order to be

SC.12 `Input`

SC.13 `Output`

SC.13 `Date`

According to (**Re.2**, **BRu.4**) the term *Assumption* is prefered to the term *Hypothesis*, since the latter is more subject to miss-spellings.

8. Tags Syntax

According to (**CH.1**) and to (**BRu.1**), tags are so defined:

!<HEADING, [attribute-name=attribute-value]>

Marks the begin of one heading

!</HEADING>

Marks the end of one heading

!<ITEM, [attribute-name=attribute-value]>

Marks one item

9. Tags Names

Tags names must respect the following rules:

NRu.1 synonyms and abbreviations are not allowed

NRu.2 names must coincided with the recognized items and headings

9.1 List of Valid Tags

```
!<Description Date=yyyy-mm-dd, Release=number, Author=n.name >
!</Description>

!<Assumption Date=yyyy-mm-dd, Release=number, Author=n.name>
!</Assumption>

!<History Date=yyyy-mm-dd, Author=n.name>
!</History>

!<Validation Date=yyyy-mm-dd, Release=number, Author=n.name, Level=string>
!</Validation>

!<Author>
!</Author >

!<Date>

!<Release>

!<Warning Date=yyyy-mm-dd, Author=n.name, weight=string>
!</Warning>

!<Bug Date=yyyy-mm-dd, Author=n.name, weight=string>
!</Bug>

!<BugFixed Date=yyyy-mm-dd, Author=n.name, weight=string>
!</BugFixed>

!<Refer>
!</Refer>

!<Requirement Date=yyyy-mm-dd, Author=n.name, weight=string>
!</Requirement>
```

```
!<Synopsis Date=yyyy-mm-dd, Author=n.name >
!</Synopsis>

!<Environment Date=yyyy-mm-dd, Author=n.name>
!</Environment>

!<Input>
!</Input>

!<Output>
!</Output>
```

10. Development Plan for the Preprocessor

The development plan for the preprocessor follows at least three steps. In the description we describe: the preprocessor task, the requirements to the authors, the development status.

10.1 Release 0

10.1.1 Preprocessor Tasks

Identification of Comment Blocks

Contest Identification.

Bare extraction of the first comment blocks inside recognized contests i.e.: function, subroutine.

Production of a presentation in text format including contest data and comments

10.1.2 Requirements to Authors

Each subroutine or function must contain a descriptive comment block located just after its definition.

10.1.3 Development Status

Development completed May, 21 2000.

10.2 Release 1

10.1.1 Preprocessor Tasks

Identification of Tagged Comment Lines, and Tagged Comment Blocks.

Generation of LaTeX text

10.1.2 Requirements to Authors

Each comment which is liked to enter the documentation must be properly tagged.

10.1.3 Development Status

Syntax completed, June 1, 2000.

Preprocessor Code in preparation.

11. Suggestions for a Deployment Plan for the User and Derived Requirements

This plan is a suggestion to allow a fast application of what described here. The deployment plan is based on the idea that the formatting rules for comments must be applied early in the code development. For this reason it is required that the mark-up language be defined and used before the end of the preprocessor development. To assure this language simplicity is a fundamental asset. Moreover, the adoption of the markup notation must be advantageous itself to the programmer, in order to prepare a better-commented FORTRAN 90 code. The style of the mark-up language not only must not interfere with the readability of the existing code, but also it has to enhance it. I believe that all this is assured by the presently proposed SGML complain markup language.