INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TEXspec:
A Computer Aided Software Engineering Tool for Scientific and Mathematical Applications

By

Stephen E. Oliver

A Practicum Report
Submitted to the Faculty of Graduate Studies, University of Manitoba
in partial fulfillment of the requirements
for the Degree of

Master of Mathematical, Computational and Statistical Sciences

Institute of Industrial Mathematical Sciences
University of Manitoba
Winnipeg, Manitoba

© Stephen E. Oliver, 2001

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-62813-2

Canada

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

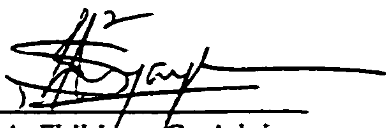
The undersigned certify that they have read, and recommended to the Faculty of Graduate Studies for acceptance, the Master's Practicum entitled:

TEXspec: A Computer Aided Software Engineering Tool for Scientific and Mathematical Applications

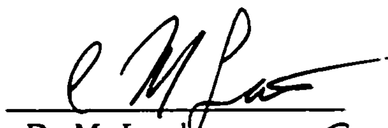
submitted by

Mr. Stephen E. Oliver

in partial fulfillment of the requirements for the degree of
Master of Mathematical, Computational and Statistical Sciences (MMCSS)

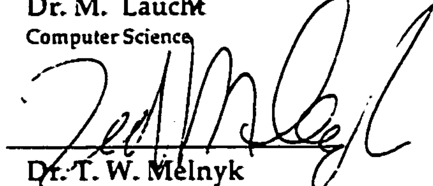


Dr. S. A. Ehikioya, Co-Advisor
Computer Science



Dr. M. Laucht
Computer Science

Mr. T. H Andres, Co-Advisor
AECL

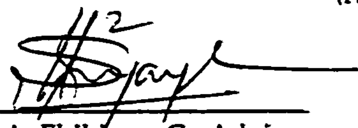


Dr. T. W. Melnyk
AECL

Date of Oral Examination: *July 10, 2001.*

The Practicum Examining Committee certifies that the practicum (and oral examination) is:

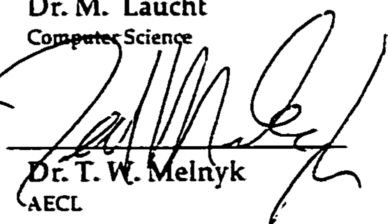
Approved
(Approved or Not Approved)



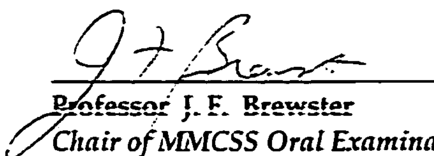
Dr. S. A. Ehikioya, Co-Advisor
Computer Science

Dr. M. Laucht
Computer Science

Mr. T. H Andres, Co-Advisor
AECL



Dr. T. W. Melnyk
AECL



Professor J. F. Brewster
Chair of MMCSS Oral Examination

(The signature of the Chair does not necessarily signify that the Chair has read the entire thesis.)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES
*****
COPYRIGHT PERMISSION

TEXspec: A COMPUTER AIDED SOFTWARE ENGINEERING TOOL FOR SCIENTIFIC AND MATHEMATICAL APPLICATIONS

BY

STEPHEN E. OLIVER

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

of

MASTER OF MATHEMATICAL, COMPUTATIONAL AND STATISTICAL SCIENCES

STEPHEN E. OLIVER © 2001

Permission has been granted to the Library of the University of Manitoba to lend or sell copies of this thesis/practicum, to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film, and to University Microfilms Inc. to publish an abstract of this thesis/practicum.

This reproduction or copy of this thesis has been made available by authority of the copyright owner solely for the purpose of private study and research, and may only be reproduced and copied as permitted by copyright laws or with express written authorization from the copyright owner.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Abstract

This report discusses the development of the TEXspec Computer Aided Software Engineering (CASE) tool, which assists with the development and documentation of software in an environment where software quality is closely monitored, perhaps by independent regulators. The tool can assist in the development of a broad range of software, but is targeted at the software that implements mathematical models.

TEXspec generates requirements specifications, design specifications and compilable code in a structured form while ensuring consistency between products.

The original application of TEXspec was to assist developers of software modeling a repository for Canada's high level nuclear waste to achive compliance with a quality assurance standard specified by goverment regulators.

This report details the form of documentation products produced by TEXspec and all required inputs. It discusses the processing that TEXspec uses to convert input into final products. The method of ensuring consistency between products is reviewed. Instruction is provided for operating TEXspec using a graphical user interface. The significance of the work is discussed and directions for future development are suggested.

Some of the requirements of TEXspec are continuing to evolve. As such, the development is of necessity of a prototype, or spiral model, nature. This report acts as a status report on the development of TEXspec and provides a reference for both users and programmers.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Acknowledgements

The author acknowledges the guidance, patience and funding provided by Ontario Power Generation supporting the development of the TEXspec CASE tool. Paul Gierszewski has acted as project officer providing valuable feedback and original ideas.

Many TEXspec documentation products have been reviewed by Ted Melnyk and Chuck Kitson. Their feedback provided valuable input to the development process.

Many of the innovative concepts implemented by the TEXspec system, including the separation of content from format of documentation, originate with Terry Andres, who co-supervised TEXspec development. Some of these concepts were researched initially by Dennis LeNeveu, whose Fortran program TEXdef inspired TEXspec.

Dr. Sylvanus Ehikioya served as co-supervisor at the University of Manitoba. He has responded to an unknown path to be travelled under tight time constraints in an effective and helpful manner.

The patience and support of Atomic Energy of Canada Ltd. management, in the persons of Alf Wikjord and Peter Sargent has been crucial to the development of TEXspec. The unusual employment situation as the research site in Pinawa is wound down has been a challenge to everyone involved.

The administration at the University of Manitoba and the Institute of Industrial Mathematical Sciences (IIMS) have reacted to the peculiar circumstances in Pinawa in a highly flexible and patient manner. Professor John Brewster directs the IIMS and has led the way.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Contents

1 Introduction 1

1.1 Problem Definition 1

1.1.1 Commercial Tools 2

1.1.2 Yourdon/DeMarco Methodology 2

1.1.3 Design Specifications 3

1.1.4 Experience with Software Quality Control 3

1.2 Objective of the Study 4

1.3 Significance of the Study 5

1.4 Limitations 5

1.5 Related Work 5

1.6 Notations 6

1.7 Organization of the Report 7

2 Specification and Design 8

2.1 The TEXspec CASE Tool 8

2.1.1 Requirements Specification for TEXspec 8

2.1.2 Architecture of TEXspec 9

2.1.3 Design of TEXspec 10

2.1.4 Implementation Language 13

2.2 Application Shared Components 14

2.2.1 Requirements Data Dictionary 14

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.2.2 Design Data Dictionary 15

2.2.3 Dictionary Listing 17

2.2.4 Equations 18

2.3 Application Composite Components 20

2.3.1 Data Flow Diagrams 20

2.3.2 Process Specifications (Mini-Specs) 25

2.3.3 Design Specifications 27

2.3.4 Structure Charts 33

2.3.5 Manuals 36

3 Graphical User Interface **37**

3.1 Architecture 37

3.2 Configuration and the Search List 38

3.3 Requirements Data Dictionary 39

3.4 Design Data Dictionary 41

3.5 Dictionary Listing 41

3.6 Process Specifications (Mini-specs) 42

3.7 Data Flow Diagrams 43

3.8 Design Specifications 45

3.9 Structure Charts 48

3.10 Manuals and Equations 49

3.11 Java ↔ Perl Interface 49

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4 Conclusions 52

4.1 Maintenance and Future Development . . . 52

A Sample Data Flow Diagram 54

B Sample Design Specification 56

B.1 Output . . . 56

B.2 Input . . . 63

C Sample PERL Script 68

D Sample Java Module (GUI) 77

E Installation 83

E.1 Prerequisite Software . . . 83

E.1.1 Perl . . . 83

E.1.2 TEX and LATEX . . . 83

E.1.3 Noweb . . . 83

E.1.4 JAVA Runtime Environment . . . 84

E.2 TEXspec Specific Installation . . . 84

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1 Introduction

The Deep Geologic Repository Technology Program (DGRTP), administered by Ontario Power Generation (OPG), is charged with developing technology to deal with Canada's high level nuclear waste. Atomic Energy of Canada Ltd (AECL), as a major contractor to the DGRTP, has accumulated considerable experience developing computer programs to model a deep geologic repository for used fuel [8, 9]. These programs require software of demonstrably high quality to support results presented to the Canadian Nuclear Safety Commission (CNSC) and the public.

In 1999 the Canadian Standards Association (CSA) adopted a standard (CSA N286.7) [4] for the development of nuclear safety related computer programs, a scope that included many DGRTP models. While the software development process used previously was considered robust, it required refinement in order to achieve compliance with the standard.

The TEXspec project seeks to address the issue of compliance with the CSA standard in a general way. The objective is to develop a tool to support a compliant software development procedure while imposing a minimum of additional overhead. The tool must support the use of diagrams and/or graphics and mathematical notation. While TEXspec is optimised to meet the particular requirements associated with modeling the disposal of Canada's nuclear fuel waste, it is hoped that TEXspec will find more general usage.

1.1 Problem Definition

The principles of the CSA N286 standards require that

- All software products be subject to a review by qualified staff,
- Genealogy of products be preserved and
- Ownership of products be clearly defined.

To adhere to these principles, products must be clearly delineated and controlled. Where multiple products share common components, this can become difficult to achieve. For example, the same mathematical equation might appear in the theory manual, requirements specification and design documentation. The equation may have been developed by one author, the requirement specification by another and the design documentation by someone else. Tracking this relationship requires that the equation be maintained separately from the products that reference it.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The requirement to document the INROC [16, 17] computer program in a CSA N286.7 compliant manner has led to the development of the TEXspec Computer Aided Software Engineering (CASE) tool. TEXspec implements support for software development methodologies used in the development of INROC documentation, including requirements specification, design description and manuals. TEXspec is designed to allow for enhancements handling other software development products outlined in CSA N286.7, and may be expanded to include other methodologies (including object oriented approaches). It is intended to be sufficiently flexible to permit enhancements to include other phases of the software development life cycle.

1.1.1 Commercial Tools

Several commercial CASE tools have been examined, including DecDesign[6], Graphical Designer[1], Software Through Pictures[11, 27], and Teamwork[3]. Each of the examined tools was found to be deficient in one or more critical areas:

- Lack of support for scientific and mathematical notations. The nature of the models demands that mathematical notations (e.g., $A_i(t) = \int_0^t [F_i^{IN}(\tau)]d\tau$) be permitted in specifications, including diagrams.
- Insufficient accountability. The principle of ownership and accountability for products is not strictly enforced. While a record of who updated products is often kept, the process control is often inadequate. For example, anyone who shares a data dictionary might be permitted to update any entry without regard to individual ownership of particular entries.
- Assembling large products from smaller components is not adequately supported. In the experience accumulated with the INROC program and it's predecessors [18], many software defects were found to be the result of transcription errors between products.
- Insufficient consistency checking between products.

1.1.2 Yourdon/DeMarco Methodology

TEXspec is based on the Yourdon/DeMarco structured analysis methodology [5, 30] for software development.

Many models have, to date, been described using a modified Yourdon/DeMarco methodology [15]. Although OO methods would perhaps be more appropriate for some models, priority is given to the more common structured analysis methodology. Products associated with this methodology are:

- Data flow diagrams (DFDs),

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- Process descriptions (mini-specs),
- Structure charts,
- Module design descriptions (Design Specifications), and
- Data dictionary listings.

Data Flow Diagram (DFD)s and Mini-specs comprise the requirements specification, while Structure Charts and Design Specifications specify the design. Data dictionary listings may be separated into requirements and design, or combined into a single product.

Although Object Oriented (OO) analysis and design is appropriate for many software applications, there are still applications for procedure/flow based software. In particular, some models which are basically linear in structure, including many scientific models, are best described using non-OO techniques.

1.1.3 Design Specifications

Module design descriptions form an engineering blueprint for code [20, 24]. A programmer serves analogously to a construction tradesman, who implements the design. This philosophy has resulted in design documentation which closely parallels the final code or pseudo-code [19]. The design specification and compilable code can be sufficiently similar that creating and maintaining both can be an inefficient use of resources. The two must also be closely monitored to ensure that they are synchronized. The duplication of effort must be reduced and the chance of incousistency between products must be addressed.

1.1.4 Experience with Software Quality Control

Many models and associated programs are most clearly specified using mathematical abstractions. While it is possible to express $A_i(t) = \int_0^t \left[F_i^{IN}(\tau) + \lambda_p A_p(\tau) - F_i^{OUT}(\tau)\right] \exp\left(-\lambda_i (t-\tau)\right) d\tau$ in plain english text, it is much more convenient and expressive to utilize the mathematical notation. It is therefore imperative to support the use of this kind of notation in software development products, including requirements and design specifications, as well as manuals and other documentation. The transcription of mathematical notation has proven to be error-prone [18], and must be minimized.

The relationship between Requirements and Design leads to other common items between their specifications, as they are different expressions of the same system. For example, a requirement specification might specify a 'density', denoted as 'ρ', with physical units '$\frac{Kg}{m^3}$'; the design might then specify a real variable 'rho' with the same attributes and description. Many commercial CASE tools maintain a common Data Dictionary to

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

handle some of this overlap between Requirements and Design. This approach, unfortunately, can compromise the principle of responsibility for products. In an environment where the genealogy of products must be known, sharing a common Data Dictionary must be carefully controlled, or multiple data dictionaries can be used. In the past, DGRTP has used multiple data dictionaries, but this has led to transcription related defects, and a propensity for dictionaries to fall out of synchronization. In addition, any attempt to merge dictionaries has had to resolve duplicate entries.

For the models implemented for a single environmental assessment, AECL invested over $1 million to verify software by unit testing [18]. The result was far from encouraging. The contractor (Science Applications International Corporation) found many defects in the documentation and transcription between products, but nothing that could materially affect results. An embarrassing number of defects was reported.

The format of software documentation may have a much shorter lifetime than the software itself. Documentation for some long lived Fortran modules have been published in Mass-11 (a word processor that is no longer supported), Wordperfect, MS-Word, and others, all with differing styles. Software supporting a single study has been published in several different formats. This experience suggests that the content of documentation should be separated from the presentation; the information should be collected independently and assembled according to the current format in use at the time of final publication.

Attributing ownership and responsibility for products is a basic principle of the CSA N286 standards. In order to effectively reuse common information, while remaining faithful to this principle, it is helpful to collect, in very small pieces, information used to assemble software products. The dependencies between products and components are easier to manage if the shared information is not contained in large packages. Keeping the granularity of components very fine also allows ownership to be tracked, without assigning ownership to more than one individual.

Verification of consistency between software products has been a costly and error prone procedure [18]. The number of products has been high, and verification has not been sufficiently automated. If a high granularity of components is desired, then automation is clearly required.

1.2 Objective of the Study

The objective of this study is to develop a tool to assist in the development of software and associated documentation compliant with the CSA N286.7 standard [4]. The tool must address some of the deficiencies observed in commercial CASE tools which make those tools difficult to deploy for the development of software that implements mathematical models.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1.3 Significance of the Study

The TEXspec tool described in this report is a stepping stone to compliance with the CSA standard for the development of nuclear safety related computer programs. This compliance is expected to be required to support future licence applications to the CNSC.

The tool offers a viable CASE capability for computer programs which are best specified with intensive use of mathematical notation.

1.4 Limitations

TEXspec is a prototype. Many features in both the underlying technology and in the usability remain to be addressed. Some of the requirements of TEXspec are continuing to evolve. As such, the initial development is of necessity a prototype, developed using a spiral model. This report is a snapshot of the current state of TEXspec development.

Currently, TEXspec can only generate design documents for Fortran-77 code. In the next stage of development, this will be expanded to include some Fortran-90 extensions, including 'modules'. In the future, this is expected to expand further to include other languages.

The Graphical User Interface (GUI) is in an early stage of development. The editors are not sophisticated, with no search-and-replace capability. Development of graphical products is based on non-graphical editors and no preview capability has been implemented. The system is usable and effective, but there is still room for development and further research.

The system has not yet been integrated with a secure configuration management system. Effective sharing of data and meaningful software audit capabilities await this development. This could be expanded to integrate with a change control system.

The data processing and the GUI are currently both run on the same machine. A client/server model might be an important development in the future, assigning the compute and I/O intensive processing to a server.

1.5 Related Work

Aside from commercial CASE tools, the work of Wieringa [29] is notable. The Toolkit for Conceptual Modeling (TCM) is implemented to support the Toolkit for Requirements And Design Engineering (TRADE). This tool generates several different diagram types and even performs some consistency checking of data flow diagrams. Unfortunately, the system runs only on (UNIX)/X-windows, does not adequately handle math-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ematical notation and does not integrate well with an acceptable Data Dictionary. Even so, a modification of TCM may provide a useful interface for TEXspec.

Also a possibility for a drawing interface, LATEXcad [14] provides a GUI capable of handling math-centric LATEX [13] labeling, but would require some modification. Like TCM, LATEXcad is also a single platform tool, running under Microsoft Windows (MS-Win).

Another CASE tool which uses Java as a front end is the ArgoUML [22] design tool. ArgoUML is exclusively an object oriented tool. The interface is mature and allows the user to interact directly with diagram components. Since ArgoUML is an 'open source' project, the code is available.

TEXspec is built on the LATEX foundation with a pair of significant extensions. The Noweb [21] system for Literate Programming is used to separate module Design Specifications and compilable code. Graphics extensions suitable for the generation of diagrams are provided by the xypic [23] package.

1.6 Notations

TEXspec input files are ASCII files, organized as 'field: value' pairs. When specifying the content of these files, the following notation is used:

FieldName: *description of value*

The 'description of value' is contained within delimiters as follows:

- $\rhd$*required field, may appear only once*$\lhd$
- $\unrhd$*required field, may appear more than once*$\unlhd$
- $\succ$*optional field, may appear only once*$\prec$
- $\succeq$*optional field, may appear more than once*$\preceq$

Where sets of 'field: value' pairs are grouped, the group is named in bold type within brackets for later expansion. The same delimiters are used. For example: $\unrhd$**[group name]**$\unlhd$ specifies a group of fields which is required and may appear more than once.

These delimiters are used rather than the more conventional bracket/brace notations to allow for non-ambiguous delimitation of TEX content, which uses brackets and braces.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1.7 Organization of the Report

The remainder of this report is organized as follows. The underlying technology of TEXspec is detailed in Chapter 2, including the requirements, design, and various file formats. The user interface for TEXspec was implemented separately from the underlying processing and is detailed in Chapter 3. Chapter 4 offers some concluding remarks and suggests some directions for further development. Appendices contain sample code listings and examples of the longer TEXspec inputs and products that are not fully shown in the text. The final appendix provides instruction for installing TEXspec.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2 Specification and Design

2.1 The TEXspec CASE Tool

2.1.1 Requirements Specification for TEXspec

The TEXspec application is based on the following requirements:

- Assemble user inputs to generate consistent publication quality Data Flow Diagrams (DFDs) and Process Specifications in a modified Yourdon/DeMarco format. This includes support for 'leveled' diagrams [30], which allow a 'parent' process to be decomposed in a 'child' diagram.
- Permit the use of composite data flows on DFDs. Break composites as required for a child DFD or Process Specification (Mini-spec).
- Ensure consistency between the data flows shown on the DFDs and Mini-spec.
- Generate Structure Charts and Design Specifications.
- Ensure consistency between the flows on the Structure Charts and the Design Specifications.
- Ensure consistency between the Design Specifications and executable code.
- Permit the use of mathematical notation in all products.
- Allow sharing of mathematical formulae between products.
- Permit ownership of products to be tracked and reported.
- Allow components under development to reference other components from a variety of sources. Stable libraries of components should be supported as a default, which new components under development supercede.
- Support the use of Fortran as a target implementation language.
- A user interface must be provided that allows users to interact with TEXspec in an intuitive way. The interface should require minimal training before a user becomes proficient.
- Information to be processed by TEXspec is assumed to have a long lifetime, perhaps exceeding that of TEXspec itself. The information must therefore be stored in a format suitable for later processing by other programs, or perhaps the human eye.
- A 'batch processing' option must be supported that can capture and log processing details.
- Learning curves for both users and implementers should not be excessive. Maintenance expertise should not be difficult to recruit or train.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- Coding languages, libraries and tools should be freely available.
- The system must be portable between computing platforms. Although the desktop environment is dominated by MS-WIN, being locked to any single system restricts deployment options and reduces the number of potential users. Also, if the application were divided into client and server portions, the server environment is likely to be more heterogeneous.

Some preferred attributes of TeXspec are not required in an absolute sense:

1. The application should run in 'reasonable' time on common desktop computers. This is a difficult requirement to quantify, since the term 'reasonable' is subject to interpretation and what is common on the desktop differs in time and location. Even so, it can be said that a responsive application is preferred over the alternative and that some design effort can reasonably be applied to achieving the best possible performance.
2. The implementation should be maintainable. Code implemented in an uncommon language is more difficult to maintain, as programmers are less likely to be familiar with it.
3. There should be a migration path to allow a gradual transition from existing methods. The 'cold turkey' implementation of new tools is rarely well received. A pilot project style of implementation is preferred, as it allows operational difficulties to be dealt with before a large commitment is made.

2.1.2 Architecture of TeXspec

TeXspec's GUI is discussed in Chapter 3, which captures interactions with the user. Most of this interaction consists of displaying and manipulating 'component' files, which form the inputs for the TeXspec scripts that select components and assemble them into products. These products are primarily LaTeX [13] or Noweb [21] input files, which can be post-processed to produce output suitable for viewing, printing, or compiling. While these outputs may be viewed as being intermediate, they are intended to be retained, as TeXspec places commentary in them to record the details of TeXspec processing.

While the GUI is a convenient way to construct components and initiate processing, it can be bypassed if required. The components can be generated by any means that can generate an ASCII output file, including a text editor. More importantly, the processing can be controlled by any means that can initiate a process, with no requirement for interaction with a GUI. When processing many components, or when a log of processing is required, this 'batch' style processing is a useful alternative.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Neither the TEXspec scripts, nor the GUI can display or print the products. Figure 2.1 indicates that an intermediate script, which is intended to be edited by the user, initiates TEXspec to produce the product files, then controls post-processing as appropriate. This flexibility allows the user to integrate TEXspec into existing procedures. For example, if a static code analyzer such as Floppy [2] is in use, it can be run automatically on code as it is generated. Interaction with a version control system might be desired, or the user may even wish to compile code as it is generated. Alternatively, processing that is not needed can be removed, such as removing documentation generation (including LATEX processing) until the code is stable.

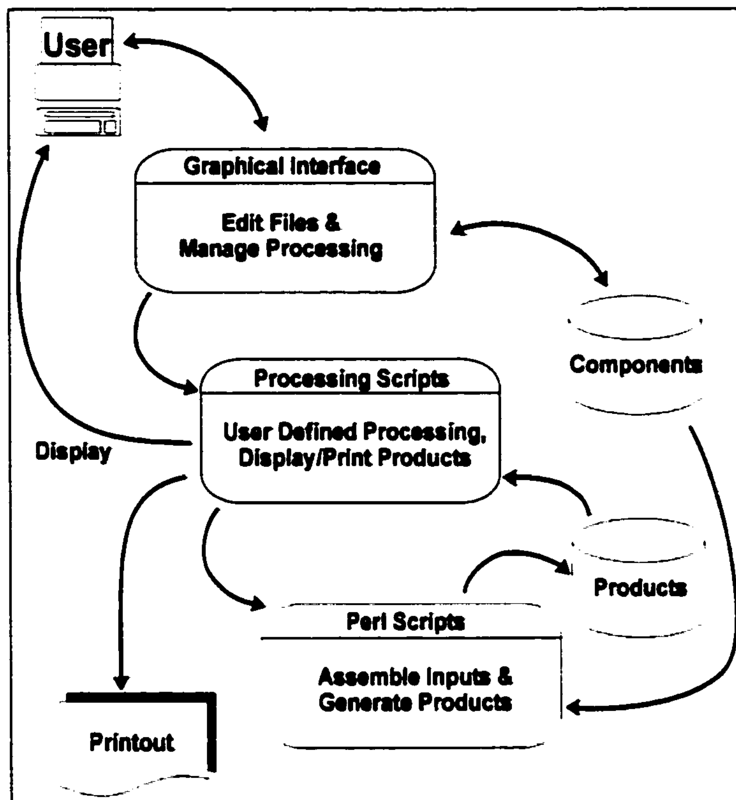


Figure 2.1: Schematic view of the TEXspec architecture

2.1.3 Design of TEXspec

All input files for TEXspec are human readable. That is, they are in ASCII format, organized in '*Label: value*' pairs, which is intended to ease visual interpretation. The input files can be created using a standard text editor and reviewed easily due to the intuitive syntax, without the overhead of an elaborate interface.

A more sophisticated interface for handling TEXspec files, which can be large in number, has been developed. Still, the ASCII format files can be edited or read by readily available tools and do not require TEXspec programs to interpret.

In order to support sharing of equations and data definitions, while tracking ownership and responsibility for content, TEXspec supports a fine granularity of components. Components are tracked independently by placing each in a unique file which is mapped by the file name to the name of the component and by the file name 'extension' (in the tradition of MS-DOS or CP/M) to the type of component.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TEXspec components, with associated file name extensions, are:

- Requirements Data Dictionary entries (**.rdd**: Section 2.2.1),
- Design Data Dictionary entries (**.ddd**: Section 2.2.2),
- Equations (**.teq**: Section 2.2.4),
- Process Specifications (**.ms**: Section 2.3.2),
- Design Specifications (**.ds**: Section 2.3.3),
- Structure Charts (**.sc**: Section 2.3.4), and
- Manuals (**.tex**: Section 2.3.5).

In order to share components between individuals and projects, the location of these files is flexible. A 'search list' provides a list of directories to be searched sequentially for component files. This is often referred to as a 'PATH'. A different 'search list' can be provided for each component type in a file named '.TeXspecrc', as shown in Figure 2.2.

```
.ms:  x:\ufdp_ea04\config\design\ver0003\work\
      x:\ufdp_ea04\config\ReqSpec\ver0002\TeXspec_format\MiniSpec\
      x:\ufdp_ea04\config\TeXspec\ver0005\inroc\MiniSpec\
.teq: x:\ufdp_ea04\config\design\ver0003\Equations\
      x:\ufdp_ea04\config\ReqSpec\ver0002\TeXspec_format\Equations\
      x:\ufdp_ea04\config\TheorMan\ver0003\Equations\
.ds:  x:\ufdp_ea04\config\design\ver0003\DesignSpec\
      w:\aba_shr\ipa\int\cc402\Cng02\vtdmen\
      x:\ufdp_ea04\config\design\ver0002\designspec\
      x:\ufdp_ea04\config\TeXspec\ver0005\inroc\DesignSpec\
.inc: x:\ufdp_ea04\config\design\ver0003\work\
      w:\aba_shr\ipa\int\cc402\Cng02\ccccde\inc\
      w:\aba_shr\ipa\int\cc402\Cng02\btcode\inc\
      w:\aba_shr\ipa\int\cc402\Cng02\dcode\inc\
.ddd: x:\ufdp_ea04\config\design\ver0003\DesignDD\
      W:\Aba_shr\IPA\Imp\CC402\Vlt\V0210\dictionary\
      W:\Aba_shr\IPA\Imp\CC402\Vlt\V0205\dictionary\cc4\
.rdd: x:\ufdp_ea04\config\design\ver0003\work\
      x:\ufdp_ea04\config\ReqSpec\ver0002\TeXspec_format\RequirementsDD\
.sc:  x:\ufdp_ea04\config\design\ver0003\structurechart\
      x:\ufdp_ea04\config\design\ver0002\structurechart\
.dfd: x:\ufdp_ea04\config\design\ver0003\work\
      x:\ufdp_ea04\config\ReqSpec\ver0002\TeXspec_format\DataFlowDiagram\
      x:\ufdp_ea04\config\TeXspec\ver0005\inroc\DataFlowDiagram\
```

Figure 2.2: Example .TeXspecrc file, specifying search lists by component type

This mechanism is intended to be supplemented in future versions of TEXspec. It is intended that components should be 'installed' in a configuration management system for future reference. Once installed, dependencies between components would be monitored to ensure that the creator of a product is not surprised by a change in his product caused by a change in a referenced component (a 'sub-component') for which someone else is responsible. For the moment, however, the directory search list meets the requirement, allowing components to reference other components from a variety of sources. This mechanism will continue to be supported as a 'working area'. That is, the 'search list' will be set up to specify that working directories be searched first, followed by libraries under the configuration management system.

Component files are each assigned a version of format '**NNA**', where '**NN**' is a two digit integer indicating the installation number and '**A**' is a single character 'draft letter' indicating changed versions between installations. This corresponds to the scheme used at AECL to configure software versions manually.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

To keep track of the components used to assemble a TEXspec product, the LaTeX input files generated by TEXspec modules contain commentary that identifies all referenced components and the version of the TEXspec module that assembled them. A date-time stamp is included and is also placed on the generated product (in the upper left corner) to uniquely associate the LaTeX file with the associated product. By retaining the LaTeX file, it is possible to audit the content of any product. This is demonstrated in Appendix B.

```
% ===> this file was generated automatically by noweave --- better not edit it
%...  output generated by X:\Ufdp_sa04\Config\TEXSPEC\Ver0006\bin\designSpec.pl on Thu Mar
15 13:29:58 2001
%...                  component  version
%...      X:\Ufdp_sa04\Config\TEXSPEC\Ver0006\bin\designSpec.pl  02F
%...                 designspec  04R
%...      W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/oo4/elem.ddd  01B
%...      W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/oo4/maxsec.ddd  01B
%...      W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/oo4/modnam.ddd  01B
%...      W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/oo4/msg.ddd  01B
%...      W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/oo4/mxchan.ddd  01B
%...      W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/oo4/neenfs.ddd  01C
%...      W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/oo4/numreg.ddd  01C
%...      W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/oo4/numsec.ddd  01C
%...      W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/oo4/regsec.ddd  01C
%...      W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/oo4/sec.ddd  01B
%...      W:/Mba_shr/IPA/Imp/CC402/V1t/V0205/dictionary/oo4/reg.ddd  01B
%...      W:/Mba_shr/IPA/Imp/CC402/V1t/V0205/dictionary/ev3/nelmnt.ddd  01A
```

Figure 2.3: Top of a LaTeX file generated by TEXspec, showing versions of components

Each formatted product has an additional configured component. The LaTeX 'class' .cls file used to specify the format of the product (in particular the page header) must be installed into the LaTeX system that TEXspec will use to produce products. At the upper right corner of each generated product, the version of the 'class' file is printed.

As discussed in Section 1.1.4, minimal formatting information is stored with the TEXspec components. Formatting is a function of the processing of the components. The hope is that as documentation formats evolve, the critical content of the components shall not be rendered obsolete.

Many scientific models benefit greatly from the ability to incorporate mathematical notation in their specification. One of the requirements of TEXspec is to support such notation in all products. To be compatible with the 'human readable' design decision, a notation is required that stores such information in ASCII format. This information must then be translated into a flexible presentation format.

Since the LaTeX system is already mature and offers leverage toward meeting the requirements stated in section 2.1.1, TEXspec produces documentation via LaTeX.

To keep code synchronized with associated documentation, a literate programming [10, 12] methodology is ideal. A single file is used to generate both a Design Specification and compilable code. Fortunately, several systems already exist to support this method in a LaTeX environment. The Noweb system was selected because it is not sensitive to programming language, allowing TEXspec to evolve (in the future) to handle languages other than Fortran. An additional benefit of adopting Noweb is that much of the syntax for the Design Specification file (Section 2.3.3) is defined in Noweb, relieving TEXspec of the requirement to define such syntax.

Components are processed by TEXspec modules according to the flow specified in Figure 2.4. Users of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TEXspec provide content in components in the 'Shared Components' and 'Product Definition' categories, which together comprise the TEXspec inputs. The 'Shared Components' are intended primarily to be referenced by the 'Product Definition' files.

The inputs are processed into 'Products' by TEXspec. These products are listed in Figure 2.4 and correspond to the products defined by Yourdon [30] and Page-Jones [19], plus the Fortran code. Note that the output from TEXspec is not publishable (or compilable), but must be post-processed by LATEX and/or Noweb to produce final products.

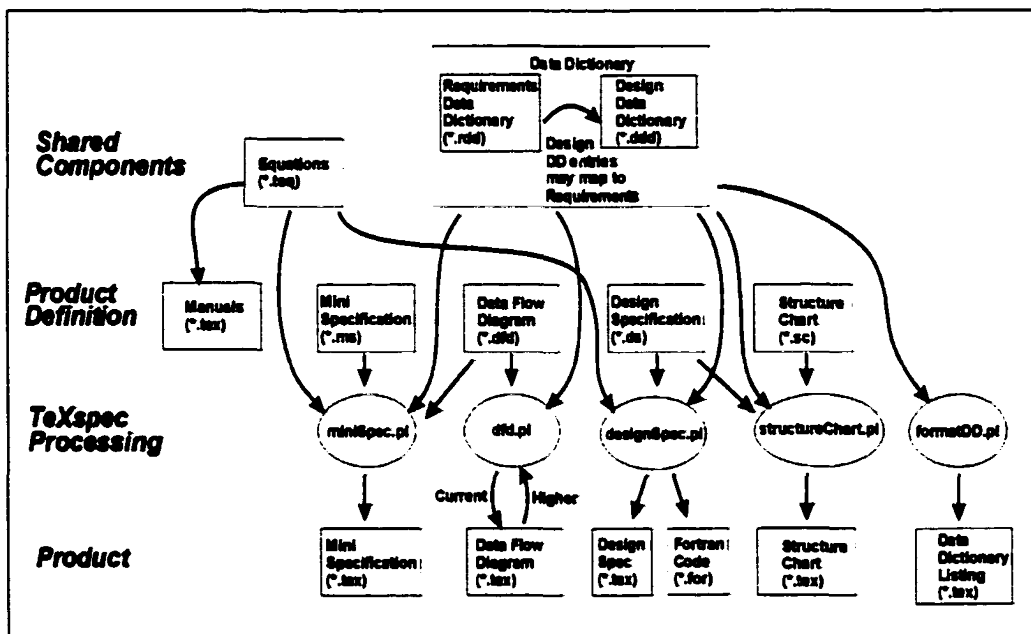


Figure 2.4: TEXspec flow, indicating the major scripts, with the relationship of inputs and outputs.

2.1.4 Implementation Language

The main TEXspec processing is performed by modules which have been implemented in Practical Extraction and Report Language (PERL) [28]. The selection of PERL was based on a number of factors:

- it does not conflict with the requirements stated in section 2.1.1 and
- it has sufficient flexibility to act as a general purpose language.

For the purpose of developing a user interface, PERL is not as good a fit. Although a simple GUI can be implemented in PERL using existing libraries, the required GUI is not sufficiently simple. The TEXspec GUI is implemented in Java. The selection of Java was based on a number of factors:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- it does not conflict with the requirements stated in section 2.1.1 and
- the associated 'Swing' [7] library can be used to develop a sophisticated GUI.

2.2 Application Shared Components

Some TeXspec input files are intended to be shared. That is, they are referenced by other input files (see Section 2.3). This relationship is illustrated in Figure 2.4.

2.2.1 Requirements Data Dictionary

Although a Data Dictionary listing (Section 2.2.3) is available as a stand-alone product, the primary use of Data Dictionary entries is to be referenced by other components such as Mini-specs or Data Flow Diagrams.

Each entry is contained in a file *name.rdd* where *name* is the argument in the 'Name:' line.

Syntax of Requirements Data Dictionary (.rdd) file	
Name:	▷*short name in ascii format - minimal for unique identification*◁
LabelName:	▷*name to appear in diagrams (if different)*◁
MathName:	≻*name using mathematical notation entered in LaTeX format*≺
LongName:	▷*descriptive name in ascii format - up to a sentence*◁
Version:	▷*version number for tracking history - appears on listings*◁
Project:	▷*project identification*◁
Subproject:	▷*sub-project identification*◁
Author:	▷*author's full name*◁
Date:	▷*date that the entry was written*◁
Implementer:	▷*full name of person who input this entry into the system*◁
ImplementDate:	▷*date that the entry was entered into TeXspec*◁
Reviewer:	≻*full name of reviewer*≺
ReviewDate:	≻*date of review*≺
CompositeOf:	≻*comma delineated list of other Requirements Data Dictionary entry 'Name's if the entry is a composite of other entries*≺
PhysicalUnits:	≻*SI units enclosed within square brackets*≺
DataType:	≻*descriptive data type e.g., 'integer'*≺
Dimension:	≻*dimensioning information*≺
Description:	▷*full description - up to a paragraph*◁

An example of a Requirements Data Dictionary entry is shown in Figure 2.5. The example is a 'composite' entry, composed of several other entries. Note the optional 'LabelName' field is used to produce labels on Data Flow Diagrams which differ from the 'Name'. TeXspec requires that *name.rdd* be a valid file name, but the dash in the 'sp-Alpha' might create an illegal *name.rdd*. Using 'LabelName' prevents the potentially offensive syntax from appearing in the 'Name' field, but diagram labels can contain the dash.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
Name:          spalph
LabelName:     sp-ALPHA
MathName:      sampled_\alpha
LongName:      Sampled parameters for alpha radiolysis
Version:       01B
Project:       CC4
Submodel:      inroc
Author:        S.E. Oliver
Date:          Oct 4, 2000
Implementer:   S.E. Oliver
ImplementDate: Oct 4, 2000

CompositeOf:   AALPHA,ALFCOF,ALPHDO,ALPHTI,BALPHA,CALPHA,DALPHA,EALPHA,FALPHA,NOALPH,STDNOA
Description:   Sampled parameters for calculation of fuel corrosion rate
               due to alpha radiolysis of water.
```

Figure 2.5: Example Requirements Data Dictionary file.

2.2.2 Design Data Dictionary

Much like Requirements Data Dictionary entries, Design Data Dictionary entries may appear in a Data Dictionary listing (Section 2.2.3), but their primary use is to be referenced by other components such as Design Specifications or Structure Charts.

A Design Data Dictionary entry may reference a Requirements Data Dictionary entry via the 'Requirements' field. If this is done, any missing fields in the Design Data Dictionary entry will default to the value found in the specified Requirements Data Dictionary entry. This is particularly useful to avoid transcription and synchronization problems with the 'MathName' and 'Description'. Fields which are specified in the Design Data Dictionary supercede any inherited defaults.

Currently, the 'CompositeOf' field is supported in the Requirements Data Dictionary only, and is unsupported in Design. As TEXspec evolves to support programming languages with more advanced data structures than Fortran-77, this will probably change.

Each entry is contained in a file *name.ddd* where *name* is the argument in the 'Name' line.

The dictionary can specify a constant value, or a 'condition' may be placed on the value. A 'condition' is interpreted as a a 'precondition' to modules for which the variable is used as input and a 'postcondition' to modules assigning a value to the variable. This is usually a physical limitation on the range of valid values.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Syntax of Design Data Dictionary (.ddd) file	
Name:	▷*short name in ascii format - minimal for unique identification*◁
LabelName:	≻*name to appear in diagrams (if different)*≺
MathName:	≻*name using mathematical notation entered in LaTeX format*≺
LongName:	▷*descriptive name in ascii format - up to a sentence*◁
Version:	▷*version number for tracking history - appears on listings*◁
Project:	▷*project identification*◁
Subproject:	▷*sub-project identification*◁
Author:	▷*author's full name*◁
Date:	▷*date that the entry was written*◁
Implementer:	▷*full name of person who input this entry into the system*◁
ImplementDate:	▷*date that the entry was entered into TeXspec*◁
Reviewer:	≻*full name of reviewer*≺
ReviewDate:	≻*date of review*≺
Requirements:	≻*'Name' of corresponding Requirements Data Dictionary entry*≺
PhysicalUnits:	≻*SI units enclosed within square brackets*≺
DataType:	▷*data type suitable for program design in target language*◁
Dimension:	▷*dimensioning information*◁
File:	≻*for shared (COMMON) variables - file to contain definition*≺
Common:	≻*name of Fortran COMMON block to contain data*≺
Value:	≻*value if constant*≺
Condition:	≻*limitation on value, used as 'pre-' or 'post-condition'*≺
Description:	▷*full description - up to a paragraph*◁

In cases where the variable can be directly mapped to a Requirements Data Dictionary entry, the 'Requirements' field can be used to specify the mapping, and any common fields are inherited from the Requirements Data Dictionary (unless overridden here).

An example of a Requirements Data Dictionary entry is shown in Figure 2.6.

```
Name:          AALPHA
MathName:      a_\alpha
LongName:      Fit coefficient a for alpha radiolysis
Version:       01C
Project:       CC4
Submodel:      INROC
Author:        S.E. Oliver
Date:          October 25, 2000
Implementer:   S.E. Oliver
ImplementDate: October 25, 2000

DataType:      double
Dimension:     scalar
File:          SPALPH.INC
Common:        SPALPH
Description:   Empirical fit coefficient `$a$' for alpha radiolysis,
               used in the calculation of of the degradation rate per
```

Figure 2.6: Example Design Data Dictionary file.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.2.3 Dictionary Listing

Data dictionary entries are incorporated into other products, but can also be assembled into a stand-alone product. TEXspec provides a module 'formatDD.pl' which provides listings of Data Dictionary entries. It can also provide a cross-reference, showing the Process Specifications (Section 2.3.2) and Design Specifications (Section 2.3.3) in which they occur (optionally colour coded to indicate the direction of flow).

A sufficient number of fields exists to make a complete listing impractical to tabulate on a single page. To ease this problem, formatting on "legal" size sheets is supported, and the default orientation is landscape. Even so, the user is obliged to select a subset of the available fields for listing. The user may also specify the width of particular fields. Usage is shown in Figure 2.7 and a sample output is shown in Figure 2.8.

The 'width' fields are specified in LaTeX-style measures including units (e.g., '0.5in'). The 'xref' option produces a cross reference column and the 'flow' sub-option causes the cross reference to be colour coded to indicate direction of flow.

```
Usage:  formatDD.pl
          R|D|M          #... "R"equirement, "D"esign, or "M"erged
          [lines=nn]     #... est max lines per page
          [chars=nn]     #... est chars/inch
          [caps=nn]      #... est CAPS/inch
          [portrait]
          [description[:width]]
          [xref[:width][:flow]]
          [longname[:width]]
          [mathname[:width]]
          [version[:width]]
          [project[:width]]
          [submodel[:width]]
          [author[:width]]
          [date[:width]]
          [implementer[:width]]
          [implementdate[:width]]
          [reviewer[:width]]
          [reviewdate[:width]]
          [physicalunits[:width]]
          [dataType[:width]]
          [dimension[:width]]
          [file[:width]]
          [common[:width]]
          [value[:width]]
          [requirements[:width]]
                      >fileout.tex
```

Figure 2.7: Usage of formatDD.pl.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Name	Version	Long Name	Symbol	Description	Common	Appears in
ALFCOF	01B	Scale factor for alpha dose	S_α	Estimates uncertainty in piece-wise linear fit of alpha dose as a function of time $\alpha(t)$.	SPALPH	ALPHDS
ALPHDO	01B	alpha dose to used fuel surface	α	values of alpha dose rate to the surface of used fuel	SPALPH	ALPHDS
ALPHRE	01D	release rate from alpha radiolysis	$A\,\hat{c}_\alpha(t)$	release rate from used fuel per container from alpha radiolysis		ALPHDS
AREABF	01B	area of the backfill	B_F	area of the backfill		VLGDEP
AREADZ	01B	area of the damaged zone	B_Z	area of the damaged zone		VLGDEP
BALPHA	01E	Fit coefficient b for alpha radiolysis	b_α	Empirical fit coefficient 'b' for alpha radiolysis, used in the calculation of of the degradation rate per unit surface area of fuel $c_\alpha(t) = [d_\alpha(t+t_c)]^{a_\alpha} 10^{b_\alpha}$	SPALPH	ALPHDS
BKFRAR	01D	frac of vault with backfill	$A_F \equiv 2T_F/S$	fraction of vault area containing backfill	VARLVG	VLGDEP VLTDEP
BUFRAR	01D	frac of vault with buffer	$A_B \equiv 2T_B/S$	fraction of vault area containing buffer	VARLVG	VLGDEP VLTDEP
EALPHA	01E	Statistical parameter alpha radiolysis	$\frac{1}{\sum(\log\alpha_i - \overline{\log\alpha})^2}$	Based on experimental data correlating alpha dose to rate of fuel corrosion. Used to estimate the standard deviation of predicted corrosion rate.	SPALPH	ALPHDS
EXPONA	01C	log(predicted alpha corrosion rate)	$\log\hat{c}_\alpha(t)$	base 10 log of predicted corrosion due to alpha dose as a function of time		ALPHDS
FALPHA	01C	Mean experimental alpha radiolysis	$\overline{\log\alpha}$	Mean experimental alpha radiolysis	SPALPH	ALPHDS

Figure 2.8: Portion of a Data Dictionary listing, including a cross reference column.
Input and output data flows are colour coded green and red, respectively. Local variables are black.

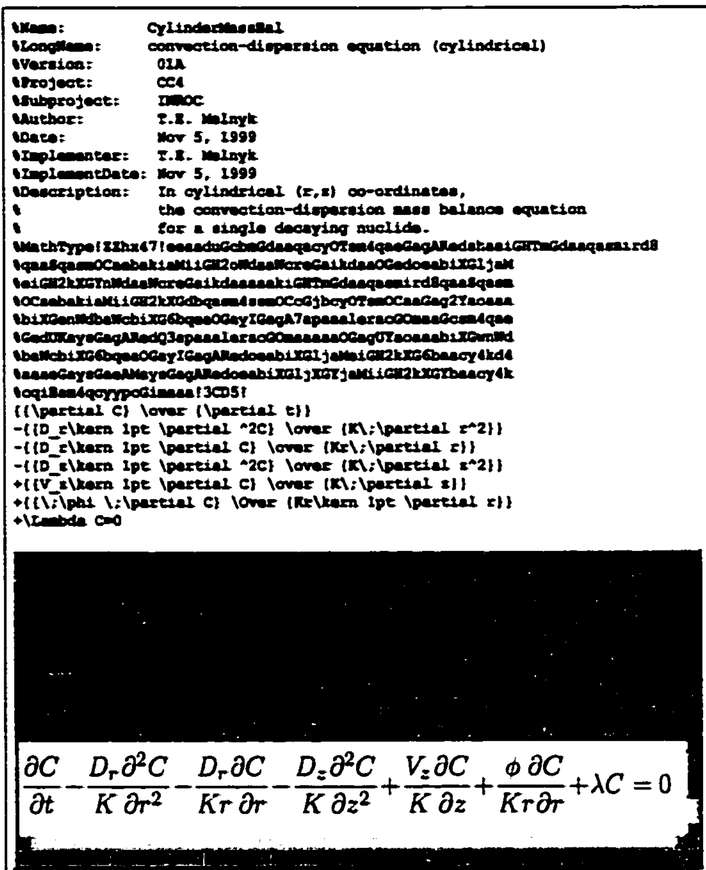
2.2.4 Equations

Equations are held in individual files, with version information similar to other TEXspec components. These files can be inserted into LATEX documents using the *\input{}* macro. A slight modification to the usual TEXspec file format stores TEXspec information in LATEX comments, as shown in Figure 2.9.

It has proved convenient to generate these files using a PC/Macintosh product called MathType, which adds additional comments to the file, containing encoded information which allows the equation to be

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

used in PC-based word processors, as well as LaTeX, as shown in Figure 2.9. The comments generated by MathType are ignored by LaTeX, but can be imported back onto the personal computer for inclusion in word processing documents. This decreases the possibility of inconsistency between TeXspec documentation products and related technical reports, memoranda, etc. that reference the same equations.



```
%Name:           CylinderMassBal
%LongName:       convection-dispersion equation (cylindrical)
%Version:         01A
%Project:         CC4
%Subproject:      IMROC
%Author:          T.E. Melnyk
%Date:            Nov 5, 1999
%Implementer:     T.E. Melnyk
%ImplementDate:  Nov 5, 1999
%Description:     In cylindrical (r,z) co-ordinates,
%                 the convection-dispersion mass balance equation
%                 for a single decaying nuclide.
%MathType!ZZhx47!eeaaduGcbaGdaaqacyOTam4qaeGagAAedakaaiGHTmGdaaqasairdS
%qaaSqasmOCaabakiaMiiGH2oWdaaWcreGaikdaaOGadoeabiXGljaM
%eiGH2kXGYnWdaaWcreGaikdaaaaakiGHTmGdaaqasmirdSqaaSqasm
%OCaabakiaMiiGH2kXGdbqasm4semOCcGjbcyOTamOCaaGag2Yaoaaa
%biXGenWdbaWcbiXG6bqaaOGayIGagA7apaaaleracGOmaaGcsm4qae
%GadUKayeGagAAedQ3epaaaleracGOmaaaaaOGagUYaoaaabiXGwnWd
%baWcbiXG6bqaaOGayIGagAAedoeabiXGljaMaiGH2kXG6baacy4kd4
%aaaeGayeGaaAMayeGagAAedoeabiXGljXGYjaMiiGH2kXGYbaacy4k
%oqiBam4qcyypoGimaaa!3CD5!
{{\partial C} \over {\partial t}}
-{{D_r\kern 1pt \partial ^2C} \over {K\;\partial r^2}}
-{{D_r\kern 1pt \partial C} \over {Kr\;\partial r}}
-{{D_z\kern 1pt \partial ^2C} \over {K\;\partial z^2}}
+{{V_z\kern 1pt \partial C} \over {K\;\partial z}}
+{{\;\phi \;\partial C} \Over {Kr\kern 1pt \partial r}}
+\Lambda C=0
```


Figure 2.9: An example Equation file, shown in ASCII format (top) and being edited by MathType (bottom)

Although the use of MathType is optional, many users prefer the use of a graphical equation editor over ASCII input of LaTeX math syntax. Figure 2.9 illustrates the use of the graphical editor and shows the ASCII equivalent. Other graphical editors are available, including TeXaide, which is available without charge from the manufacturers of MathType (but lacks the word processor interface).

The syntax for the Equation file is as follows:

Syntax of Equation (.teq) file	
%Name:	▷*short name in ascii format - minimal for unique identification*◁
%LongName:	▷*descriptive name in ascii format - up to a sentence*◁
%Version:	▷*version number for tracking history - appears on listings*◁
%Project:	▷*project identification*◁
%Subproject:	▷*sub-project identification*◁
%Author:	▷*author's full name*◁
%Date:	▷*date that the entry was written*◁
%Implementer:	▷*full name of person who input this entry into the system*◁
%ImplementDate:	▷*date that the entry was entered into TeXspec*◁
%Reviewer:	≻*full name of reviewer*≺
%ReviewDate:	≻*date of review*≺
%Description:	▷*full description - up to a paragraph*◁
≻*comments from MathType*≺	
▷LaTeX equation◁	

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.3 Application Composite Components

Some TEXspec input files are directly associated with a final product. They typically reference the shared components discussed in Section 2.2.

Each of these files is the primary input for TEXspec processing as shown in Figure 2.4. Note that Design Specifications act as both a primary input for *designSpec.pl* and a shared component for *structureChart.pl*.

2.3.1 Data Flow Diagrams

DFDs are stored by name, and are assigned a number only when the processing script (dfd.pl) is run. This mechanism allows a project to be re-numbered without necessarily changing the content of the diagram. The output from the processing script is named according to the specified number, which is then processed by LaTeX. This naming convention is important for consistency checking, as discussed below.

Figure 2.10 illustrates this process. The diagram '*DiagramName*' is assigned number 1.2.3, which is represented as '1_2_3' in file names. Consistency checking is performed against the parent Data Flow Diagram (DFD 1.2) as described below.

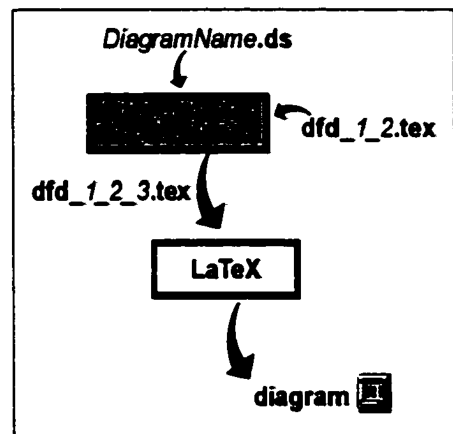


Figure 2.10: Dataflow Diagram processing, specifying the diagram number (1.2.3) at run time.

Syntax for processes (often called 'bubbles' when speaking of Data Flow Diagrams) and data stores are described by Yourdon [30]. Of particular importance is the distinction between 'atomic' processes (i.e., processes which have an associated Process Specification), which are shown with double circles, and processes with lower level DFDs (i.e., processes associated with child DFDs which decompose the process further) which are shown with a single circle.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The syntax for the DFD file is as follows:

Syntax of Data Flow Diagram (.dfd) file	
Name:	▷*short name in ascii format - minimal for unique identification*◁
Version:	▷*version number for tracking history - appears on listings*◁
Project:	▷*project identification*◁
Subproject:	▷*sub-project identification*◁
Author:	▷*author's full name*◁
Date:	▷*date that the entry was written*◁
Implementer:	▷*full name of person who input this entry into the system*◁
ImplementDate:	▷*date that the entry was entered into TeXspec*◁
Reviewer:	≻*full name of reviewer*≺
ReviewDate:	≻*date of review*≺
Units:	≻*valid LaTeX units of measure*≺
Labels:	▷ *'math','short','med', or 'long'*◁
⊵[**process spec**]⊴	
⊵[**connector spec**]⊴	
⪰ [**datastore spec**] ⪯	
⊵[**flow spec**]⊴	
≻ [**legend spec**] ≺	
Notes:	≻*annotation associated with the diagram*≺
Where:	
process spec =	
Process ▷#◁:	▷*dfd or mini-spec name (name may include '\\' = line beaks for labeling)*◁
At:	▷*x,y coordinates in specified units*◁
atomic	≻*flag to indicate that process is a mini-spec*≺
connector spec =	
Connector:	▷*label for off-page connector*◁
At:	▷*x,y coordinates in specified units*◁
datastore spec =	
DataStore:	▷*label for data store*◁
At:	▷*x,y coordinates in specified units*◁
flow spec =	
Flow:	▷*entry in Requirements Data Dictionary*◁
From:	▷*process, connector, or data store*◁
To:	▷*process, connector, or data store*◁
Type:	≻*'static' or 'temporal'*≺
Inflection:	≻*curvature of arrow*≺
RelPos:	≻*position of label along the curve (0,1 are the ends)*≺
LabelOffset:	≻*offset of label away from the curve (99 = do not label)*≺
legend spec =	
Legend:	▷ *'vertical' or 'horizontal'*◁
At:	▷*x,y coordinates in specified units*◁

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Connectors are placed at the ends of arrows representing flows that terminate outside the current diagram. Processes, data stores and connectors are all located on the diagram by specifying *(x,y)* coordinates, in units selected by the user.

The specified positions are relative, but the scale is absolute. The origin will be located so that negative values will not be placed off the page. Distances between objects that are larger than the available drawing area causes the diagram to be truncated; no scaling is performed.

Flows are specified by stating the end points (processes, data stores, or connectors), the inflection of the curve and label location. The meaning of the values for curve inflection and label location are defined by the xypic [23] package. The inflection is specified as the offset from linear at the midpoint of the curve, in the same units as the rest of the diagram, with positive values bending up and to the left and negative values bending down and to the right. Label location is specified relative to the flow, with 0 being the start of the flow and 1 being the end of the flow, but values less than 0 or greater than 1 are permitted. Label offset values place the label the specified distance from the curve, with positive values being above the curve and negative offsets being below the curve.

```
Title:          Determine Speciation of Groundwater
Version:        01B
Project:        INROC-LX
Submodel:       iareo
Author:         Ted Melnyk
Date:           Feb 22, 2000
Implementer:    Steve Oliver
ImplementDate:  Sep 29, 2000

Units:          inches
Labels:         Name

Process 1:      Calculate\\Calcium-Sulphate\\Concentrations
At:             3,1
atomic

Process 2:      Calculate\\Minor\\Anion\\Concentrations
At:             3.25,3

Process 3:      Adjust\\Sodium-Chloride\\Concentrations
At:             3,5
atomic

Connector:      Input NCASUL
At:             0.5,1
Connector:      Input fprion
At:             0.5,2
Connector:      Input spmjon
At:             0.5,2.5
Connector:      Input minor_eq
At:             0.5,3.5
Connector:      Input spmion
At:             0.5,4
Connector:      Input spph
At:             0.5,4.5
Connector:      Input KF
At:             0.5,5
Connector:      Output ISF
At:             6,5.5
Connector:      Output CCL
At:             6,5
Connector:      Output conc_minoranions
At:             6,3.95
Connector:      Output CSUL
At:             6,1.4
```

```
Flow:           NCASUL
From:           Input NCASUL
To:             Calculate\\Calcium-Sulphate\\Concentrations
Inflection:     -0.3
RelPos:         0
LabelOffset:    -0.15

Flow:           fprion
From:           Input fprion
To:             Calculate\\Calcium-Sulphate\\Concentrations
Inflection:     0.1
RelPos:         0.15
LabelOffset:    -0.15

Flow:           fprion
From:           Input fprion
To:             Adjust\\Sodium-Chloride\\Concentrations
Inflection:     -0.4
LabelOffset:    99

        •
        •
        •

Flow:           CH2PS
From:           Calculate\\Minor\\Anion\\Concentrations
To:             Adjust\\Sodium-Chloride\\Concentrations
Inflection:     -0.6
RelPos:         0.35
LabelOffset:    -0.27

Notes:
Implemented by SPCGCM
```

Figure 2.11: Example DFD file.
Not all flows are shown. Note the use of the \\ to denote a line break in the 'Process' names. The 'Notes' are supplemented by generated notes from TEXspec, as shown in Figure 2.12.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

An example of a DFD file is shown in Figure 2.11, and the output generated by TEXspec and LATEX is shown in Figure 2.12. A complete example is contained in Appendix A.

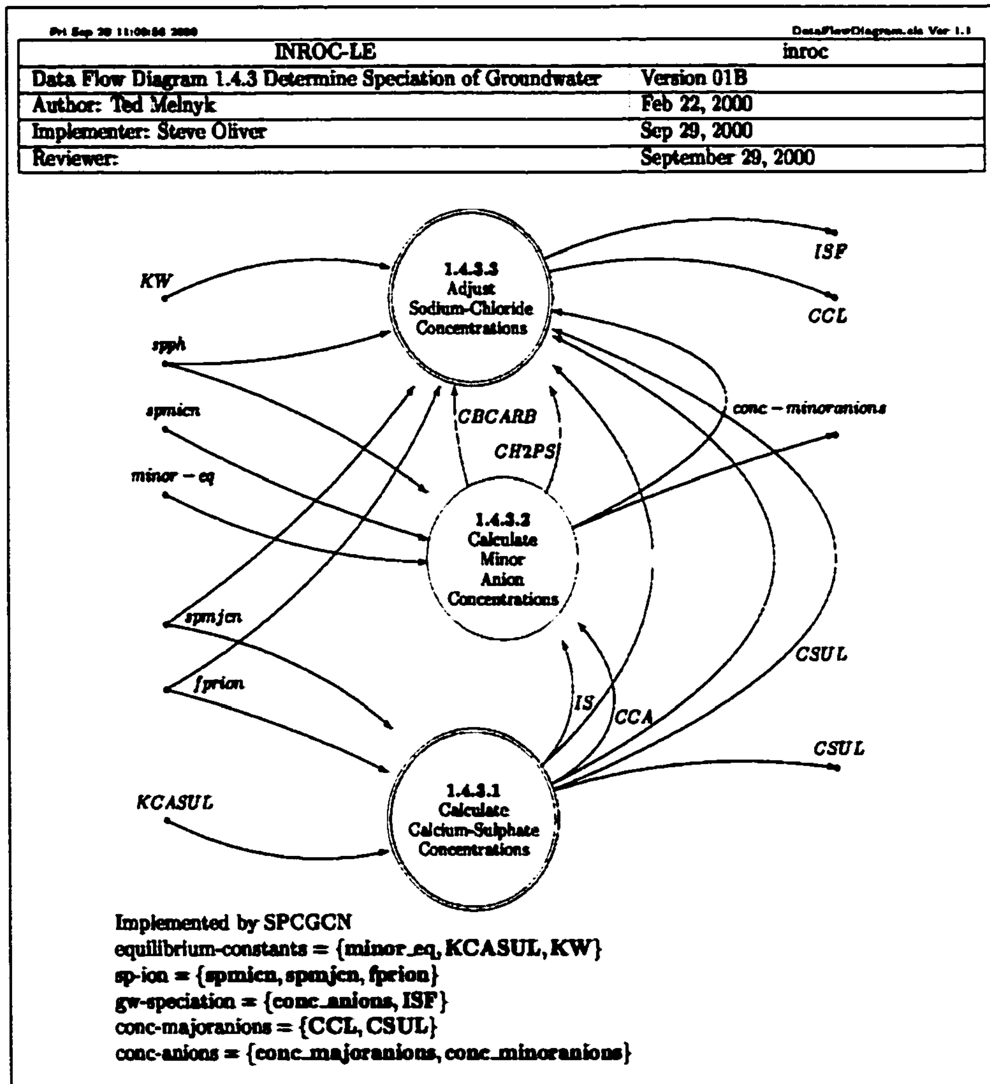


Figure 2.12: Example Data Flow Diagram.
'Notes' are generated to detail the contents of any composite flow whose contents appear on the diagram. The components which appear on the diagram are shown in bold type.

TEXspec supports two types of flows: 'static' (not time dependent) and 'temporal' (time dependent). This contrasts with the Yourdon [30] specification, which supports 'data' and 'control' flows. Both 'static' and 'temporal' flows would be considered 'data' flows by Yourdon. The visual presentation of two distinct types of flow is similar and only a generated legend (which is optional) would betray the user who redefined the two TEXspec flow types for the purposes specified by Yourdon. In the future, TEXspec may be enhanced to support a third ('control') flow type, or perhaps an arbitrary number of flow types.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Consistency between DFDs is monitored by TEXspec. As shown in Figure 2.10, at run time dfd.pl accepts an input parameter to define the diagram number. Generated output is tagged with the diagram number, by including the number in the name of the file containing the generated output. The script looks for output from the parent of the assigned diagram number by searching for the file name containing the parent's diagram number. If output from the parent diagram does not exist, then a warning message is generated. If a parent diagram does exist, then consistency is checked, allowing for composite flows. The input and output flows on the current diagram must correspond to the flows to/from the appropriately numbered Process on the parent diagram and all flows belonging to that Process must be represented on the child diagram. This can be either an exact match, or flows on the child diagram may be contained in composite flows on the parent.

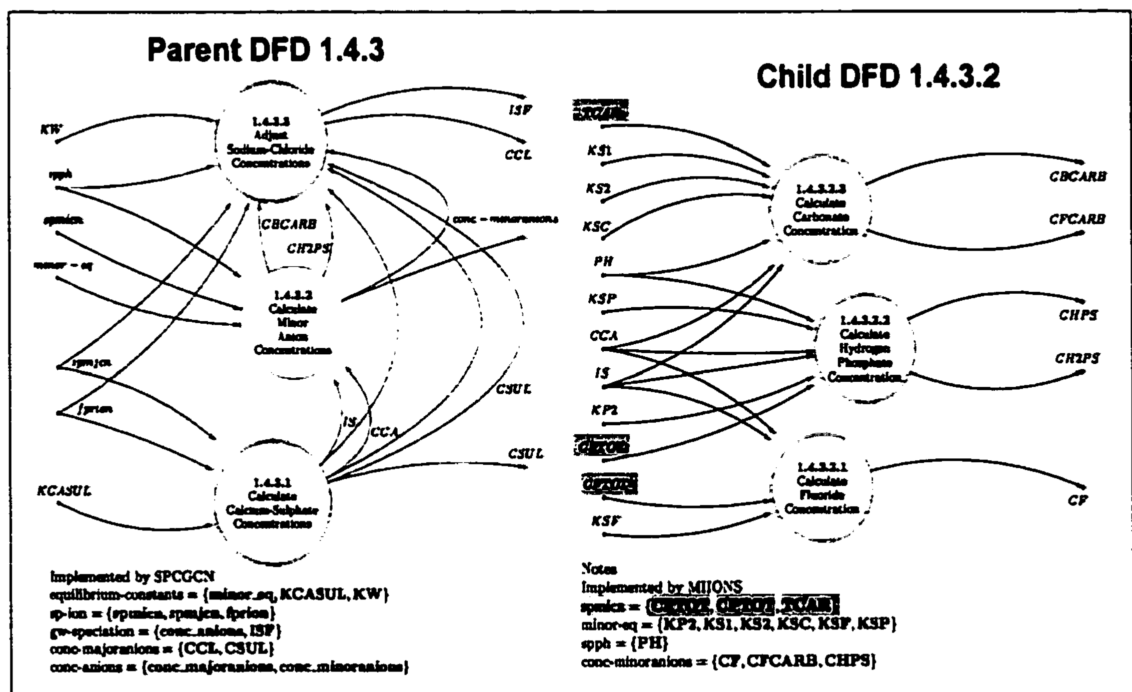


Figure 2.13: Consistency Checking of DFDs.
Parent DFD 1.4.3 is shown on the left and it's only child (DFD 1.4.3.2) is shown on the right. The highlighted fields illustrate consistent use of a composite flow - no highlighting appears on actual output.

Figure 2.13 illustrates consistency checking. The parent diagram (DFD 1.4.3, on the left) contains three Processes. Process 1 and Process 3 are represented by double lined circles, indicating that they are 'atomic' and are detailed in an equivalently numbered Process Specification. Process 2 is represented by a single lined circle, indicating that a child diagram (DFD 1.4.3.2) exists, as shown on the right.

To illustrate the treatment of composite flows, 'spmicn' is highlighted in red and it's components are highlighted in green. The child diagram (on the right) shows inputs of 'CFTOT', 'CPTOT' and 'TCAR', which is consistent with flow 'spmicn' into Process 2 on the parent diagram. Detail of the decomposition is contained

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

in the 'Notes' section on the child Diagram. Note that 'spmicn' is itself a component of flow 'sp-ion', which would appear on the grandparent diagram (DFD 1.4).

2.3.2 Process Specifications (Mini-Specs)

Process Specifications are stored by name and are assigned a number only when the processing script (miniSpec.pl) is run. This mechanism allows a project to be re-numbered without necessarily changing the content of the specifications.

Input and output flows are specified as Requirements Data Dictionary entries. If the parent Data Flow Diagram (Section 2.3.1) has been processed, then the flows are verified for consistency, otherwise a warning message indicates that no verification was performed. Flows in the Process Specification must be atomic, but the corresponding flow on the Data Flow Diagram may be composite (although this is discouraged). Otherwise, consistency checking is analogous to checking between a Data Flow Diagram and it's parent.

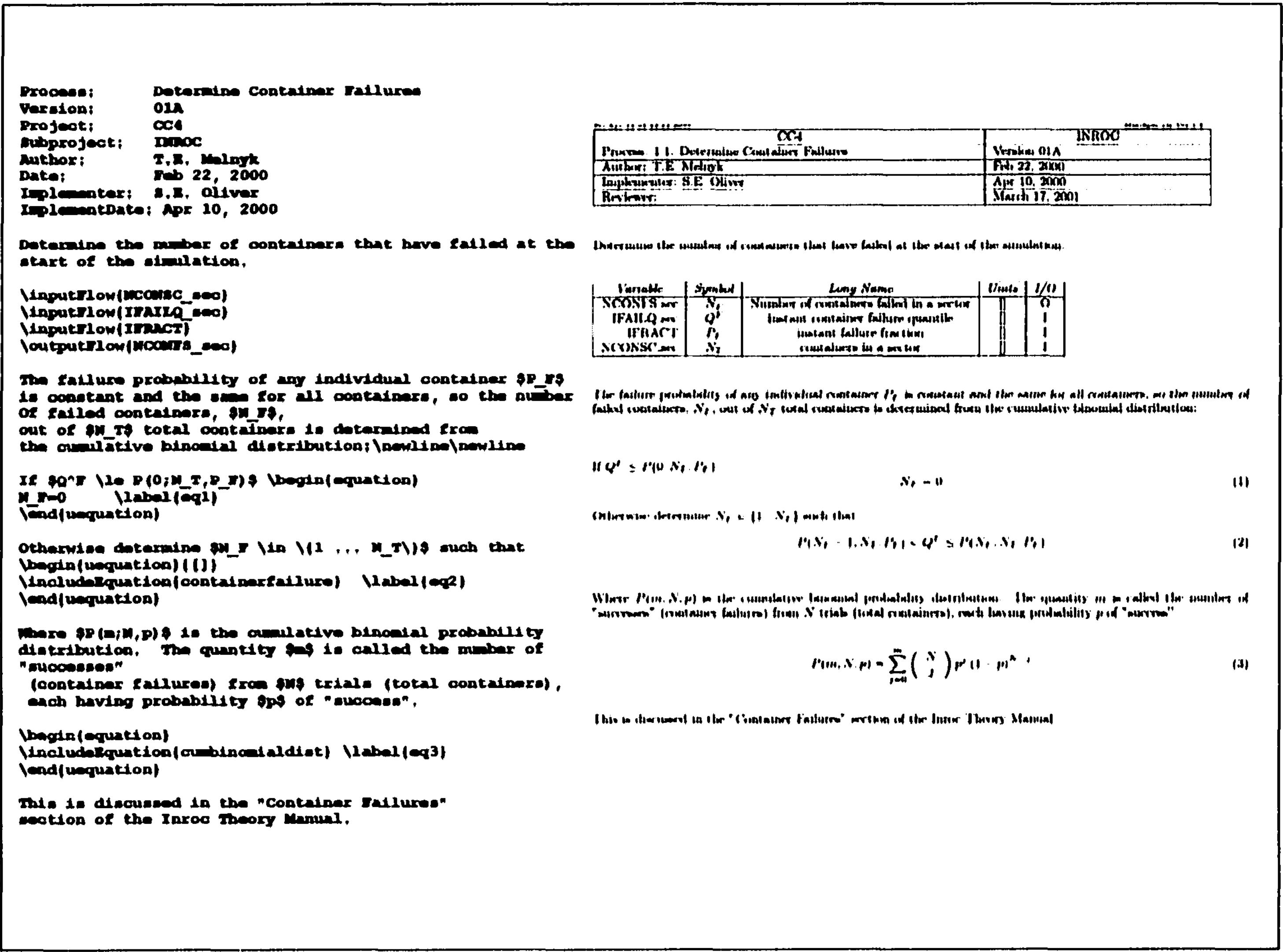
The detail of the process is specified in free form LaTeX. No consistency checking is performed between this and the specified flows. A macro is provided to allow the user to include a TeXspec equation. The macro *includeEquation*{name} causes TeXspec to scan the search list for name.*teq* and insert the contents at the specified position.

The syntax for the Process Specification file is as follows:

Syntax of Process Specification (.ms) file	
Process:	▷*short name in ascii format - minimal for unique identification*◁
Version:	▷*version number for tracking history - appears on listings*◁
Project:	▷*project identification*◁
Subproject:	▷*sub-project identification*◁
Author:	▷*author's full name*◁
Date:	▷*date that the entry was written*◁
Implementer:	▷*full name of person who input this entry into the system*◁
ImplementDate:	▷*date that the entry was entered into TeXspec*◁
Reviewer:	≻*full name of reviewer*≺
ReviewDate:	≻*date of review*≺
\begin{description}	
▷short description◁	
\end{description}	
⪰ \inputFlow{*Requirements Data Dictionary entry*}⪯	
⪰ \outputFlow{*Requirements Data Dictionary entry*}⪯	
▷*LaTeX description of process*◁	

An example of a Process Specification file is listed in Figure 2.14, with the corresponding specification as generated by TeXspec and LaTeX.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



```
Process:        Determine Container Failures
Version:        01A
Project:        CC4
Subproject:     INROC
Author:         T.E. Melnyk
Date:           Feb 22, 2000
Implementer:    S.E. Oliver
ImplementDate:  Apr 10, 2000

Determine the number of containers that have failed at the
start of the simulation.

\inputFlow(NCONSC_sec)
\inputFlow(IFAILQ_sec)
\inputFlow(IFRACT)
\outputFlow(NCONFS_sec)

The failure probability of any individual container $P_F$
is constant and the same for all containers, so the number
Of failed containers, $N_F$,
out of $N_T$ total containers is determined from
the cumulative binomial distribution:\newline\newline

If $Q^F \le P(0;N_T,P_F)$ \begin(equation)
N_F=0      \label(eq1)
\end(uequation)

Otherwise determine $N_F \in \{1 ... N_T\}$ such that
\begin(uequation)([])
\includeEquation(containerfailure)  \label(eq2)
\end(uequation)

Where $P(m;N,p)$ is the cumulative binomial probability
distribution.  The quantity $m$ is called the number of
"successes"
 (container failures) from $N$ trials (total containers),
 each having probability $p$ of "success",

\begin(equation)
\includeEquation(cumbinomialdist) \label(eq3)
\end(uequation)

This is discussed in the "Container Failures"
section of the Inroc Theory Manual.
```

Process 1.1: Determine Container Failures (CC4)	INROC Version 01A
Author: T.E. Melnyk	Feb 22, 2000
Implementer: S.E. Oliver	Apr 10, 2000
Reviewer:	March 17, 2001

Determine the number of containers that have failed at the start of the simulation.

Variable	Symbol	Long Name	Units	I/O
NCONFS_sec	N_f	Number of containers failed in a sector	[]	O
IFAILQ_sec	Q^f	Instant container failure quantile	[]	I
IFRACT	P_f	instant failure fraction	[]	I
NCONSC_sec	N_T	containers in a sector	[]	I

The failure probability of any individual container P_f is constant and the same for all containers, so the number of failed containers, N_f, out of N_T total containers is determined from the cumulative binomial distribution:

If $Q^f \le P(0;N_T,P_f)$

$$N_f = 0 \quad (1)$$

Otherwise determine $N_f \in \{1 \ldots N_T\}$ such that

$$P(N_f - 1; N_T, P_f) < Q^f \le P(N_f; N_T, P_f) \quad (2)$$

Where $P(m;N,p)$ is the cumulative binomial probability distribution. The quantity m is called the number of "successes" (container failures) from N trials (total containers), each having probability p of "success".

$$P(m;N,p) = \sum_{j=0}^{m} \binom{N}{j} p^j (1-p)^{N-j} \quad (3)$$

This is discussed in the "Container Failures" section of the Inroc Theory Manual.

Figure 2.14: Example Process Specification.

The input file on the left resulted in the specification on the right.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.3.3 Design Specifications

Module design documentation adheres to the concept of literate programming [10, 12], which uses a single source file to generate both the Design Specification and compilable code (the same file is also referenced to assure consistency with Structure Charts, as discussed in Section 2.3.4).

The syntax follows a hybrid format, with TeXspec specific syntax providing an interface with the rest of the system. This component is processed by TeXspec to produce a Noweb [21] input file, which is then processed into final products.

Noweb is a literate programming tool that permits a module to be broken down into code chunks which consist of blocks of LaTeX commentary and corresponding compilable code. It has a simple syntax that is portable to most programming languages, including Fortran.

Noweb code chunks that are not referenced in other chunks are placed in the default code chunk << * >>. The description, declaration, "include", and directive chunks are generated automatically by TeXspec.

Although the generated code is not intended to be a maintained product, the description is replicated (as comments) in the generated code. The code 'chunks' are also commented, by practice, to allow easy navigation when using a symbolic debugger.

Also carried through to the code are the variable definitions from the Design Data Dictionary, These definitions are placed next to the variable declarations. This includes the 'Physical Units' assigned to each variable and allows the use of AECL's unit checking program 'UNITCK' on the generated Fortran code. UNITCK is a proprietary static analysis tool that balances physical units in each executable Fortran statement.

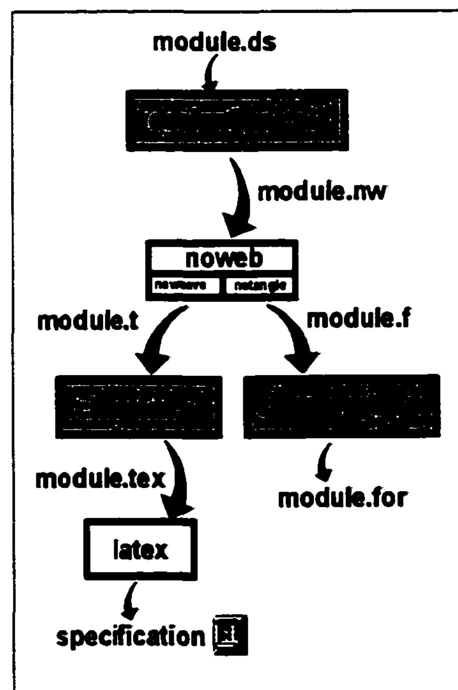


Figure 2.15: Design Specification Processing

The actual processing of a Design Specification occurs in stages, as shown in Figure 2.15. The processes performed by TeXspec PERL scripts appear in highlighted boxes. Other processes are shown as unshaded boxes.

A Design Specification file is processed by 'designSpec.pl' to produce a Noweb [21] input file. Noweb's two constituent programs 'noweave' and 'notangle' independently process this file to produce a LaTeX input file containing the formatted specification (see Figure 2.17), and an ASCII file containing the compilable code.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Noweb output contains declarations in 'code chunks' which would be printed in the specification. TEXspec prints a superset of this information in tabular format, so the 'cleantex.pl' PERL script removes the redundant code chunks before generating the Design Specification, without impacting the generated code.

The code output by Noweb contains, by default, many blank lines which make it difficult to use a symbolic debugger. The PERL 'cleanfortran.pl' is used to remove the extraneous blank lines.

Further reformatting of the code is up to the user. For example, it is possible to pass the Fortran through CERNs Floppy [2] package to reformat the code and produce a rudimentary static analysis. Most processing that users would perform on manually generated code can be applied to the generated code.

When revising and debugging code, it may be advantageous to eliminate the overhead of generating the documentation as shown on the left branch of Figure 2.15 (starting at 'noweave') until the code is stable.

Design Specifications are checked for internal consistency between declared variables and the Fortran code. Since information in the Design Data Dictionary is not repeated, but is extracted and placed in the Design Specifications (and hence the code), these products cannot be inconsistent with the Data Dictionary.

Information that appears in both the Design Specifications and the Structure Charts (Section 2.3.4) is also not repeated. The Design Specification acts as the repository of the shared information that the Structure Charts reference so they cannot be inconsistent.

Similarly, users are encouraged to share equations in a common pool (see Section 2.2.4). Although there is no requirement to do so, it is helpful to keep notation consistent and to propagate changes through all affected products.

Since both the code and the formatted specification are produced from the same file, TEXspec(through Noweb), acquires the attributes of literate programming [10, 12] systems, including consistency of the specification and the code. Correct code documented with an inconsistent Design Specification can result in many software defects [18], which cannot occur with literate programming techniques.

Arguments and shared variables must have a declared direction of flow: **'input'**, **'output'** or **'input,output'**. This information is reflected in tabular listings in the specification (the table for call arguments is similar to the table for shared variables shown in Figure 2.17). It is also used in the generation of Structure Charts (Section 2.3.4).

When the design specification is processed by TEXspec, the Fortran code itself is examined for internal consistency with the declared variables, including direction of flow. The use of undeclared variables is flagged, as is the declaration of variables that are not used. TEXspec issues a warning message if variables designated as 'output' flows are never the subject of a Fortran assignment statement, or if 'input' variables are changed. It is critical to have 'input' and 'output' correctly tagged, to ensure a correct Structure Chart (see Section 2.3.4), Dictionary Listings (see Section 2.2.3) and Design Specification.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The syntax for the Design Specification file is as follows:

Syntax of Design Specification (.ds) file

Module:	▷*module name in ascii format*◁
LongName:	≻*descriptive name in ascii format for Structure Chart*≺
Version:	▷*version number for tracking history - appears on listings*◁
Project:	▷*project identification*◁
Subproject:	▷*sub-project identification*◁
Author:	▷*author's full name*◁
Date:	▷*date that the entry was written*◁
Implementer:	▷*full name of person who input this entry into the system*◁
ImplementDate:	▷*date that the entry was entered into TeXspec*◁
Reviewer:	≻*full name of reviewer*≺
ReviewDate:	≻*date of review*≺
Language:	▷*'Fortran-77,' 'PROGRAM' or 'SUBROUTINE' or 'FUNCTION'*◁
Standard:	▷*applicable programming standard*◁

<<description>>=

▷*text description*◁

◁ %def description◁

⪰ [**argument**] ⪯

⪰ [**shared**] ⪯

Constant: ⪰*variable with an assigned value in Design Data Dictionary*⪯

⪰ [**local**] ⪯

⊵[**chunk**]⊴

Where:

argument =

Argument:	▷*variable in Design Data Dictionary*◁
Flow:	▷*'input' or 'output' or 'input,output'*◁
Dimension:	▷*Dimension to override definition in Design Data Dictionary*◁

≻ [**prepost**] ≺

shared =

Shared:	▷*variable in Design Data Dictionary*◁
Flow:	▷*'input' or 'output' or 'input,output'*◁

≻ [**prepost**] ≺

local =

Local:	▷*variable in Design Data Dictionary*◁
Dimension:	▷*Dimension to override definition in Design Data Dictionary*◁
Data:	▷*Initial value*◁

prepost =

Precondition:	≻*ascii text*≺ or
Postcondition:	≻*ascii text*≺

chunk =

▷ <<*chunk name*>>=◁

▷*code*◁

▷ %def chunk name◁

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

An example of a Design Specification file is shown in Figure 2.16, and portions of the output generated by TEXspec and LATEX are shown in Figure 2.17.

```
Module:         VLGDEP
Version:        06E
Project:        CC4
Submodel:       DMROC
Author:         S.E. Oliver
Date:           Feb 28, 2001
Implementer:    S.E. Oliver
ImplementDate:  Mar 13, 2001
LongName:       Determine time-independent vault parameters
Language:       FORTRAN77,SUBROUTINE
Standard:       none

<<description>>=
Determine time-independent vault parameters that require
parameters determined in GEOVLP

Based on code VERSION 06C  (2001-MAR-08) T. MELNYK
@ %def description

Shared:         MGFRAR
Flow:           input

Shared:         MGPERM
Flow:           input

Shared:         BUFRAR
Flow:           input

Shared:         CAPDMR
Flow:           output
                .
                .
                .

Shared:         NELMNT
Flow:           input
Precondition:   1 $\leq$ NELMNT $\leq$ MAXELM
                .
                .
                .

Local:          MAGRVO

Local:          MODNAM
Data:           'VLGDEP'

Local:          MSG
Dimension:      120

Local:          RETDAM

Local:          RKVCOS

Local:          RKVSIN
                .
                .
                .
```

```
VLGDEP implements Data Flow Diagram process
`Interface with Surrounding Geosphere'.

Additionally, VLGDEP derives parameters for vault
regions, based on the properties of the component
vault sectors.  The accumulation of multiple
vault sectors into a single vault region is
a design artifact intended to improve computational
efficiency.


The module consists of two sections
\begin{itemize}
\item Evaluate Darcy velocities and dispersion coefficients
      (`Interface with the Surrounding Geosphere'
      in the Theory Manual).
\item Evaluate regionalized vault properties.
\end{itemize}
<<main>>=
      <<geosphere>>
      <<regional>>
      RETURN
      END
@ %def main

\eep
            .
            .
            .
Evaluate components of Darcy velocity in rock for one sector
(SEC).

The room axis is assumed to be parallel to the X component
of the geosphere network cartesian coordinate system
so the axial component is simply
$\includeEquation{DarA_rock}$.

The transverse groundwater velocity in the rock
is correspondingly assumed to be in the YZ plane
of the geosphere network cartesian coordinate system
and is evaluated as
$\includeEquation{DarT_rock}$.

Define $\theta \equiv$ angle between the axis of the room
and the direction of water flow.  Compute
$\sin\left(\theta\right)$ and $\cos\left(\theta\right)$.

Assume permeability of buffer is zero, and
hence, Darcy velocity in buffer is zero
$\math{DARBV}=0$.
<<darcyComponents>>=
C.....Compute axial and radial components of Darcy velocity
      DARKVA(SEC) = DARRKX(SEC)
      DARKVR(SEC) = SQRT(DARRKY(SEC)**2 + DARRKZ(SEC)**2)
C.....Evaluate sin and cos of angle between room axis and
flow
      RKVSIN = DARKVR(SEC) / DARRK(SEC)
      RKVCOS = DARKVA(SEC) / DARRK(SEC)
@ %def darcyComponents

\eep
            .
            .
            .
```

Figure 2.16: Example Design Specification file.Not all flows or 'code chunks' are shown.

Figure 2.16 is not a complete listing, but illustrates the format of all sections of the Design Specification file. The complete listing is contained in Appendix B. Note the 'description' code chunk just after the initial fields. This chunk receives special treatment: it is reformatted into comments and placed near the top of the generated code, but is not reformatted in the specification (observe the 'description' code chunk in Figure 2.17).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The example module in Figure 2.16 has no arguments. Argument flows are placed in the flow list in the order that they occur in the argument list. By convention these flows would be declared before any shared or local variables to make this order clear.

Arguments and shared variables may optionally specify a precondition and/or a postcondition, depending on the direction of flow. If the variable has a 'Condition' in the Design Data Dictionary, then that condition is taken to be a precondition and/or postcondition, as appropriate. Explicitly stated preconditions and postconditions in the Design Specification file are added to anything contained in the Design Data Dictionary. For example, the variable *BKFRAR* is declared in Figure 2.16 without a precondition, but the precondition $0 \leq BKFRAR \leq 1$ is extracted from the Design Data Dictionary and appears below the table of shared variables.

Preconditions and postconditions for arguments and shared variables can optionally be accumulated together in the specification, but after some experimentation, the default behaviour has been set to place the conditions separately, below the appropriate table. This generates a longer specification, but keeps associated elements at close proximity, which makes the specification easier to read. In some cases (perhaps code which involves few variables), the accumulated format may be preferred, so the option to override the default behaviour remains.

Tables are formatted dynamically, so that no blank columns are produced. If no mathematical symbols exist for any variable in a table, then that table will not contain the 'Symbol' column.

The first major heading in the Design Specification is 'Module Components', which identifies the Noweb code chunks that comprise the default code chunk << * >>. This section is generated by TeXspec to include any chunks specified in the Design Specification file (which are not referenced by other code chunks), plus chunks generated by TeXspec. The generated chunks correspond to the sections of the document, but the order in the Design Specification is different from the order in << * >>, which specifies a compilable sequence.

For example, the << include >> code chunk is generated and placed before any executable code chunks in << * >>, but is detailed near the end of the Design Specification. This is because few readers wish to use this section, yet it can become quite large. Any declared variables whose Design Data Dictionary entry specifies a 'File' causes the file to be included in the << include >> code chunk ('INCLUDE' files in Fortran). This relieves the user from the burden of assembling the correct header files, as the job is performed automatically.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

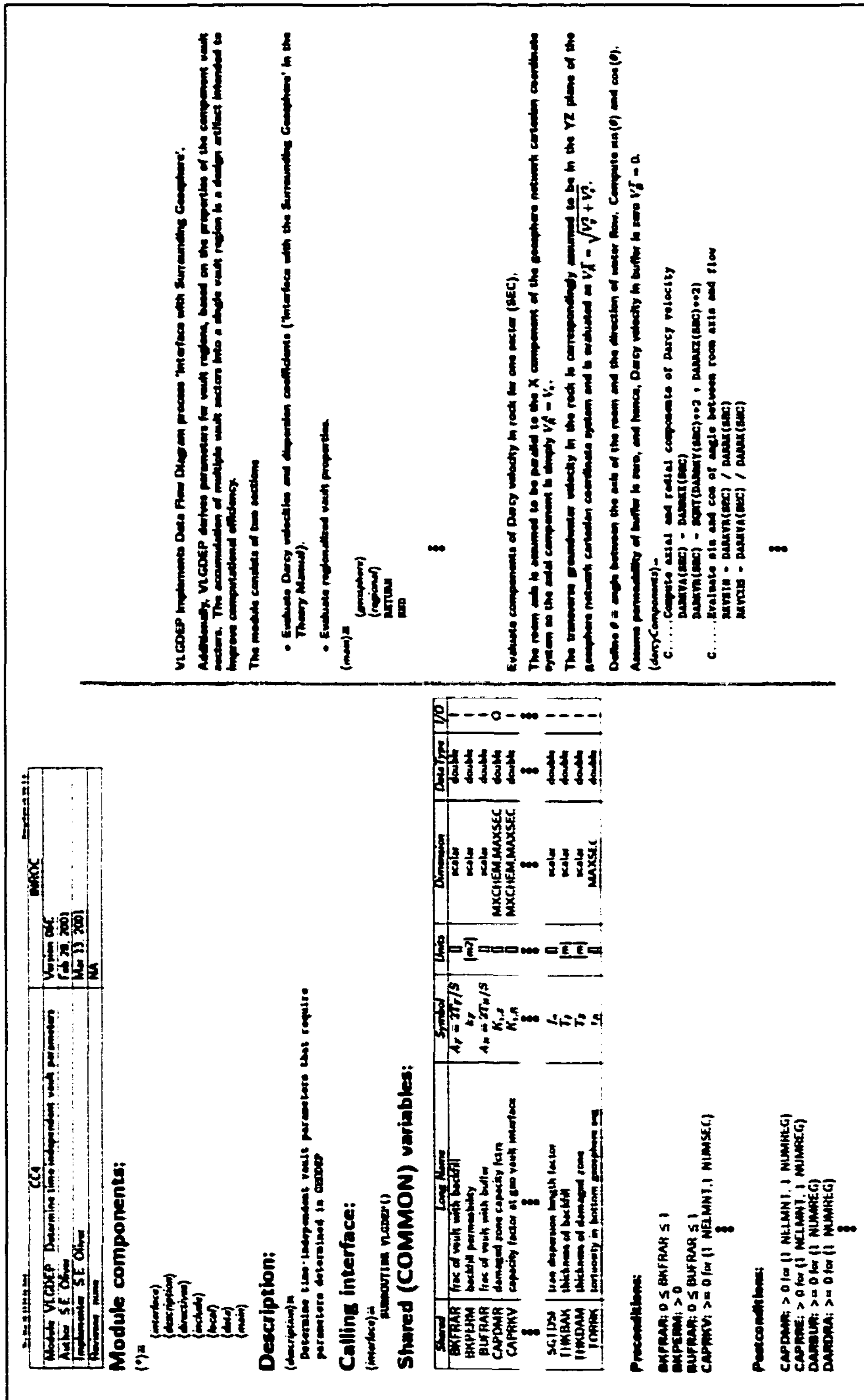


Figure 2.17: Example Design Specification.

Since the product is quite lengthy, only portions are shown here.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Only one table of variables is shown in Figure 2.17). There are no arguments to the example module and the table of local variables is not shown to conserve space in the figure. The table of local variables is similar, but does not have a column for the direction of flow ('I/O'). It would, however, have a column of values if any local variables were assigned a constant 'Value' in the Design Data Dictionary.

Two executable code chunks, '<< *main* >>' and '<< *darcyComponents* >>' are shown in Figures 2.16 and 2.17. There are several other chunks, but they are not shown. The 'Module Components' section of the specification specifies the content of the default code chunk '<< * >>'. Note that this references a number of generated code chunks and the input code chunk '<< *main* >>', but not '<< *darcyComponents* >>'. TeXspec places all input code chunks that do not appear in other code chunks into '<< * >>' in the order that they occur in the Design Specification file. Code chunks that are referenced by other code chunks, such as '<< *darcyComponents* >>', which is referenced by '<< *main* >>', are not placed in '<< * >>'.

The description of the '<< *darcyComponents* >>' code chunk illustrates the use of mathematical notation to clarify the specification. Some of this notation is input locally, and some is extracted from shared equations in **.teq** files via the *includeEquation*{} macro, which causes TeXspec to scan the search list for name.*teq* and insert the contents at the specified position.

2.3.4 Structure Charts

Structure Charts form the high level system design abstraction. They are similar to the format specified by Page-Jones [19], but include some additional information and use colour coding, rather than symbols and arrows, to specify the direction of data flow.

Structure Charts assemble Design Specifications in a manner roughly analagous to Data Flow Diagrams assembling Mini-specs. One difference is that Structure Charts are not layered, so each Module is 'atomic' and is not decomposed. The result is that a Structure Chart can be very large, so support is provided for off-page connectors which allow the user to break a Structure Chart into sections that can be sized convienently for publication. If multiple Structure Chart sections are connected with off-page connectors, then TeXspec verifies consistency between them using a method similar to that used for Data Flow Diagrams. For each off-page connector, TeXspec searches for a previously processed Structure Chart with the same name. If such a Structure Chart is found, then the connection is validated, otherwise a warning message is generated.

Options supported by TeXspec specifically for Fortran-77 display the status of 'COMMON' variables within each module, as well as in the argument list.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The syntax for the Structure Chart file is as follows:

Syntax of Structure Chart (.sc) file	
Chart:	▷*chart name in ascii format*◁
LongName:	▷*descriptive name in ascii format for Structure Chart*◁
Version:	▷*version number for tracking history - appears on listings*◁
Project:	▷*project identification*◁
Subproject:	▷*sub-project identification*◁
Author:	▷*author's full name*◁
Date:	▷*date that the entry was written*◁
Implementer:	▷*full name of person who input this entry into the system*◁
ImplementDate:	▷*date that the entry was entered into TeXspec*◁
Reviewer:	≻*full name of reviewer*≺
ReviewDate:	≻*date of review*≺
Units:	≻*valid LaTeX units of measure*≺
Labels:	▷*'long':maximum width and/or 'shared'*◁
EntryPoint:	≻*x,y coordinates in specified units*≺
SubmodelColour:	⪰*submodel name:colour code (default for submodel)*⪯
⪰ **[submodelcolour]** ⪯	
⊵**[module]**⊴	
⪰ **[offpage]** ⪯	
Where:	
module =	
Module:	▷*Design Specification*◁
At:	▷*x,y coordinates in specified units*◁
Background:	≻*colour code*≺
Caption:	≻*override of module long name*≺
CallString:	≻*x,y:maximum length*≺
[call]	
call =	
Call:	▷*module or off-page connector that appears on this chart*◁
Via:	⪰*x,y point on connecting line*⪯
offpage =	
OffPage:	▷*name of child Structure Chart*◁
At:	▷*x,y coordinates in specified units*◁

An example of a Structure Chart file is shown in Figure 2.18, and portions of the output generated by TeXspec and LaTeX is shown in Figure 2.19.

Much of the information on a Structure Chart is extracted from the referenced Design Specifications. The call interface, including the argument list and direction of data flow is extracted from each referenced Design Specification and placed above the module. if *Labels:shared* is specified, then any Fortran COMMON blocks are shown, in alphabetical order, with referenced variables colour coded by direction of data flow.

TeXspec performs some consistency checking between the source code contained in the Design Specifications and the Structure Chart. If the referenced (called) modules do not agree, TeXspec issues a warning message

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

that extra or extraneous calls are shown on the chart.

If *Labels: long* is specified, then the 'Long Name' in each Design Specification is placed with the module, as shown in Figures 2.18 and 2.19. If these names are too long, the boxes become excessively wide and the user can then specify *Labels: long:len* to specify a maximum width before a line break is used. Likewise, the *CallString: x,y:len* syntax allows an interface string to be broken over multiple lines.

```
Chart:          SIMALL
LongName:       Inventory of All Nuclides
Version:        01A
Project:        CC4
Submodel:       INROC
Author:         S. Oliver
Date:           December 17, 2000
Implementer:    S. Oliver
ImplementDate:  December 17, 2000

Units:          inches
Labels:         long,shared

EntryPoint:     2.1,9

Module:         SIMALL
At:             2.1,7
Call:           NUCINF
Via:            2.1,6.25
Via:            0,6.25
Call:           SOURCE
Via:            2.1,6.25
Via:            1.75,6.25
Call:           ZAPINT
Via:            2.1,6.25
Via:            2.75,6.25
Call:           PRECIF
Via:            2.1,6.25
Via:            4,6.25

Module:         NUCINF
At:             0,4.75
CallString:     0.5,6:22

Module:         SOURCE
At:             1.75,3
Call:           REPFUN
Background:     yellow

Module:         ZAPINT
At:             2.75,3.5
Background:     yellow

Module:         PRECIF
At:             4,5.25
CallString:     4.5,6.05:22

Module:         REPFUN
At:             1.75,2.25
Call:           INVTRY
Background:     yellow

OffPage:        INVTRY
At:             1.75,1.5
```

Figure 2.18: Example Structure Chart file.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

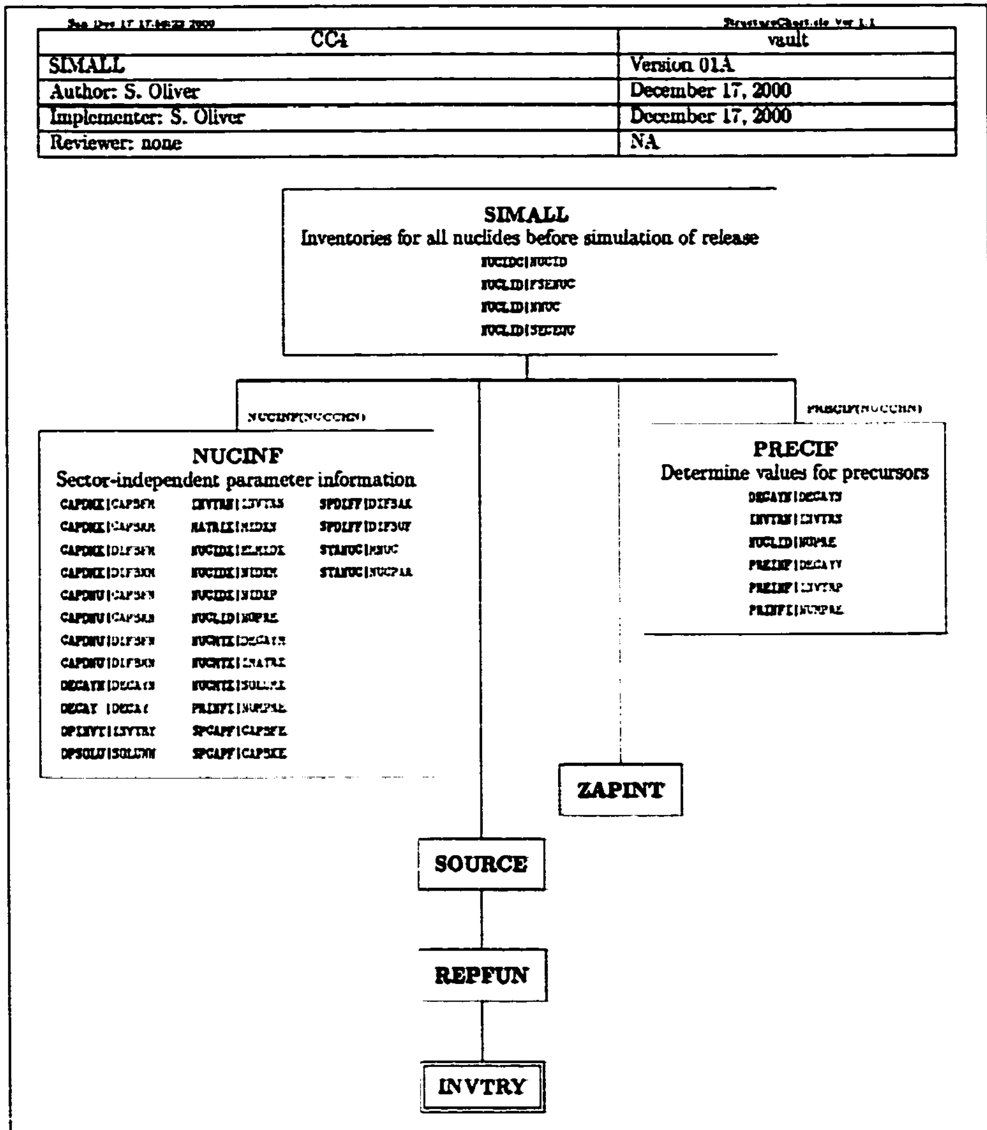


Figure 2.19: Example Structure Chart. Input and output data flows are colour coded green and red, respectively. INVTRY is an off-page connector.

2.3.5 Manuals

For the most part, manuals are simply LaTeX documents. TeXspec simply defines the syntax of the equation (Section 2.2.4) files to be inserted with the *input*{} macro.

Further support for manuals will be provided once a configuration management system is incorporated into TeXspec.

Also, the CSA standard [4] demands a number of specific documents, and templates will be provided.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3 Graphical User Interface

To assist in the operation of the TEXspec system, a GUI is provided, making the application much more intuitive to operate.

For the purpose of this practicum, the intention was to implement a 'simple but effective' GUI. Unfortunately, these objectives are not always consistent and the program now comprises over 32,000 lines of source code in 85 modules. This compares to 8,000 lines of PERL code in 8 scripts to implement the core TEXspec technology. At this writing, the GUI is in regular use, and has proven to be fairly robust.

3.1 Architecture

The GUI fits into the TEXspec architecture as shown in Figure 2.1. It is implemented as a Java application. It manipulates the input files, executes the PERL scripts, and handles the output.

The application is distributed as a Java archive (.JAR) file and is initiated by a Java runtime environment. From a command line, this often looks like:

```
java -jar TeXspecGUI
```

The initial presentation is as shown in Figure 3.1. The user must identify himself and declare a default project on which he will be working.

The options presented on the 'login' screen indicate the future development path of the product. At the moment, the options (user identification, project, and subproject) are 'hard coded' into the application and the 'Password' field does not process input. These fields will have meaning when the application is divided into client and server portions (Section 2.1.1)

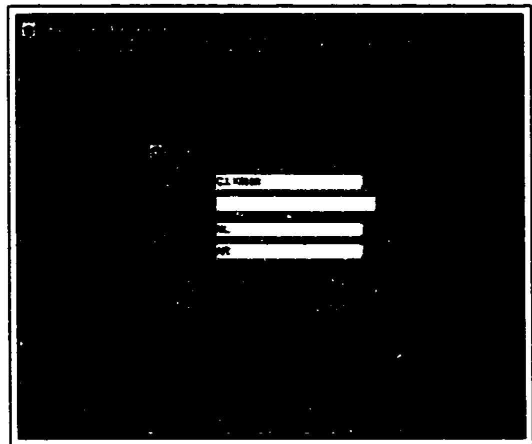


Figure 3.1: TEXspec GUI Initial Screen.

The GUI is based on components provided with Java and two additional libraries;

- 'regexp': regular expression parser from The Free Software Foundation (FSF)
- 'format': Henrik Bengtsson's printf package (for non-commercial use).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TeXspec components tend to be small and held in many files. Repeatedly opening a large number of files tends to inhibit performance on many systems, so the GUI has an abstract class 'TeXspecComponent' which establishes and maintains an inventory (cache) of components that have already been parsed. A background process periodically scans the search list directories (Section 2.1.2) for files that have been updated since they were last parsed and placed in inventory.

Figure 3.2: TeXspec GUI Desktop Screen. The bar along the bottom is a 'progress bar' and message area.

The number of windows generated by the GUI can be large. A desktop window is used to contain these windows, which avoids cluttering the user's main desktop with many TeXspec windows and icons. The desktop window also provides a convenient place for a progress bar, as shown in Figure 3.2.

3.2 Configuration and the Search List

Since the GUI is used to create and edit TeXspec components, as well as process them, the search-list has an additional role to play beyond the base functionality. The first directory (for each file type) defines the directory in which output will be written. No output is written to directories lower in the search-list, although they can be deleted. If a component is accessed from a lower directory, then edited and saved, the edited copy will be written to the first directory in the search-list. By placing a working directory at the top of the list for each file type, the user can collect his working files as they are modified and move them to the appropriate directories once the products are known to be satisfactory.

The Search List (Section 2.1.2) can contain a large list of directories to be searched. This would be onerous to regenerate each time the GUI is invoked. To avoid this, the GUI allows the user to load a 'Configuration' (which may in the future contain more than the search list). This allows the user to work on multiple projects without having to manipulate the search list on every invocation. To load a configuration, use `Options->Load Configuration` to bring up the chooser window, as shown in Figure 3.3.

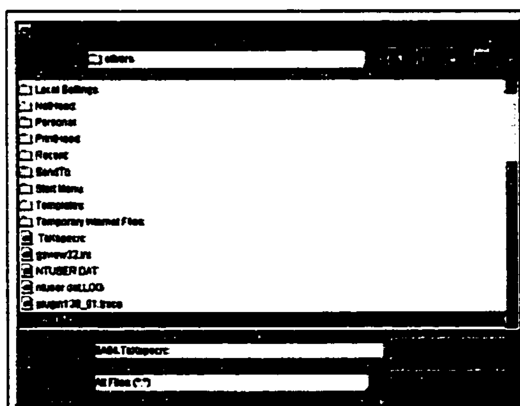


Figure 3.3: Chooser to select a file containing a search-list.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

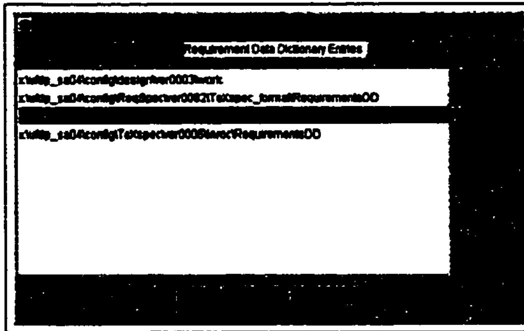


Figure 3.4: Window to edit a search-list.

Initially, however, a search list must be built before it can be saved and subsequently reloaded. To manipulate the list of directories, use `Options->Search List` to bring up the search list editing window, as shown in Figure 3.4. A drop-down menu allows the user to select a file type and directories can be added (via a pop-up chooser) or deleted using the edit buttons. The 'Up' and 'Down' buttons allow the order in which the directories are searched to be manipulated. When the list is complete, use `Options->Save Configuration As..` to save the search list.

3.3 Requirements Data Dictionary

Requirements Data Dictionary entries define flows or components of flows which occur on DFDs (Section 2.2.1).

They can be accessed by the following methods, from the `File` menu:

- `File->New->Requirements->Data Dictionary Entry`
  to create a new Data Dictionary entry.
- `File->Open->Requirements->Data Dictionary Entry`
  to edit an existing Data Dictionary entry (or create similar ones).
- `File->List->Requirements->Data Dictionary Entry`
  to see a list of existing Data Dictionary entries, or generate a formatted listing. suitable for printing (Section 3.5).

The `File->List` method allows the dictionary to be accessed from an alphabetical listing. After scanning the search list for the appropriate files, a window, as shown in Figure 3.5, displays the candidate entries. By default, these are in alphabetical order, but an option allows the entries to be sorted by project. The 'Refresh' button causes the search-list to be scanned for changed entries.

Multiple entries can be selected for 'Edit' or 'Delete' by holding down the 'shift' or 'control' buttons while selecting with the mouse. Editing is initiated with either the 'Edit' button or a mouse double-click.

The 'Generate Listing' button activates the Dictionary Listing window, as shown in Section 3.5.

Keeping a dictionary listing on hand is a useful method of avoiding logically duplicate entries. Scanning the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

'Long Name' column can quickly identify any existing definitions that might be used instead of a new entry.

Since the listing is generated from files and the time stamp on each file is checked before the listing is displayed, the dictionary listing can be slow to generate, particularly if a long search list is employed. It is usually a good practice to request a dictionary listing when the GUI is started and to keep the window for reference (perhaps shrunk to an icon).

Figure 3.5: List of Requirements Data Dictionary entries.

Having arrived at the 'Edit' window, via one of the mechanisms outlined above, as shown in Figure 3.6, fields are analogous to the Requirements Data Dictionary (.rdd) file (Section 2.2.1). Note the support for composite entries: the 'Add' button brings up the full list of available Requirements Data Dictionary entries, from which the desired components can be selected. The 'New' button brings up an empty Requirements Data Dictionary entry, which can be filled in and, when saved, becomes incorporated into the current entry. Similarly, the 'Edit' button can be used to edit a child entry (if there is one).

Figure 3.6: Edit a Requirements Data Dictionary entry.

Changing the 'Name' and saving creates a new Requirements Data Dictionary entry. This is a quick method to create several similar Requirements Data Dictionary entries.

The 'Math Name' field is intended to have a preview button, to allow for the fact that LaTeX equations often require more than one attempt to achieve a correctly formatted result. This has not yet been implemented.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.4 Design Data Dictionary

Largely analogous to the Requirements Data Dictionary, the Design Data Dictionary entries are referenced in much the same manner, but the editing window is slightly larger to handle the increased number of fields (defined in Section 2.2.2). Recall that the Design Data Dictionary entry may be mapped to a Requirements Data Dictionary entry, which can eliminate the need for some of these fields. Inheriting a 'Math Name' or 'Description' can save both typing and maintenance effort.

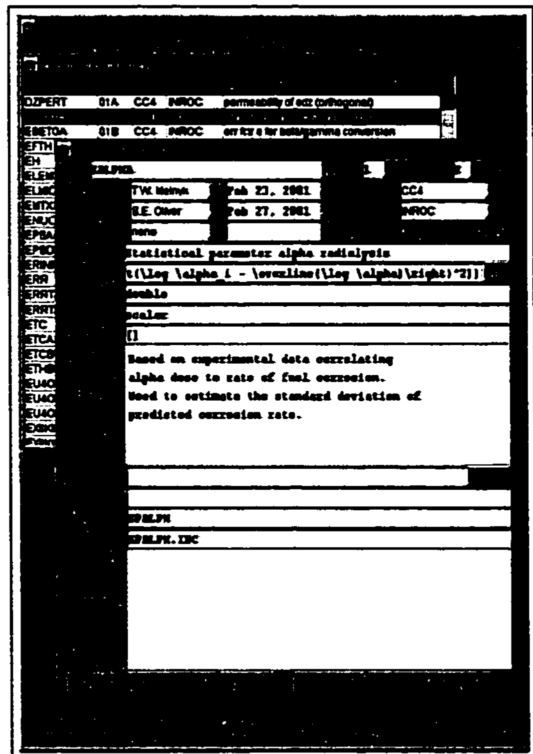


Figure 3.7: Edit a Design Data Dictionary entry.

The 'Select' button brings up the full list of available Requirements Data Dictionary entries, from which the corresponding entry can be selected. Alternatively, the name can simply be typed in.

3.5 Dictionary Listing

From the `File->List` window for either type of Data Dictionary entries, the 'Generate Listing' button will bring up the window shown in Figure 3.8. This window provides an interface with 'formatDD.pl' outlined in Section 2.2.3, through the script file 'formatDD.bat'. The mechanism is outlined in Section 3.11.

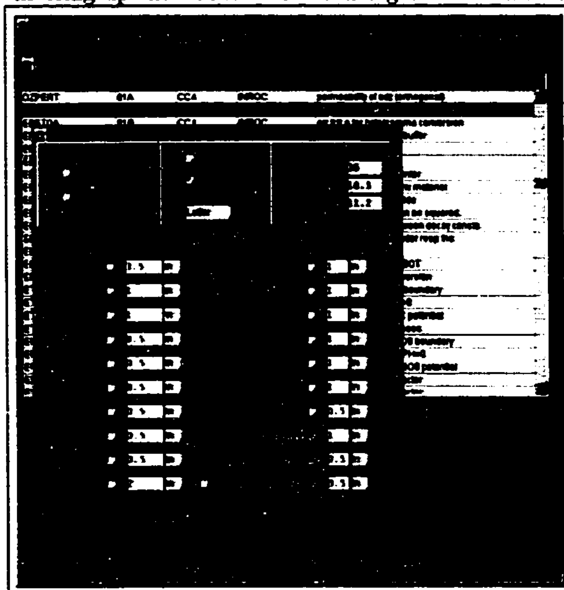


Figure 3.8: Generate a Data Dictionary Listing.

The dictionary listing module is very flexible (see Figure 2.7), and capturing all of that flexibility might result in an unnecessarily complicated interface. Some of the flexibility is compromised to achieve a more intuitive interaction. The available columns are easily seen and the column width can be adjusted, but the order of the columns cannot be controlled. Should experience prove that the order of the columns is important, then the design of this window may be reviewed.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.6 Process Specifications (Mini-specs)

Process Specifications are required for all atomic processes which occur on DFDs (Section 2.3.2).

They can be accessed by the following methods, from the `File` menu:

- `File->New->Requirements->Process Spec`
  to create a new Requirements Specification.
- `File->Open->Requirements->Process Spec`
  to edit an existing Requirements Specification (or create similar ones).
- `File->List->Requirements->Process Specs`
  to see a list of existing Requirements Specifications, or generate formatted listings. suitable for printing.

Flows on a Process Specification are Requirements Data Dictionary entries and are shown in tabular form on the editing screen, as illustrated in the leftmost window in Figure 3.9. Selecting a 'Flow' and pushing the 'Edit' button causes a Requirements Data Dictionary edit window (Section 3.3) to come up.

In Figure 3.9, the 'Add' button was used to bring up the list of Requirements Data Dictionary entries at the upper right. Selecting an entry from this window to form a new flow caused the window on the lower right to prompt for the direction of the flow (the remainder of the window echoes the content of the selected Requirements Data Dictionary entry in non-editable form).

Flows can be resorted according to several sorting schemes by toggling the 'Sort' button.

Note the support for a bibliography using BibTEX. Filling in the bibliography fields will cause the appropriate BibTEXcommands to be generated.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

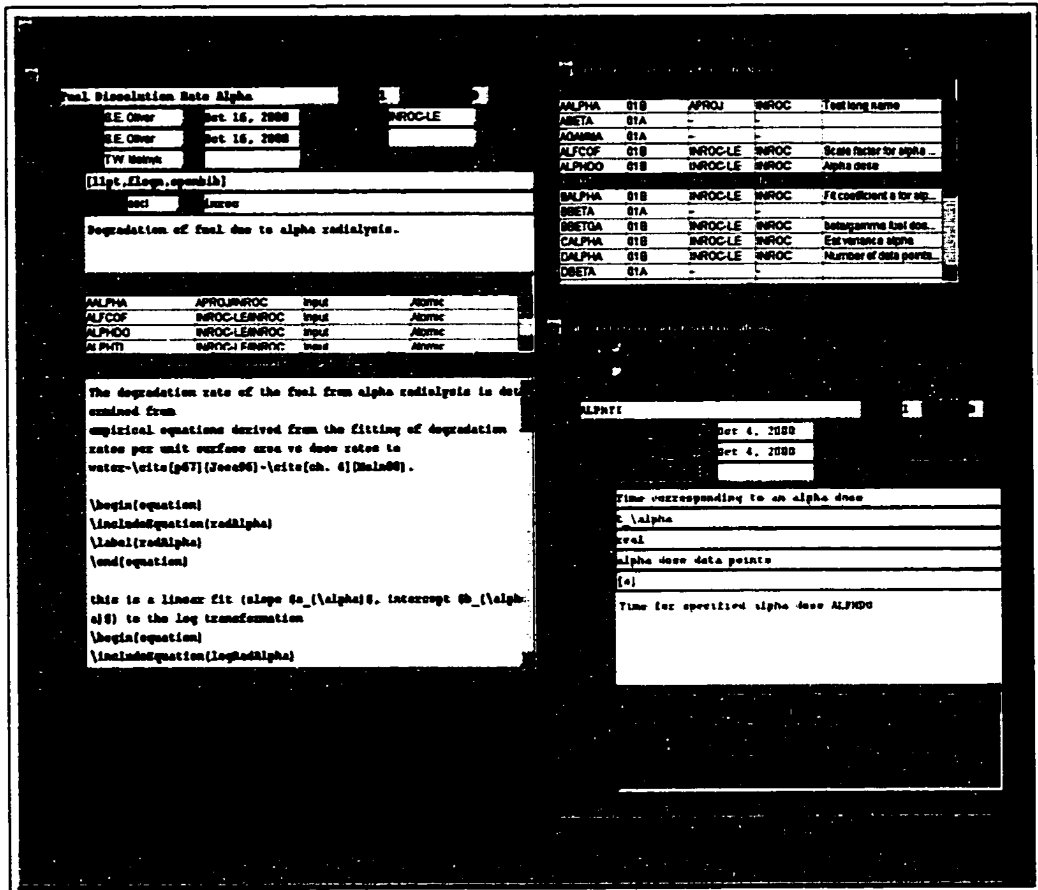


Figure 3.9: Edit a Process Specification.

3.7 Data Flow Diagrams

Data Flow Diagrams are high level abstractions of requirements, specifying conceptual processes and the flow of data between them. TEXspec DFDs use a modified Yourdon/DeMarco format traditionally employed by the DGRTP.

They can be accessed by the following methods, from the `File` menu:

- `File->New->Requirements->Data Flow Diagram`
  to create a new Data Flow Diagram.
- `File->Open->Requirements->Data Flow Diagram`
  to edit an existing Data Flow Diagram (or create similar ones).
- `File->List->Requirements->Data Flow Diagrams`
  to see a list of existing Data Flow Diagrams, or generate formatted listings suitable for printing.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The fields on the Data Flow Diagram editing screen are analogous to those in the Data Flow Diagram file (Section 2.3.1).

'Elements' on a Data Flow Diagram may be:

- atomic processes with a corresponding Process Specification,
- a child Data Flow Diagram,
- a data store, or
- an off-page connector.

The first two options are represented as circles (often called bubbles), and are grouped together as 'Processes'. They are distinguished in the 'Type' column of the 'Elements' section of the edit window. Selecting a process and pressing the 'Mini-spec' button will make the process 'atomic', create a Process Specification (Sections 2.3.2 and 3.6) and bring up an edit window as shown in Figure 3.9.

'Flows' on a Data Flow Diagram are shown in tabular form in the 'Flows' section of the editing screen, as illustrated in Figure 3.10. The 'Content' of 'Flows' on a Data Flow Diagram are Requirements Data Dictionary entries.

The edit screen shows the relationship between 'Elements' and 'Flows' by changing the typeface of the 'Flows' associated with the selected 'Elements' to a bold font. Likewise, the 'Elements' at either end of selected 'Flows' are shown in bold type.

Since the number of fields associated with both 'Elements' and 'Flows' are fairly small, they are placed on the edit window and no child windows are used. Valid data must appear in the data fields before the 'Add/Update' buttons become active.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

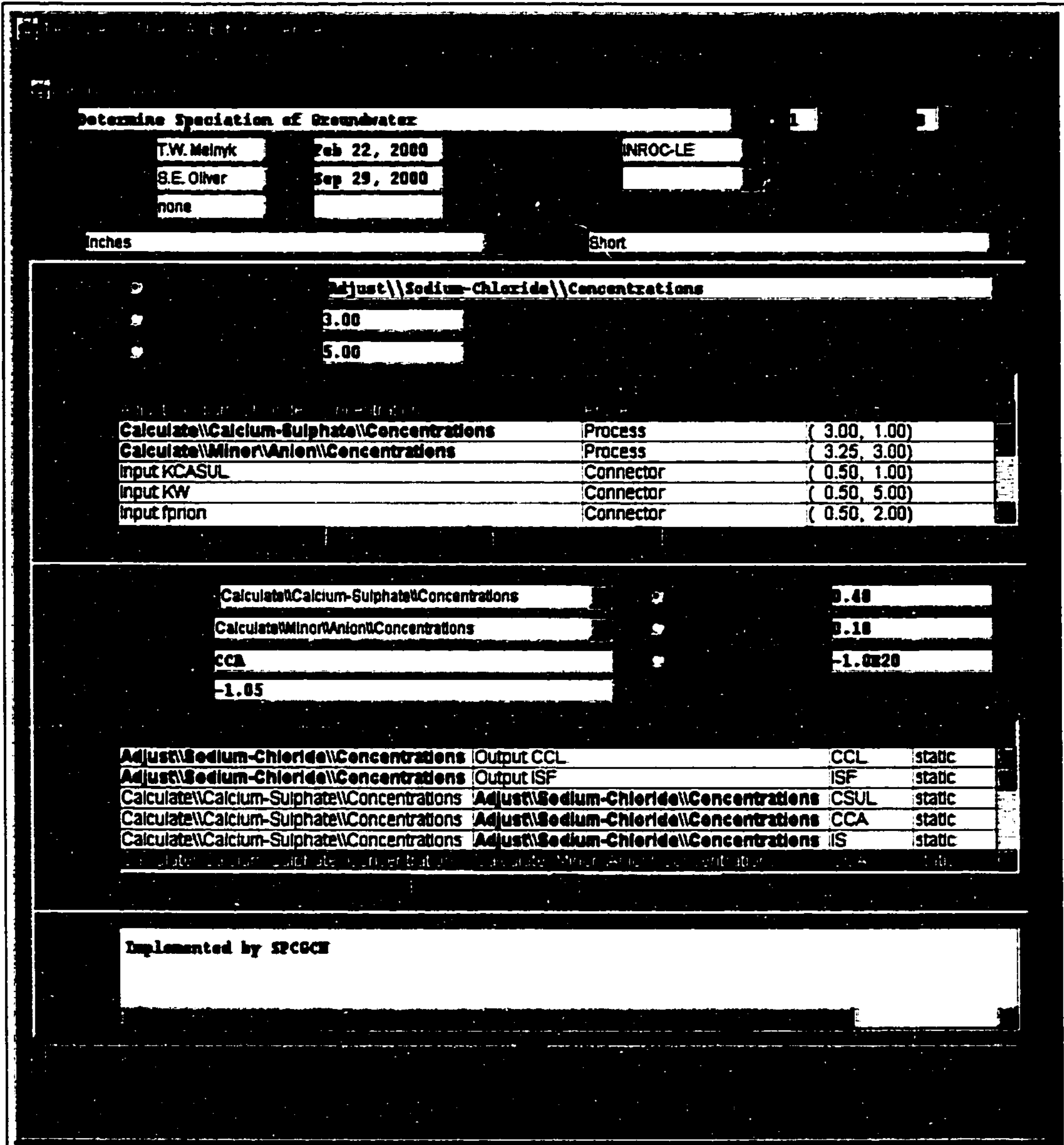


Figure 3.10: Edit a Data Flow Diagram.

3.8 Design Specifications

Design Specifications are required for all code modules (Section 2.3.3).

They can be accessed by the following methods, from the `File` menu:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- `File->New->Design->Module Spec`
  to create a new Design Specification.
- `File->Open->Design->Module Spec`
  to edit an existing Design Specification (or create similar ones).
- `File->List->Design->Module Specs`
  to see a list of existing Design Specifications, or generate formatted listings. suitable for printing.

Symbols in a Design Specification are Design Data Dictionary entries, and are shown in tabular form on the editing screen, as illustrated in the leftmost window in Figure 3.11. Selecting a 'Symbol' and pushing the 'Edit' button causes a Design Data Dictionary edit window (Section 3.4) to come up.

Note that the symbols are presented as two tabbed tables, one for 'Arguments' and the other for 'Variables'. Data flows can be considered to be all 'Arguments', plus those 'Variables' that are in shared storage (COMMON blocks in Fortran).

In Figure 3.11, the 'Add' button was used to bring up the list of Design Data Dictionary entries at the upper right. Selecting an entry from this window to form a new flow caused the window on the lower right to prompt for the direction of the flow (the remainder of the window echoes the content of the selected Design Data Dictionary entry in non-editable form). While the window is labeled 'Flow', in fact it declares a symbol, and specifying no flow direction causes non-shared symbols to become local variables.

The 'Flow' edit window allows the user to specify preconditions, postconditions and initialising data. Having the non-editable Design Data Dictionary fields displayed in the same window helps to avoid conflicts or duplication. The 'Units' and 'Dimension' of the Design Data Dictionary entry are subject to override here.

Note the support for a bibliography using BibTEX. Filling in the bibliography fields will cause the appropriate BibTEXcommands to be generated.

Noweb code 'Chunks' are input in commentary-code pairs in the tabbed panes on the edit (leftmost) window. Pressing the 'Add' button causes a a dialog to prompt for a name and a new pair is generated. Because 'designspec.pl' places Chunks into the default Chunk in the order that they occur, the Chunks are numbered and the 'Up'/'Down' buttons causes the selected Chunk to change it's position in the sequence.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 3.11: Edit a Design Specification.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.9 Structure Charts

Structure Charts are high level abstractions showing the relationships between code modules (Section 2.3.4).

They can be accessed by the following methods, from the `File` menu:

- `File->New->Design->Structure Chart`
  to create a new Structure Chart.
- `File->Open->Design->Structure Chart`
  to edit an existing Structure Chart (or create similar ones).
- `File->List->Design->Structure Charts`
  to see a list of existing Structure Charts, or generate formatted listings. suitable for printing.

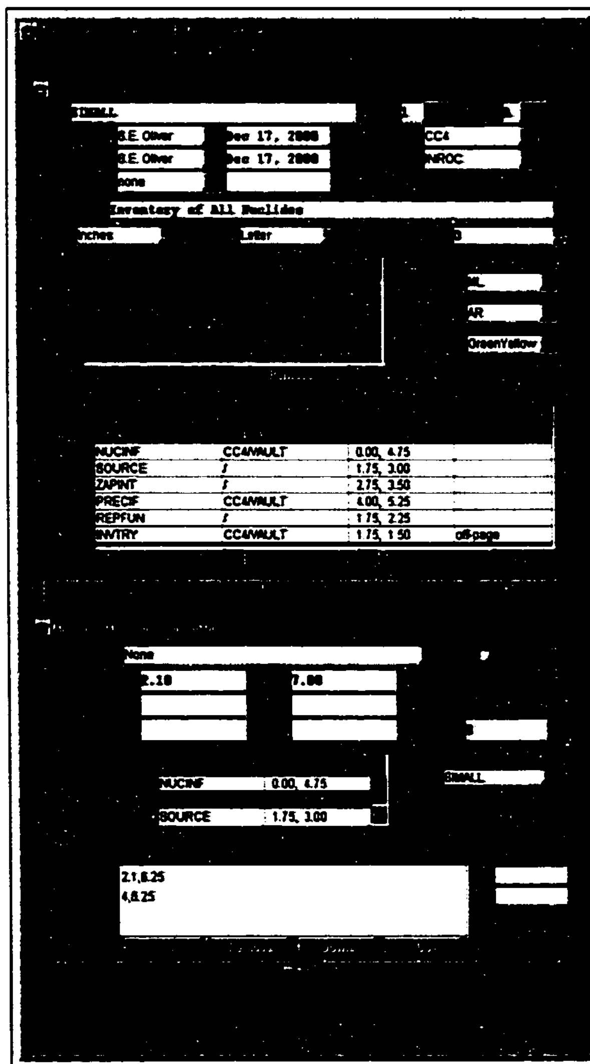


Figure 3.12: Edit a Structure Chart. The sub-window on the bottom edits a single module on the chart.

Having arrived at the 'Edit' window, shown in Figure 3.12 (top), fields are analogous to the Structure Chart (.sc) file (Section 2.3.4).

The top of the window identifies the chart and sets up some page layout parameters. The next section allows default background colours to be assigned to modules by sub-project, which is useful if the code calls modules from libraries that are not considered part of the same project.

The final section of the main editing screen is the list of modules that are to appear on the chart. There are a sufficient number of fields associated with each module on the chart that a sub-window is used for editing them. The drop-down list of other modules on the chart (on the right of 'Sub-Program Calls') allows the selection of modules which are to be called by the current module.

Changing the 'Name' and saving creates a new Structure Chart. This is a quick method to create several similar charts.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.10 Manuals and Equations

Currently, no support is provided for TEXspec equations or manuals, but these items are present on the menus as an indication of future development.

3.11 Java ↔ Perl Interface

The underlying TEXspec technology is implemented as Perl scripts, but the user interface is a Java application. In order for the user to generate TEXspec products, the Java application must interface with the Perl scripts.

Both Perl and Java are relatively portable, but there is no portable interface between them defined in the Application Program Interface (API) of either. It is necessary, then, to define such an interface for TEXspec.

The interface could be implemented in several ways. It would be possible, for example, to set up an interprocess communication system [25, 26] between the GUI and a server application which would be responsible for running the TEXspec Perl scripts. Such a server application could be implemented in Perl in a portable manner and would be a stepping stone to future TEXspec development.

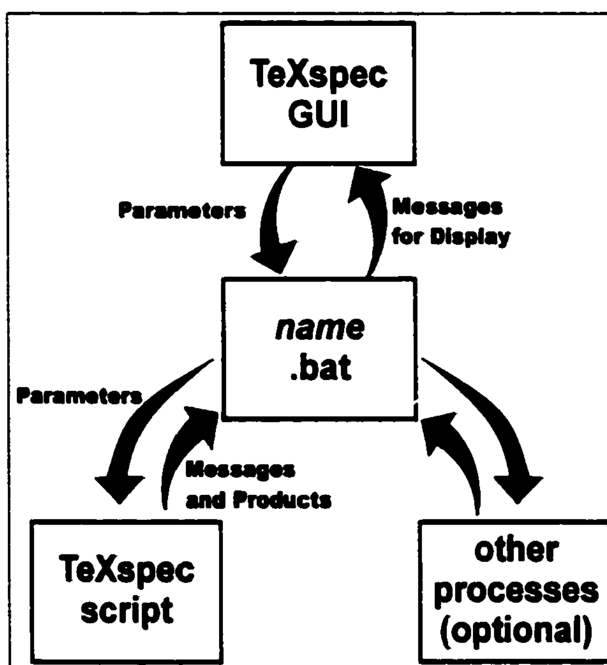


Figure 3.13: TEXspec Architecture for running Perl scripts from the Java GUI. The user asks for a listing to be generated which initiates '*name*.bat' to execute the TEXspec script, and optionally perform other functions.

For the sake of simplicity, however, the GUI uses the Java 'Runtime.exec()' function to execute a command, which is itself the name of a script. For each Perl script '*name*.pl', there exists a corresponding initializations script '*name*.bat' which the GUI can 'exec()' to run the Perl script, as illustrated in Figure 3.13.

The script name '*name*.bat', is selected to make the implementation as portable as possible. MS-DOS prefers scripts with such a name, and UNIX accepts it. Although a Macintosh implementation has not been written, no difficulty is foreseen.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The script based interface offers a further advantage. Since '*name*.bat' is typically a short script, it can be customized to perform other functions in addition to running the TEXspec Perl scripts. Since the TEXspec outputs are primarily LATEX files, it is convenient to run LATEX once the TEXspec script has run to successful completion. A viewer can then be initiated to show the product on the screen.

This is particularly useful in the case of '*designSpec*.bat', since in this case TEXspec produces a Noweb file as output. The script can continue processing to generate both the Design Specification and the corresponding code. The documentation can be displayed and the code can be further processed, including compilation. The sample '*designSpec*.bat', provided with TEXspec, executes the 'Floppy' [2] tool to reformat and provide a static analysis of the generated Fortran code.

The location of the scripts to be run (both interface and base TEXspec) is defined to the GUI using the same 'search list' arrangement used to locate other files. By modifying the search list, it is possible to override the default processing with revised scripts which reflect the current project, user preferences, or the particular job at hand.

Output from the processing of '*name*.bat' is displayed to the GUI user. The display is in three sections:

- Output, which includes both 'standard output' and 'standard error' listings.
- Errors, to reduce the possibility of error messages going unnoticed in voluminous 'standard output' and
- a button to interrupt the process or dismiss the display.

Figure 3.14 illustrates the format of the display.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

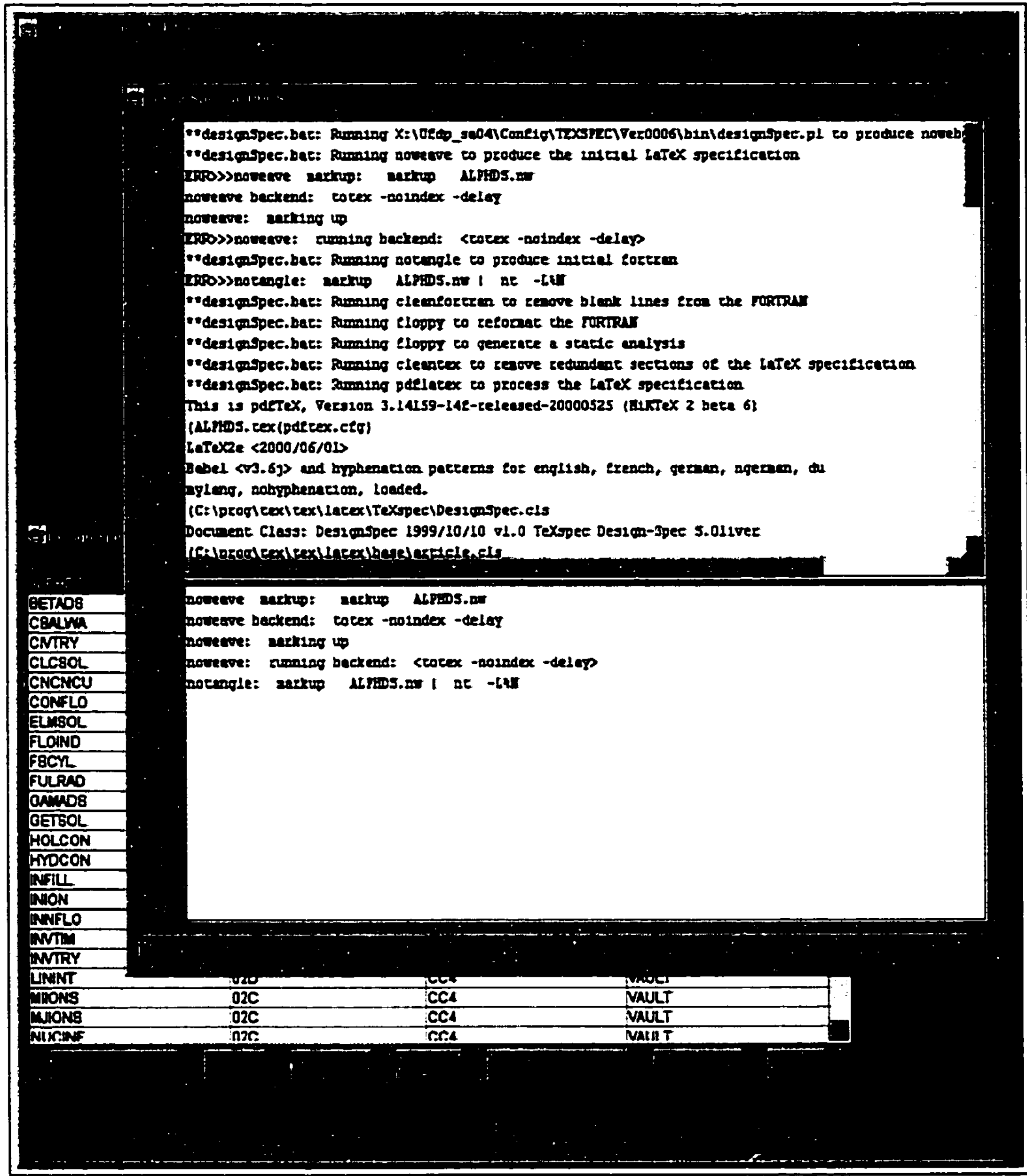


Figure 3.14: TEXspec script being run from the GUI. The 'errors' are any output directed to the 'standard error' output stream. Noweb sends some messages to this stream.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4 Conclusions

Prior to the development of TEXspec no CASE tool could be found which could simutaneously

- produce Yourdon/Demarco structured analysis documentation,
- support scientific and mathematical notations,
- enforce ownership of components,
- permit sharing of components,
- assemble large products from smaller components, and
- verify consistency between products.

TEXspec is a fully usable tool capable of producing highly presentable and reliable software documentation, featuring robust mathematical notation. Reuse of components and automatic checking between products reduces the chance of inconsistent documentation, which has been a major source of software defects in the past.

TEXspec satisfies the requirements specified in Section 2.1.1.

The TEXspec tool achieves the objective of offering automated support to assist developers of technical software who wish to comply with the CSA N286.7 standard [4]. Compliance is expected to become a requirement for licence applications to the CNSC.

4.1 Maintenance and Future Development

It should be noted that TEXspec development has been, to date, a one man show. If the product is to be developed in another manner, the following skills are essential to an understanding of the technical aspects of the implementation:

- Java, including Swing,
- PERL,
- LATEX, including the generation of 'class' files, and
- Noweb

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Fortunately, these skills are common and none is difficult to learn, with the possible exception of LaTeX 'class' files.

The following items are considered priorities for future development:

- Editors could be added to the GUI to handle equations and manuals.
- The parsing of the TeXspec files by the GUI is performed by an ad-hoc implementation based on the FSF regular expression parser 'regexp'. In fact, there exists a YACC-style parser for Java. JYACC could replace the current parsing. This would make the parsing code more compact and easier to modify or extend.
- Input file formats may be converted to a format which is easier to parse. For example, TeXspec may be a natural fit for Extensible Markup Language (XML). This would make processing of multiple line fields easier to process. Internal flags used to keep track of what field is being parsed could be eliminated.
- The configuration file is named as a 'resource file', which typically retains settings between runs. The configuration file might be one entry in a true resource file and could be loaded at invocation.
- The GUI support for the graphical products (Data Flow Diagrams and Structure Charts) could be based on editable graphics, or perhaps provide a 'preview' window. Having to process the file to see the format of the output is not optimal.
- More types of diagrams could be supported, including Object Oriented abstractions. Object Oriented technology from the ArgoUML [22] project might be reusable for this purpose.
- Data flow diagrams could support 'control' flows, as defined by Yourdon/DeMarco [5, 30]. This differentiates between flows that control the nature of the processing from flows containing data to be processed.
- Languages other than Fortran-77 could be supported.
- Some allowance for tracing between design and requirements could be provided. Currently, the most useful link between requirements and design is the mathematical specification of Design Data Dictionary entries, which may correspond to Requirements Data Dictionary entries, which allows a reader to associate variables in Design Specifications to terms in Process Specifications. It would be advantageous to allow a Design Specification to explicitly declare what requirement is being met.
- The TeXspec system could be divided into client and server portions, with traffic between them over a network.
- The system could allow installation of files into a configuration management system. Dependencies between files should be monitored from this system, and security would be enforced.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A Sample Data Flow Diagram

Figure A.2 details the input required to produce the Data Flow Diagram shown in Figure A.1. The syntax is discussed in Section 2.3.1. For convienence, the input has been divided into sections, delimited by a line of hashes. Note that this file would typically be generated and maintained through the GUI.

The first section contains the identification information common to all TEXspec components.

The second section indicates that the positions on the diagram are specified in inches, and that the 'Name' field in the Requirements Data Dictionary entries are to be used to label the flows. Alternatively, the 'long' name or mathematical symbol could be used.

The third section specifies the process 'bubbles' to appear on the chart. Note that processes 1 and 3 are specified to be 'atomic', indicating that they are associated with a Process Specification (Mini-spec), while process 2 is associated with a child diagram.

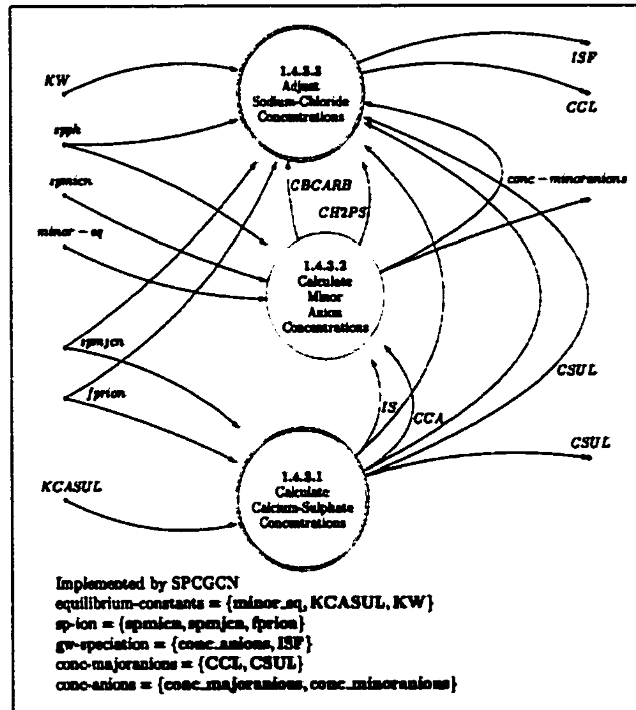


Figure A.1: Sample Data Flow Diagram.

The fourth section specifies the location of off-page connectors. This particular diagram employs a convention placing inputs on the left and outputs on the right, but this is not a requirement.

The largest section details the 'Flows' to appear on the diagram. Each 'Flow' in the diagram is defined with a Requirements Data Dictionary entry similar to Figure 2.5.

The final section contains notes to be placed on the diagram. This is often supplemented by notes generated by TEXspec to indicated the treatment of composite Requirements Data Dictionary entries.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
Title:          Determine Speciation of Groundwater
Version:        01B
Project:        INROC-LE
Submodel:       inroc
Author:         Ted Melnyk
Date:           Feb 22, 2000
Implementer:    Steve Oliver
ImplementDate:  Sep 29, 2000

################################################

Units:          inches
Labels:         Name

################################################

Process 1:      Calculate\\Calcium-Sulphate\\Concentrations
At:             3,1
atomic

Process 2:      Calculate\\Minor\\Anion\\Concentrations
At:             3.25,3

Process 3:      Adjust\\Sodium-Chloride\\Concentrations
At:             3,5
atomic

################################################

Connector:      Input NCASUL
At:             0.5,1

Connector:      Input fprion
At:             0.5,2

Connector:      Input spmjcn
At:             0.5,2.5

Connector:      Input minor_eq
At:             0.5,3.5

Connector:      Input spmicn
At:             0.5,4

Connector:      Input spph
At:             0.5,4.5

Connector:      Input KN
At:             0.5,5

Connector:      Output ISF
At:             6,5.5

Connector:      Output CCL
At:             6,5

Connector:      Output conc_minoranions
At:             6,3.95

Connector:      Output CSUL
At:             6,1.4

################################################

Flow:           NCASUL
From:           Input NCASUL
To:             Calculate\\Calcium-Sulphate\\Concentrations
Inflection:     -0.3
RelPos:         0
LabelOffset:    -0.15

Flow:           fprion
From:           Input fprion
To:             Calculate\\Calcium-Sulphate\\Concentrations
Inflection:     0.1
RelPos:         0.15
LabelOffset:    -0.15

Flow:           fprion
From:           Input fprion
To:             Adjust\\Sodium-Chloride\\Concentrations
Inflection:     -0.4
LabelOffset:    99

Flow:           spmjcn
From:           Input spmjcn
To:             Calculate\\Calcium-Sulphate\\Concentrations
Inflection:     0.3
RelPos:         0.12
LabelOffset:    -0.13

Flow:           spmjcn
From:           Input spmjcn
To:             Adjust\\Sodium-Chloride\\Concentrations
Inflection:     -0.2
LabelOffset:    99

Flow:           minor_eq
From:           Input minor_eq
To:             Calculate\\Minor\\Anion\\Concentrations
Inflection:     -0.2
RelPos:         0
LabelOffset:    -0.15

Flow:           spmicn
From:           Input spmicn
To:             Calculate\\Minor\\Anion\\Concentrations
Inflection:     -0.1
RelPos:         0
LabelOffset:    -0.15

Flow:           spph
From:           Input spph
To:             Calculate\\Minor\\Anion\\Concentrations
Inflection:     0.2
RelPos:         0
LabelOffset:    -0.15

Flow:           spph
From:           Input spph
To:             Adjust\\Sodium-Chloride\\Concentrations
Inflection:     -0.15
LabelOffset:    99

Flow:           KN
From:           Input KN
To:             Adjust\\Sodium-Chloride\\Concentrations
Inflection:     0.3
RelPos:         0
LabelOffset:    -0.15

Flow:           ISF
From:           Adjust\\Sodium-Chloride\\Concentrations
To:             Output ISF
Inflection:     0.3
RelPos:         1
LabelOffset:    0.15

Flow:           CCL
From:           Adjust\\Sodium-Chloride\\Concentrations
To:             Output CCL
Inflection:     0.3
RelPos:         1
LabelOffset:    0.15

Flow:           conc_minoranions
From:           Calculate\\Minor\\Anion\\Concentrations
To:             Output conc_minoranions
Inflection:     0.05
RelPos:         0.95
LabelOffset:    -0.25

Flow:           conc_minoranions
From:           Calculate\\Minor\\Anion\\Concentrations
To:             Adjust\\Sodium-Chloride\\Concentrations
Inflection:     -2
LabelOffset:    99

Flow:           CSUL
From:           Calculate\\Calcium-Sulphate\\Concentrations
To:             Output CSUL
Inflection:     0.2
RelPos:         1
LabelOffset:    -0.15

Flow:           CSUL
From:           Calculate\\Calcium-Sulphate\\Concentrations
To:             Adjust\\Sodium-Chloride\\Concentrations
Inflection:     -3
RelPos:         0.35
LabelOffset:    0.2

Flow:           CCA
From:           Calculate\\Calcium-Sulphate\\Concentrations
To:             Calculate\\Minor\\Anion\\Concentrations
Inflection:     -1.05
RelPos:         0.48
LabelOffset:    0.18

Flow:           CCA
From:           Calculate\\Calcium-Sulphate\\Concentrations
To:             Adjust\\Sodium-Chloride\\Concentrations
Inflection:     -2.5
LabelOffset:    99

Flow:           IS
From:           Calculate\\Calcium-Sulphate\\Concentrations
To:             Calculate\\Minor\\Anion\\Concentrations
Inflection:     -0.7
RelPos:         0.5
LabelOffset:    0.1

Flow:           IS
From:           Calculate\\Calcium-Sulphate\\Concentrations
To:             Adjust\\Sodium-Chloride\\Concentrations
Inflection:     -1.5
LabelOffset:    99

Flow:           CHCARB
From:           Calculate\\Minor\\Anion\\Concentrations
To:             Adjust\\Sodium-Chloride\\Concentrations
Inflection:     0.25
RelPos:         0.55
LabelOffset:    0.35

Flow:           CH2PS
From:           Calculate\\Minor\\Anion\\Concentrations
To:             Adjust\\Sodium-Chloride\\Concentrations
Inflection:     -0.6
RelPos:         0.35
LabelOffset:    -0.27

################################################

Notes:          Implemented by SPCGCM
```

Figure A.2: Input required to produce Figure A.1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

B Sample Design Specification

Figure B.3 details the input required to produce the Design Specification shown in Figure B.1. The syntax is discussed in Section 2.3.3. For convienence, the input has been divided into sections, delimited by a line of '%' characters.

B.1 Output

The header contains the identification information common to all TeXspec components. To keep track of the components used to assemble a Design Specification, or any TeXspec product, the LaTeX files generated by TeXspec contain commentary that identifies all referenced components, and the version of the TeXspec module that assembled them. A date-time stamp is visible above the header in Figure B.1 (in the upper left corner) to uniquely associate the LaTeX file with the associated product. By retaining the LaTeX file, it is possible to audit the content of any product. Figure B.2 shows the top of the LaTeX file associated with Figure B.1. Note the matching date stamps and the list of components, including version identification.

Below the header is the default code chunk << * >>. The code chunks that are represented by tables (<< argument >>, << local >>, etc.) are generated and are not identified in << * >> by obvious association. The TeXspec module *cleantex.pl* removes these code chunks from the Design Specification, since the tables contain a superset of the information in the associated code. Generated code chunks that are not represented by tables (<< interface >>, << description >>, etc.) are displayed using the usual Noweb notation.

User written code chunks, which are not referenced in other code chunks, are placed in << * >> in the order that they occur in the input. In this case << checkArrayBounds >>, << initialize >>, and << main >> are in this category.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CC4	INROC
Module: ALPHDS : Simulate fuel release from alpha radiolysis	Version 02H
Author: S.E. Oliver	Feb 23, 2001
Implementer: S.E. Oliver	Mar 16, 2001
Reviewer: T.W. Melnyk	Mar 16, 2001

Module components:

⟨*⟩≡

⟨interface⟩
⟨description⟩
⟨directives⟩
⟨include⟩
⟨argument⟩
⟨local⟩
⟨data⟩
⟨checkArrayBounds⟩
⟨initialize⟩
⟨main⟩

Description:

⟨description⟩≡

```
Simulate fuel release from alpha radiolysis.
```

Calling interface:

⟨interface⟩≡

```
      SUBROUTINE ALPHDS(CALTYP,NT,TIMSS,ALPHRE,CONTIN,OK)
```

Arguments:

Argument	*Long Name*	*Symbol*	*Units*	*Dimension*	*DataType*	*I/O*
CALTYP	call type: "TIMES" or "VALUES"		[]	*	character	I
NT	number of times in subseries		[]	scalar	integer	I/O
TIMSS	times for user time series		[a]	*	double	I/O
ALPHRE	release rate from alpha radiolysis	$\Lambda\, \dot{c}_{r}(t)$	[mol/a]	*	double	O
CONTIN	continuation flag		[]	scalar	boolean	O
OK	operations thus far ok flag		[]	scalar	boolean	O

Preconditions:

NT: none if CALTYP = "TIMES"
≥ 1 if CALTYP = "VALUES"
TIMSS: none if CALTYP = "TIMES"
≥ 0 for (1..NT) if CALTYP = "VALUES"

Postconditions:

NT: ≤ 6 + NOALPH if CALTYP = "TIMES"
unchanged if CALTYP = "VALUES"
TIMSS: ≥ 0 for (1..NT) if CALTYP = "TIMES"
unchanged if CALTYP = "VALUES"
ALPHRE: unset if CALTYP = "TIMES"
≥ 0 for (1..NT) if CALTYP = "VALUES"
CONTIN: = .TRUE.
OK: = .TRUE.

Figure B.1: Example Design Specification (1 of 5).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Constants (PARAMETER):

Constant	*Long Name*	*Units*	*DataType*	*Value*
MXUDOS	Max dose rate ts ents for radiolysis	[]	integer	20
TDELTA	duration of delta fn input	[a]	double	0.01

Shared (COMMON) variables:

Shared	*Long Name*	*Symbol*	*Units*	*Dimension*	*DataType*	*I/O*
AALPHA	Fit coefficient a for alpha radiolysis	a_α		scalar	double	I
ALFCOF	Scale factor for alpha dose	S_α		scalar	double	I
ALPHDO	alpha dose to used fuel surface	α	[Gy/a]	MXUDOS	double	I
ALPHTI	time values for alpha dose rate		[a]	MXUDOS	double	I
BALPHA	Fit coefficient b for alpha radiolysis	b_α	[]	scalar	double	I
CALPHA	Est variance alpha	s_α	[]	scalar	double	I
DALPHA	Number of data points for alpha dose fit	$1 + \frac{1}{n_\alpha}$	[]	scalar	double	I
EALPHA	Statistical parameter alpha radiolysis	$\sum(\log\alpha_i - \overline{\log\alpha})^2$	[]	scalar	double	I
FALPHA	Mean experimental alpha radiolysis	$\overline{\log\alpha}$	[]	scalar	double	I
NOALPH	num entries in alpha dose ts	n_α	[]	scalar	double	I
STDNOA	std normal variate for alpha dose rate	$N_n(0,1)$	[]	scalar	double	I
TCOOL	effective cooling time	t_c	[a]	scalar	double	I
USURFA	effective surface area	A	[m2]	scalar	double	I

Preconditions:

AALPHA: set
ALFCOF: > 0 for if CALTYP = "TIMES"
none if CALTYP = "VALUES"
ALPHDO: ≥ 0 for (1..NOALPH) if CALTYP = "TIMES"
none if CALTYP = "VALUES"
ALPHTI: ≥ 0 for (1..NOALPH) if CALTYP = "TIMES"
none if CALTYP = "VALUES"
BALPHA: set
CALPHA: set
DALPHA: set
EALPHA: set
FALPHA: set
NOALPH: 1 ≤ NOALPH ≤ MXUDOS
STDNOA: set
TCOOL: ≥ 0
USURFA: > 0

Local variables:

Local	*Long Name*	*Symbol*	*Units*	*Dimension*	*DataType*	*Note*
ALFDRL	dimensionless and factored alpha dose		[]	MXUDOS	double	save
ALFREL	relative alpha dose rate		[Gy/a]	scalar	double	
ALFTRL	dimless time for alpha dose rate		[]	MXUDOS	double	save
DOAFLG	DOALOG is calculated		[]	scalar	boolean	
DOALOG	log(predicted alpha dose rate)	$\log(\hat{\alpha})$	[]	scalar	double	
EXPONA	log(predicted alpha corrosion rate)	$\log\dot{c}_\alpha(t)$	[]	scalar	double	
I	general index		[]	scalar	integer	
J	general index		[]	scalar	integer	
MODNAM	module name		[]	6	character	
MSG	error message		[]	64	character	
REFRAA	relative alpha dissolution rate		[mol/(m2.a)]	scalar	double	
STOPP	signal to stop processing		[]	scalar	boolean	
TIMREL	relative time		[a]	scalar	double	

Data:

MODNAM: 'ALPHDS'

STOPP: TRUE

Figure B.1: Example Design Specification (2 of 5).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Module Preconditions

ALPHTI(1) ≤ TDELTA + TCOOL

This module is used by SYVAC3 to set up a time series. For each time series, it is first called once with CALTYP = "TIMES"; then it is called possibly many times with CALTYP = "VALUES" until the time series is complete.

Exceptions

If a value cannot be interpolated at any particular time because of bad dose-time data an error message is written using WERR

If CALTYP ≠ ("TIMES" or "VALUES") NT,TIMSS,ALPHRE and CONTIN are not set and an error message is written using WRERR

Summary

ALPHDS implements Data Flow Diagram processes 'Fuel Dissolution Rate Alpha', and scales to the surface area of the fuel (is part of 'Calculate Total Fuel Dissociation Rate').

The dose-time relationship is provided numerically as n_α ordered time-dose pairs. The final value provided is continued as a constant for all longer times. Linear interpolation on the logarithmically transformed values is used for all intermediate times. To avoid numerical problems with logarithms of small times, the dose is assumed to be zero for times smaller than the small time TDELTA.

The primary function is to implement the theory manual equations in the 'Degradation Rate of Fuel' section, for α-radiolysis, scaled to the fuel surface area. That is, we are computing $A\,\dot{c}_\alpha(t)$. Here, the theory manual notation would have $d = \alpha$ in $\dot{c}_d(t) = c_d(t)\,10^{V_d(t,1)\ldots}$.

This implementation generates a SYVAC3 time series, and is designed in accordance with the template provided with SYVAC3. The input and output arguments are defined by the template.

Check the numerical dose-time function that the user supplied in the input file (as sampled parameters). If too many data pairs have been supplied then write an error message.

(*checkArrayBounds*)≡

```
  C.....Check array bounds
        IF (NINT(NOALPH) .GT. MXUDOS) THEN
           MSG = 'FUEL DOSE VALUES OUTSIDE ARRAY BOUNDS '
           CALL WRERR(MODNAM,MSG,STOPP)
        END IF
```

[illegible]

Initialize local variables and the output argument "OK".

ALFREL, TIMREL and REFRAA are always unity, and are used to resolve physical units of the values. This assists the UNITCK (unit checker) static analysis tool.

(*initialize*)≡

```
  C.....Initialize
        OK = .TRUE.
        ALFREL = 1.D0
        REFRAA = 1.D0
        TIMREL = 1.D0
```

Figure B.1: Example Design Specification (3 of 5).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Determine the module flow based on the 'call type'. Signal an error if the call type is neither "TIMES" nor "VALUES".

⟨*main*⟩≡

```
       IF (CALTYP .EQ. 'TIMES') THEN
          (initialTimes)
       ELSE IF (CALTYP .EQ. 'VALUES' ) THEN
          (supplyValues)
       ELSE
          MSG = 'UNIDENTIFIABLE CALL TYPE '// CALTYP //
     1    ', SHOULD BE "TIMES" OR "VALUES"'
          CALL WRERR(MODNAM,MSG,STOPP)
       END IF

       RETURN
       END
```

Provide some times to initialize the time series. The input file contains (time,alpha-dose) pairs in parameters (ALPHTI,ALPHDO). These times are offset by the cooling time TCOOL. For initialization times use all the times on the dose-time function supplied and a number of times around TDELTA where a discontinuity occurs.

Also, initialize the dose values in the dose-time function. by scaling by the uncertainty factor ALFCOF. A single uncertainty is applied to the data for each simulation using the sampled parameter STDNOA, then used consistently (regardless of time or dose rate) throughout the simulation.

*****NOTE*** This code could be changed to redefine ALFDRL to contain the log(dose). This would remove some of the overhead for interpolation between points. Also, the interpolation itself could be performed by SYVAC3 if the dose-time function were represented as a time series.**

⟨*initialTimes*⟩≡

```
       NT = 6
       TIMSS(1) = TDELTA*1.01D0
       TIMSS(2) = TDELTA*1.000001D0
       TIMSS(3) = TDELTA*0.99D0
       TIMSS(4) = TDELTA*0.999999D0
       TIMSS(5) = TDELTA*0.5D0
       TIMSS(6) = TDELTA*0.1D0
       DO J = 1,NINT(NOALPH)
          IF ((ALPHTI(J)-TCOOL) .GE. 0.D0) THEN
             NT = NT+1
             TIMSS(NT) = ALPHTI(J)-TCOOL
          END IF
 C........Initialize the dose-time function by applying
 C........the uncertainty factor ALFCOF and normalizing to
 C........remove physical units
          ALFDRL(J) = ALPHDO(J)*ALFCOF/ALFREL
          ALFTRL(J) = ALPHTI(J)/TIMREL
       END DO
       CONTIN = .TRUE.
```

Figure B.1: Example Design Specification (4 of 5).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Compute corrosion rate due to α radiolysis at specified times.

SYVAC3 supplies times at which the corrosion rate is to be evaluated in TIMSS(1..NT). The calculated vales are returned in ALPHRE(1..NT).

The value of $\log \hat{c}_d(t) = \log c_d(t) - N_d(0,1)\, s_{\log c_d}$ is computed in local variable EXPONA.

Multiplying by the surface area yields the corrosion rate for an entire container.

(supplyValues)≡

```
      DO J = 1,NT
C........Convert used fuel dose rate to dissolution rate
         IF (TIMSS(J) .LE. TDELTA) THEN
            ALPHRE(J) = 0.D0
         ELSE
            (LogAlphaDoseAtTime)
            EXPONA = BALPHA + AALPHA*DOALOG -
     1               STDNOA*CALPHA*
     1                SQRT(DALPHA+EALPHA*(DOALOG-FALPHA)**2)
            ALPHRE(J) = REFRAA*USURFA*10.D0**EXPONA
         END IF
      END DO
```

Evaluate the log of the predicted alpha dose $\log(\hat{\alpha})$ at a particular time TIMSS(J).

(LogAlphaDoseAtTime)≡

```
      IF (TIMSS(J)+TCOOL .LT. ALPHTI(NINT(NOALPH))) THEN
         I = 1
         DOAFLG = .TRUE.
         DO WHILE ((I .LE. NINT(NOALPH)-1) .AND. DOAFLG)
C...........If time is greater than TDELTA determine log dose rate
            IF (ALPHTI(I). EQ. TIMSS(J)+TCOOL) THEN
               DOALOG = LOG10(ALFDRL(I))
               DOAFLG = .FALSE.
C...........Interpolate dose rate values on a LOG10 basis
            ELSE IF ((TIMSS(J)+TCOOL .GT. ALPHTI(I)) .AND.
     :      (TIMSS(J)+TCOOL .LT. ALPHTI(I+1))) THEN
               DOALOG = LOG10(ALFDRL(I))+
     :                  (LOG10(ALFDRL(I+1))-LOG10(ALFDRL(I)))/
     :                  (LOG10(ALFTRL(I+1))-LOG10(ALFTRL(I)))*
     :                   (LOG10((TIMSS(J)+TCOOL)/TIMREL)-
     :                    LOG10(ALFTRL(I)))
               DOAFLG = .FALSE.
            END IF
            I = I+1
         END DO
         IF (DOAFLG) THEN
C..............Values cannot be interpolated
               MSG = 'FUEL DOSE VALUES CANNOT BE INTERPOLATED '
               CALL WRERR(MODNAM,MSG,STOPP)
         END IF
      ELSE
         DOALOG = LOG10(ALFDRL(NINT(NOALPH)))
      END IF
```

(include)≡

```
C
      INCLUDE 'MXUDOS.INC'
      INCLUDE 'SPALPH.INC'
      INCLUDE 'SPRADI.INC'
      INCLUDE 'TDELTA.INC'
C
```

(directives)≡

```
      IMPLICIT NONE
C
```

Figure B.1: Example Design Specification (5 of 5).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
% ===> this file was generated automatically by noweave --- better not edit it
%...  output generated by X:\Ufdp_sa04\Config\TEXSPEC\Ver0006\bin\designSpec.pl on Fri Mar 16 14:08:40 2001
%...                component  version
%...        X:\Ufdp_sa04\Config\TEXSPEC\Ver0006\bin\designSpec.pl  02F
%...               designspec  02E
%...        W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/cc4/alfdrl.ddd  01B
%...        W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/cc4/alfrel.ddd  01B
%...        W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/cc4/alftrl.ddd  01B
%...        W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/cc4/alphti.ddd  01B
%...        W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/cc4/contin.ddd  01B
%...        W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/cc4/deaflg.ddd  01B
%...        W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/cc4/i.ddd  01B
%...        W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/cc4/j.ddd  01B
%...        W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/cc4/modnam.ddd  01B
%...        W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/cc4/msg.ddd  01B
%...        W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/cc4/mrudos.ddd  01B
%...        W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/cc4/nt.ddd  01B
%...        W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/cc4/ok.ddd  01B
%...        W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/cc4/refraa.ddd  01B
%...        W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/cc4/stopp.ddd  01B
%...        W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/cc4/tdelta.ddd  01C
%...        W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/cc4/timrel.ddd  01B
%...        W:/Mba_shr/IPA/Imp/CC402/V1t/V0204/dictionary/cc4/times.ddd  01B
%...        x:/ufdp_sa04/config/TheorMan/ver0003/Equations/predictedCorrlin.teq  01A
%...        x:/ufdp_sa04/config/TheorMan/ver0003/Equations/predictedCorrlog.teq  01A
%...        x:/ufdp_sa04/config/design/ver0003/DesignDD/aalpha.ddd  01C
%...        x:/ufdp_sa04/config/design/ver0003/DesignDD/alfcof.ddd  01B
%...        x:/ufdp_sa04/config/design/ver0003/DesignDD/alphdo.ddd  01B
%...        x:/ufdp_sa04/config/design/ver0003/DesignDD/alphre.ddd  01D
%...        x:/ufdp_sa04/config/design/ver0003/DesignDD/balpha.ddd  01E
%...        x:/ufdp_sa04/config/design/ver0003/DesignDD/calpha.ddd  01C
%...        x:/ufdp_sa04/config/design/ver0003/DesignDD/caltyp.ddd  01C
%...        x:/ufdp_sa04/config/design/ver0003/DesignDD/dalpha.ddd  01C
%...        x:/ufdp_sa04/config/design/ver0003/DesignDD/dealog.ddd  01D
%...        x:/ufdp_sa04/config/design/ver0003/DesignDD/ealpha.ddd  01E
%...        x:/ufdp_sa04/config/design/ver0003/DesignDD/expona.ddd  01C
%...        x:/ufdp_sa04/config/design/ver0003/DesignDD/falpha.ddd  01C
%...        x:/ufdp_sa04/config/design/ver0003/DesignDD/noalph.ddd  01B
%...        x:/ufdp_sa04/config/design/ver0003/DesignDD/stdnoa.ddd  01B
%...        x:/ufdp_sa04/config/design/ver0003/DesignDD/tcool.ddd  01C
%...        x:/ufdp_sa04/config/design/ver0003/DesignDD/usurfa.ddd  01C
\renewcommand{\familydefault}{cmss}
\newcommand{\nwstartdeflinemarkup}{}
\newcommand{\nwenddeflinemarkup}{}
\documentclass{DesignSpec}
\usepackage{noweb}
\pagestyle{empty}

\productID{Fri Mar 16 14:08:40 2001}
\project{CC4}
\submodel{INROC}
\title{ALPHDS : Simulate fuel release from alpha radiolysis}
\author{S.E. Oliver}
\version{02E}
\date{Feb 23, 2001}
\implementer{S.E. Oliver}
\implementDate{Mar 16, 2001}
\reviewer{T.W. Malnyk}
\reviewDate{Mar 16, 2001}

\begin{document}
\maketitle \thispagestyle{empty}

\setlength{\parindent}{0in}
\settoheight{\parskip}{X}
\mdseries
\newcommand{\sep}{\begin{picture}(100,20)(0,0) \put(0,10){\line(100,0){100}}\end{picture}\newline}
\vbox{
\noindent \bfseries\LARGE Module components:
\mdseries \normalsize
```

Figure B.2: Portion of the LaTeX file generated by TeXspec/Noweb from the Design Specification file listed in Figure B.3.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

B.2 Input

Figure B.3 shows the input Design Specification file used to generate Figure B.1. Other input information was extracted from the Design Data Dictionary, as indicated in Figure B.2. Note that this file would typically be generated and maintained through the GUI.

The first section contains the identification information common to all TEXspec components.

The second section provides a description for the module to be placed in both the code and the Design Specification. For the code, Fortran comment characters (a 'C' in column 1) are added.

The third lists the arguments to the module, in the order that they are to occur in the interface. Direction of data flow must be stated. Preconditions and postconditions are optional, and are added to any conditions in the Design Data Dictionary.

The fourth section lists global (COMMON) variables. The order does not impact any products, but alphabetical order is often easier to read. Direction of data flow, preconditions and postconditions are similar to the arguments.

The fifth section lists the local variables and any initializing data. Note the variable 'MSG' for which the dimension in the Design Data Dictionary has been overridden.

The next section lists constants used in the module. Values are extracted from the Design Data Dictionary.

The next several sections are free form LaTeX, which is processed to the commentary associated with the first code chunk.

The remaining sections are the user supplied code chunks.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
Module:         ALPHDS
Version:        021
Project:        CC4
Submodel:       INROC
Author:         S.E. Oliver
Date:           Feb 23, 2001
Implementer:    S.E. Oliver
ImplementDate:  Mar 22, 2001
Reviewer:       T.W. Melnyk
ReviewDate:     Mar 16, 2001
LongName:       Simulate fuel release from alpha radiolysis
Language:       FORTRAN77,SUBROUTINE
Standard:       none

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

<<description>>=
Simulate fuel release from alpha radiolysis.
@ %def description

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Argument:       CALTYP
Flow:           input

Argument:       NT
Flow:           input,output
Precondition:   none if CALTYP = "TIMES"
                $\geq$ 1 if CALTYP = "VALUES"
Postcondition:  $\leq$ 6 + NOALPH if CALTYP = "TIMES"
                unchanged if CALTYP = "VALUES"

Argument:       TIMES
Flow:           input,output
Precondition:   none if CALTYP = "TIMES"
                $\geq$ 0 for (1,,NT) if CALTYP = "VALUES"
Postcondition:  $\geq$ 0 for (1,,NT) if CALTYP = "TIMES"
                unchanged if CALTYP = "VALUES"

Argument:       ALPHRE
Flow:           output
Postcondition:  unset if CALTYP = "TIMES"
                $\geq$ 0  for (1,,NT) if CALTYP = "VALUES"
Argument:       CONTIN
Flow:           output
Postcondition:  = .TRUE.

Argument:       OK
Flow:           output
Postcondition:  = .TRUE.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Shared:         AALPHA
Flow:           input
Precondition:   set

Shared:         ALFCOF
Flow:           input
Precondition:   > 0 for if CALTYP = "TIMES"
                none if CALTYP = "VALUES"

Shared:         ALPHDO
Flow:           input
Precondition:   $\geq$ 0 for (1,,NOALPH) if CALTYP = "TIMES"
                none if CALTYP = "VALUES"

Shared:         ALPHTI
Flow:           input
Precondition:   $\geq$ 0 for (1,,NOALPH) if CALTYP = "TIMES"
                none if CALTYP = "VALUES"

Shared:         BALPHA
Flow:           input
Precondition:   set

Shared:         CALPHA
Flow:           input
Precondition:   set

Shared:         DALPHA
Flow:           input
Precondition:   $= 1 + {1\over(NOALPH)}$

Shared:         EALPHA
Flow:           input

Shared:         FALPHA
Flow:           input

Shared:         NOALPH
Flow:           input

Shared:         STDNOA
Flow:           input
Precondition:   set

Shared:         TCOOL
Flow:           input
Precondition:   $\geq$ 0

Shared:         USURFA
Flow:           input

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure B.3: Input required to produce Figure B.1 (1 of 4).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
Local:          ALFDRL
Save

Local:          ALFREL

Local:          ALFTRL
Save

Local:          DOAFLG

Local:          DOALOG

Local:          EXPONA

Local:          I

Local:          J

Local:          MODNAM
Data:           'ALPHDS'

Local:          MSG
Dimension:      64

Local:          REFRAA

Local:          STOPP
Data:           .TRUE.

Local:          TIMREL

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

Constant:       MXUDOS

Constant:       TDELTA

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
\sep
\bf(\underline(Module Preconditions))

$\mathrm(ALPHTI(1)) \le \mathrm(TDELTA) + \mathrm(TCOOL)$

This module is used by SYVAC3 to set up a time series.
For each time series, it is first called once with
CALTYP = "TIMES"; then it is called possibly many times with
CALTYP = "VALUES" until the time series is complete.

\sep
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

\Bf(\underline(Exceptions))

if a value cannot be interpolated at any particular time because
 of bad dose-time data an error message is written using WERR

if CALTYP $\ne$ ("TIMES" or "VALUES")
    NT,TIMES,ALPHRE and CONTIN are not set and
    an error message is written using WRERR

\sep
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

\bf(\underline(Summary))

ALPHDS implements Data Flow Diagram processes
'Fuel Dissolution Rate Alpha',
and scales to the surface area of the fuel (ie part of
'Calculate Total Fuel Dissociation Rate').

The dose-time relationship is provided numerically as $\math(NOALPH)$
ordered time-dose pairs.  The final value provided is continued as a
constant for all longer times.  Linear interpolation on the
logarithmically transformed values is used for all intermediate times.
To avoid numerical problems with logarithms of small times, the dose
is assumed to be zero for times smaller than the small time TDELTA.

The primary function is to implement the theory
manual equations in the 'Degradation Rate of Fuel' section, for
($\alpha$)-radiolysis, scaled to the fuel surface area.
That is, we are computing $\math(ALPHRE)$.  Here,
the theory manual notation would have $d \equiv \alpha$ in
$\includeEquation(predictedCorrlin)$.

This implementation generates a SYVAC3 time series, and is
designed in accordance with the template provided with SYVAC3.
The input and output arguments are defined by the template.

\sep
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

Check the numerical dose-time function that the user supplied in
the input file (as sampled parameters).
If too many data pairs have been supplied then write an error message.
<<checkArrayBounds>>=
C.....Check array bounds
      IF (NINT(NOALPH) .GT. MXUDOS) THEN
         MSG = 'FUEL DOSE VALUES OUTSIDE ARRAY BOUNDS '
         CALL WRERR(MODNAM,MSG,STOPP)
      END IF      @ %def checkArrayBounds

\sep
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

Figure B.3: Input required to produce Figure B.1 (2 of 4).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
Initialize local variables and the output argument "OK".

ALFREL, TIMREL and REFRAA are always unity, and are used to
resolve physical units of the values.  This assists the
UNITCK (unit checker) static analysis tool.
<<initialize>>=
C.....Initialize
      OK = .TRUE.
      ALFREL = 1.D0
      REFRAA = 1.D0
      TIMREL = 1.D0
@ %def initialize

\sep
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Determine the module flow based on the 'call type'.
Signal an error if the call type is neither "TIMES" nor "VALUES".
<<main>>=
      IF (CALTYP .EQ. 'TIMES') THEN
         <<initialTimes>>
      ELSE IF (CALTYP .EQ. 'VALUES' ) THEN
         <<supplyValues>>
      ELSE
         MSG = 'UNIDENTIFIABLE CALL TYPE '// CALTYP //
     1   ', SHOULD BE "TIMES" OR "VALUES"'
         CALL WRERR(MODNAM,MSG,STOPP)
      END IF

      RETURN
      END
@ %def main

\sep
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Provide some times to initialize the time series.
The input file contains (time,alpha-dose) pairs in parameters
(ALPHTI,ALPHDO).  These times are offset by the cooling time TCOOL.
For initialization times use all the times on the dose-time
function supplied and a number of times around TDELTA where
A discontinuity occurs.

Also, initialize the dose values in the dose-time function.
by scaling by the uncertainty factor ALFCOF.
A single uncertainty is applied to the data for each simulation
using the sampled parameter STDNOA,
then used consistently (regardless of time or dose rate)
throughout the simulation.
```

```
***NOTE*** This code could be changed to redefine
ALFDRL to contain the log(dose).  This would
remove some of the overhead for interpolation between points.
Also, the interpolation itself could be performed by SYVAC3
if the dose-time function were
represented as a time series.
<<initialTimes>>=
      NT = 6
      TIMSS(1) = TDELTA*1.01D0
      TIMSS(2) = TDELTA*1.000001D0
      TIMSS(3) = TDELTA*0.99D0
      TIMSS(4) = TDELTA*0.999999D0
      TIMSS(5) = TDELTA*0.5D0
      TIMSS(6) = TDELTA*0.1D0
      DO J = 1,NINT(NOALPH)
         IF ((ALPHTI(J)-TCOOL) .GE. 0.D0) THEN
            NT = NT+1
            TIMSS(NT) = ALPHTI(J)-TCOOL
         END IF
C........Initialize the dose-time function by applying
C........the uncertainty factor ALFCOF and normalizing to
C........remove physical units
         ALFDRL(J) = ALPHDO(J)*ALFCOF/ALFREL
         ALFTRL(J) = ALPHTI(J)/TIMREL       END DO
      CONTIN = .TRUE.
@ %def initialTimes

\sep
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Compute corrosion rate due to $\alpha$ radiolysis
at specified times.

SYVAC3 supplies times at which
the corrosion rate is to be
evaluated in TIMSS(1..NT).
The calculated vales are returned in ALPHRE(1..NT).

The value of $\includeEquation{predictedCorrlog}$
is computed in local variable EXPONA.

Multiplying by the surface area yields the corrosion
rate for an entire container.
<<supplyValues>>=
      DO J = 1,NT
C........Convert used fuel dose rate to dissolution rate
         IF (TIMSS(J) .LE. TDELTA) THEN
            ALPHRE(J) = 0.D0
         ELSE
            <<LogAlphaDoseAtTime>>
            EXPONA = BALPHA + AALPHA*DOALOG +
     1               STDNOA*CALPHA*
     1                SQRT(DALPHA+EALPHA*(DOALOG-FALPHA)**2)
            ALPHRE(J) = REFRAA*USURFA*10.D0**EXPONA
         END IF
      END DO
@ %def supplyValues
```

Figure B.3: Input required to produce Figure B.1 (3 of 4).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
\sep
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Evaluate the log of the predicted alpha dose
$\math{DOALOG}$ at a particular time TIMSS(J).
<<LogAlphaDoseAtTime>>=
      IF (TIMSS(J)+TCOOL .LT. ALPHTI(NINT(NOALPH))) THEN
         I = 1
         DOAFLG = .TRUE.
         DO WHILE ((I .LE. NINT(NOALPH)-1) .AND. DOAFLG)
C...........If time is greater than TDELTA determine log dose rate
            IF (ALPHTI(I). EQ. TIMSS(J)+TCOOL) THEN
               DOALOG = LOG10(ALFDRL(I))
               DOAFLG = .FALSE.
C...........Interpolate dose rate values on a LOG10 basis
            ELSE IF ((TIMSS(J)+TCOOL .GT. ALPHTI(I)) .AND.
     1                  (TIMSS(J)+TCOOL .LT. ALPHTI(I+1))) THEN
               DOALOG = LOG10(ALFDRL(I))+
     1                   (LOG10(ALFDRL(I+1))-LOG10(ALFDRL(I)))/
     1                   (LOG10(ALFTRL(I+1))-LOG10(ALFTRL(I)))*
     1                    (LOG10((TIMSS(J)+TCOOL)/TIMREL)-
     1                     LOG10(ALFTRL(I)))
               DOAFLG = .FALSE.
            END IF
            I = I+1
         END DO
         IF (DOAFLG) THEN
C..............Values cannot be interpolated
               MSG = 'FUEL DOSE VALUES CANNOT BE INTERPOLATED '
               CALL WRERR(MODNAM,MSG,STOPP)
         END IF
      ELSE
         DOALOG = LOG10(ALFDRL(NINT(NOALPH)))
      END IF
@ %def LogAlphaDoseAtTime
\sep
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\bibliographystyle{aecl}
\bibliography{inroc}
```

Figure B.3: Input required to produce Figure B.1 (4 of 4).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

C Sample PERL Script

Figure C.1 lists one of the smaller TEXspec PERL modules '*minispec.pl*'. This listing is intended to illustrate the coding design and style used in all such modules.

TEXspec contains are over 8,000 lines of PERL over 7 modules.

```
#!/usr/bin/perl -w
#------------------------------------------------------------- miniSpec.pl
#  TeXspec routine to process a "mini-spec" specification
#  to LaTeX compliant input file
#
#  usage:  miniSpec.pl processNumber < miniSpecFile.ms
#
#      2000 Oct 11   Ver 01F    R. Oliver
#
#-------------------------------------------------------------------------
use Cwd;

#... MAKE SURE that this is set to the script version
$version{$0}  = "01F";   #... keep track of the versions of all components

die "usage: $0 processNumber < minispec.ms" if @ARGV != 1;  #... one input parameter

#... other global variables: %composite, %compositeUsed
$nCompositeUsed = 0;  #... global variable counts how many composite flows

#... declare and initialize local variables  (in case this becomes a subroutine later)
my($author) = "author not defined";
my(@bibligraphy);   #... abridged bibliography
my($bibTeX) = 0;  #... flag that bibTeX bibliography processing will be used
my(%cite);   #... citations referenced with "\cite"
my($currentSection) = "none";
my($currentSectionName) = "";
my($date) = "date not defined";
my(@dfdIn);             #... input flows for this process on the data flow diagram
my(@dfdInType);         #... type of flow (static/temporal) on the data flow diagram
my(@dfdOut);            #... output flows for this process on the data flow diagram
my(@dfdOutType);        #... type of flow (static/temporal) on the data flow diagram
my($diagramNumber) = $ARGV[0];
my($formatOptions) = "";
my(@freeForm);          #... user supplied TeX
my($implementer) = "implementer not defined";
my($implementDate) = "implementDate not defined";
my(@inFlow);                                        #... input data flows
my($nDfdIn) = 0;       #... number of input flows for this process on the data flow diagram
my($nDfdOut) = 0;      #... number of output flows for this process on the data flow diagram
my($nFreeFormBeforeIO) = 0;
my($nInFlow) = 0;                                   #... number of input data flows
my($nOutFlow) = 0;                                  #... number of output data flows
my(@outFlow);                                       #... output data flows
my($parentMsg) = "";
my($processName) = "";
my($processNumber) = $ARGV[0];
my($project) = "project not defined";
my($pwd) = cwd();   #... present working directory
my($reviewer) = "reviewer not defined";
my($reviewDate) = "reviewDate not defined";
my($runtime);
my($submodel) = "submodel not defined";
my(%usePackage);   #... TeX packages that are used
my($versionID) = "version number not defined";

&readRC;

$runtime = localtime();
```

Figure C.1: Example PERL Module (1 of 9).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
#... read the input file
while (<STDIN>)
   if(/^Process\:\s*(\S.*)$/)          #...  Process:
      $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
      $processName = $1;
      $currentSection = "none";
      $currentSectionName = "";
     elsif(/^Version\:\s*(\S.*)$/)          #...  Version:
      $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
      $versionID = $1;
      $version "miniSpec"  = $versionID;
      $currentSection = "none";
      $currentSectionName = "";
     elsif(/^Project\:\s*(\S.*)$/)          #...  Project:
      $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
      $project = $1;
      $currentSection = "none";
      $currentSectionName = "";
     elsif(/^Submodel\:\s*(\S.*)$/)          #...  Submodel:
      $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
      $submodel = $1;
      $currentSection = "none";
      $currentSectionName = "";
     elsif(/^Author\:\s*(\S.*)$/)          #...  Author:
      $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
      $author = $1;
      $currentSection = "none";
      $currentSectionName = "";
     elsif(/^Date\:\s*(\S.*)$/)          #...  Date:
      $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
      $date = $1;
      $currentSection = "none";
      $currentSectionName = "";
     elsif(/^Implementer\:\s*(\S.*)$/)          #...  Implementer:
      $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
      $implementer = $1;
      $currentSection = "none";
      $currentSectionName = "";
     elsif(/^ImplementDate\:\s*(\S.*)$/)          #...  ImplementDate:
      $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
      $implementDate = $1;
      $currentSection = "none";
      $currentSectionName = "";
     elsif(/^Reviewer\:\s*(\S.*)$/)          #...  Reviewer:
      $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
      $reviewer = $1;
      $currentSection = "none";
      $currentSectionName = "";
     elsif(/^ReviewDate\:\s*(\S.*)$/)          #...  ReviewDate:
      $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
      $reviewDate = $1;
      $currentSection = "none";
      $currentSectionName = "";
     elsif(/^Format\:\s*(\S.*)$/)          #...  Format:
      $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
      $formatOptions = $1;
      #... check for optional enclosing square brackets
      $formatOptions = "\[ ".$formatOptions
         if ! $formatOptions =~ /^\s*\ /;
      $formatOptions = $formatOptions."\]"
         if ! $formatOptions =~ /\ \s*$/;
     elsif(/^\\inputFlow\ ( \w\s|+)\ \s*$/)          #...  inputFlow:
      $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
      $inFlow $nInFlow++| = $1;
      $currentSection = "none";
      $currentSectionName = "";
      $nFreeFormBeforeIO = (@freeForm)-1 if $nFreeFormBeforeIO==0;
     elsif(/^\\outputFlow\ ( \w\s|+) \s*$/)          #...  outputFlow:
      $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
      $outFlow $nOutFlow++| = $1;
      $currentSection = "none";
      $currentSectionName = "";
      $nFreeFormBeforeIO = (@freeForm)-1 if $nFreeFormBeforeIO==0;
     else
      if($_ ne "")
         while(/( '\\|)\%/)     #... remove comments
            $_ = $`.$1;


      if($_ ne "")
         while(/\\includeEquation\ ( ^\ |+)\ /)     #... laXspec equation insertion
            my($before) = $`;
            my($after) = $';
            my($eqationName) = $1;
            my($fnEqn) = inputFile("teq","$eqationName\.teq");
            $version $fnEqn  = "not specified";
            open(EQN,"<$fnEqn") || die "could not open equation \<$fnEqn\> for input";
            while(<EQN>)
               chomp;
               #... try to get the version id correct
               my($save) = $_;
               tr/a-z/A-Z/;
               if(     /^\%.*VERSION\s+(\S+)\s/ || /^\%.*VERSION\s+(\S+)$/
                   || /^\%.*VER\s+(\S+)\s/     || /^\%.*VER\s+(\S+)$/ )
                  $_ = $1;
                  if(/^(\d+)( A-Z| )$/)
                     my($newVer) = $1;
                     my($newSubVer) = $2;
                     $_ = $version $fnEqn ;
                     if(/^(\d+)( A-Z| )$/)
                        my($oldVer) = $1;
                        my($oldSubVer) = $2;
                        $version $fnEqn  = $newVer.$newSubVer
                           if (    ($newVer>$oldVer)
                                || (   ($newVer==$oldVer)
                                    && ($newSubVer gt $oldSubVer)));
                     else
                        $version $fnEqn  = $newVer.$newSubVer;

                  else
                     $version $fnEqn  = $_ if $version $fnEqn  eq "not specified";

               else
               $  = $save;
               #... end of version id
               $_ = "" if /^\s*\%/;
               if(eof(EQN))
                  $_ = $before.$_.$after;
                  $freeForm @freeForm| = notQuiteFreeForm($_) if ($_ && !/^\s*\%/);
                  $_ = "";
```

Figure C.1: Example PERL Module (2 of 3).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
          else
           my($str) = notQuiteFreeForm($before,$_);
           $freeForm @freeForm| = $str if length($str)>0;

         $before = "";

      close(EQN);
      last if !$_;

  else
if($_)
   while(/\\includeTeX\s*\ ( ^\ ^\ |+)\ /)     #... TeX insertion
      my($inc) = $1;
      my($before) = $`;
      my($after) = $';
      $inc =~ s/\s//g;
      $fnInc = inputFile("tex",$inc);
      $fnInc = inputFile("tex",$inc,"\.tex") if ! -e $fnInc;
      $fnInc = inputFile("ms",$inc) if ! -e $fnInc;
      $fnInc = inputFile("ms",$inc,"\.tex") if ! -e $fnInc;
      die "could not find \\insertTeX \"$inc\"" if ! -e $fnInc;
      $version $fnInc  = "not specified";
      open(INC,"<$fnInc") || die "could not open \"$fnInc\" for input";
      while(<INC>)
         #... try to get the version id correct
         my($save) = $_;
         tr/a-z/A-Z/;
         if(      /^\s.*VERSION\s+(\S+)\s/ || /^\s.*VERSION\s+(\S+)$/
            || /^\s.*VER\s+(\S+)\s/      || /^\s.*VER\s+(\S+)$/ )
            $_ = $1;
            if(/^(\d+)( A-Z|)$/)
               my($newVer) = $1;
               my($newSubVer) = $2;
               $_ = $version $fnInc ;
               if(/^(\d+)( A-Z|)$/)
                  my($oldVer) = $1;
                  my($oldSubVer) = $2;
                  $version $fnInc  = $newVer.$newSubVer
                     if (   ($newVer>$oldVer)
                         || (    ($newVer==$oldVer)
                              && ($newSubVer gt $oldSubVer)));
               else
                  $version $fnInc  = $newVer.$newSubVer;

            else
             $version $fnInc  = $_ if $version $fnInc  eq "not specified";

          else  ;
         $_ = $save;
         #... end of version id
         if(eof(INC))
            chomp;
            $_ = $before.$_.$after;
            $freeForm @freeForm| = notQuiteFreeForm($_) if ($_ && !/^\s*\s/);
            $_ = "";
          else
            $  = $before.$ ;
            $freeForm @freeForm| = notQuiteFreeForm($_) if ($_ && !/^\s*\s/);
            $before = "";

                                        close(INC);
                                        last if !$_;

                                      $freeForm @freeForm| = notQuiteFreeForm($_) if ($_ && !/^\s*\s/);
                                     else
                                      $freeForm @freeForm| = " ";


                                   #... input has been read

                              #... check to make sure that minimal input was provided
                              die "Author: required" if $author eq "author not defined";
                              die "Date: required" if $date eq "date not defined";
                              die "Project: required" if $project eq "project not defined";
                              die "Submodel: required" if $submodel eq "submodel not defined";
                              die "Version: required" if $versionID eq "version number not defined";

                              #... implementer defaults to author
                              if ($implementer eq "implementer not defined")
                                 $implementer = $author;
                                 $implementDate = $date;


                              #... sort the flows in alphabetical order
                              @inFlow  = sort  lc($a) cmp lc($b)  @inFlow;   #... sort without regard to case
                              @outFlow = sort  lc($a) cmp lc($b)  @outFlow;  #... sort without regard to case

                              #... look for parent DFD
                              $_ = $processNumber;
                              if(/(.*)\..+$/)
                                 $fn = "dfd\ $1";
                                 $fn =~ tr/\./\_/;
                                 $fn = inputFile("dfd",$fn,".tex");
                                 if(open(IN,"<$fn"))
                                    my($inProcessBubbles) = 0;
                                    my($processBubbleFound) = 0;
                                    my($inDataFlows) = 0;
                                    while(<IN>)
                                       if(/^%\.\.\.  output generated by dfd.pl on (.*)/)
                                          $version $fn  = $1;

                                       if(/^  %\.\.\. place process bubbles$/)   $inProcessBubbles = 1;
                                       if(/^  %\.\.\. place data stores$/)   $inProcessBubbles = 0;
                                       if(/^  %\.\.\. draw data flows$/)   $inDataFlows = 1;
                                       if(/^  %\.\.\. end of data flows$/)   $inDataFlows = 0;
                                       if($inProcessBubbles==1)
                                          if(/^    %\.\.\. \|process\: (.*)  \|name\: (.*)  \|type\: (.*)$/)
                                             my($dfdProcessNumber) = $1;
                                             my($dfdProcessName) = $2;
                                             my($dfdProcessType) = $3;
                                             if($dfdProcessNumber eq $processNumber)
                                                $processBubbleFound = 1;
                                                if($dfdProcessName ne $processName)
                                                   printf STDERR
                                                      "*WARNING: matched DFD process <%s> to minispec process <%s>\n",
                                                      $dfdProcessName, $processName;

                                             if($dfdProcessType ne "atomic")
                                                printf STDERR "*ERROR: process <%s> is not atomic\n",
```

Figure C.1: Example PERL Module (3 of 9).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
                  $dfdProcessName;
               }
            }
         }
      }
   if($inDataFlows) {
      if(/^     $\.\.\. \|from\: (.*)  \|to\: (.*)  \|name\: (.*)  \|type\: (.*)$/) {
         my($from) = $1;
         my($to) = $2;
         my($nam) = $3;
         my($typ) = $4;
         #... flows into this process on parent diagram
         if($to eq "process ".$processNumber) {
            $dfdIn[$nDfdIn] = $nam;
            $dfdInType[$nDfdIn++] = $typ;
         }
         #... flows out of this process on parent diagram
         if($from eq "process ".$processNumber) {
            $dfdOut[$nDfdOut] = $nam;
            $dfdOutType[$nDfdOut++] = $typ;
         }
      }
   }
} #... finished reading the data flow diagram
close(IN);

#... input flows ont the DFD should be present in the minispec
for($i=0 ; $i<$nDfdIn ; $i++) {
   my(@component) = get_all_components($dfdIn[$i], $project);
   my($flowOK) = 0;
   my($flow);
   foreach $flow (@inFlow) {
      my($component);
      foreach $component (@component) {
         if( $flow eq $component ) {
               $flowOK = 1;
               last;
         }
      }
      if($flowOK) { last; }
   }
   if(!$flowOK) {
      printf STDERR
         "*WARNING: input flow %s from parent not found in minispec\n", $dfdIn[$i];
      $parentMsg = $parentMsg,
         "%..  *WARNING: input flow ",$dfdIn[$i].
         " from parent not found in minispec\n";
   }
}

#... output flows ont the DFD should be present in the minispec
for($i=0 ; $i<$nDfdOut ; $i++) {
   my(@component) = get_all_components($dfdOut[$i], $project);
   my($flowOK) = 0;
   my($flow);
   foreach $flow (@outFlow) {
      my($component);
      foreach $component (@component) {
         if( $flow eq $component ) {
               $flowOK = 1;
               last;
         }
      }
      if($flowOK)   last;
   }
   if(!$flowOK)
      printf STDERR
         "*WARNING: output flow %s from parent not found in minispec\n", $dfdOut $i];
      $parentMsg = $parentMsg,"%... *WARNING: output flow ",
                   $dfdOut $i]," from parent not found in minispec\n";


#... input flows on the minispec should be on the DFD
   for($f=0 ; $f<$nInFlow; $f++)
      my($isComposite) = 1;  #... boolean indicates that flow is in a composite
      for($i=0 ; $i<$nDfdIn ; $i++)
         if($dfdIn $i] eq $inFlow $f])
            $isComposite = 0;
            last;

      if($isComposite)     #... flow must be from a composite on the DFD
         my($foundComposite) = 0;  #... boolean indicates that composite has been resolved
         for($i=0 ; $i<$nDfdIn ; $i++)
            if(get_composite($dfdIn $i],$inFlow $f],$project))
               $foundComposite = 1;
               last;

         if(!$foundComposite)
            print STDERR "*WARNING: input flow ",$inFlow $f]," not found on DFD\n";
            $parentMsg = $parentMsg,"%... *WARNING: input flow ",
                         $inFlow $f]," not found on DFD\n";


#... output flows on the minispec should be on the DFD
   for($f=0 ; $f<$nOutFlow; $f++)
      my($isComposite) = 1;  #... boolean indicates flow is in a composite on the DFD
      for($i=0 ; $i<$nDfdOut ; $i++)
         if($dfdOut $i] eq $outFlow $f])
            $isComposite = 0;
            last;

      if($isComposite)     #... flow must be from a composite on the DFD
         my($foundComposite) = 0;  #... boolean indicates that composite has been resolved
         for($i=0 ; $i<$nDfdOut ; $i++)
            if(get_composite($dfdOut $i],$outFlow $f],$project))
               $foundComposite = 1;
               last;

         if(!$foundComposite)
            print STDERR "*WARNING: output flow ",$outFlow $f]," not found on DFD\n";
            $parentMsg = $parentMsg,"%... *WARNING: output flow ",
                         $outFlow $f]," not found on DFD\n";
```

Figure C.1: Example PERL Module (4 of 9).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
      else
       print STDERR
          "*WARNING: Data Flow Diagram $fn not found for consistency check\n";
       $parentMsg = $parentMsg,
          "%... *WARNING: Data Flow Diagram $fn not found for consistency check\n";

   else
    die "usage: $0 processNumber < miniSpec.ms";

#... read in the references for the bibliography
if((keys(%cite)>0) && (!$bibTeX))
   print STDERR "\\cite used but no BibTeX - looking for References.tex\n";
   $fnBib = inputFile("bib","References.tex");
   die "could not find References.tex" if ! -e $fnBib;
   open(BIB,"<$fnBib") || die "could not open \<$fnBib\> for input";
   $version $fnBib  = "not specified";
   my($inItem) = 0;
   my($started) = 0;
   while(<BIB>)
      #... try to get the version id correct
      my($save) = $_;
      tr/a-z/A-Z/;
      if(     /^\%.*VERSION\s+(\S+)\s/ || /^\%.*VERSION\s+(\S+)$/
         || /^\%.*VER\s+(\S+)\s/     || /^\%.*VER\s+(\S+)$/ )
         $_ = $1;
         if(/^(\d+)( A-Z| )$/)
            my($newVer) = $1;
            my($newSubVer) = $2;
            $_ = $version $fnBib ;
            if(/^(\d+)( A-Z| )$/)
               my($oldVer) = $1;
               my($oldSubVer) = $2;
               $version $fnBib  = $newVer.$newSubVer
                   if (   ($newVer>$oldVer)
                       || (   ($newVer==$oldVer)
                           && ($newSubVer gt $oldSubVer)));
            else
             $version $fnBib  = $newVer.$newSubVer;

         else
          $version $fnBib  = $_ if $version $fnBib  eq "not specified";

     else
      $_ = $save;
      #... end of version id
      if(/^\s*\\begin\s*\ \s*thebibliography\s*\ /)
         $started = 1;
         $bibligraphy @bibligraphy| = $_;
        else
      $inItem = $1 if /\\bibitem\s*\ [^\]]*\]\s*\ ( ^\ |+)\ \s*\;/;
      die "did not find \\begin\ thebibliography\ " if (($inItem) && (!$started));
      if(/^\s*\\end\s*\ \s*thebibliography\s*\ /)
         $bibligraphy @bibligraphy| = $_;
         $inItem = 0;
               last;
              else
               if(($inItem) && (exists($cite $inItem )))
                  $cite $inItem  = 1;  #... flag that bibliographic reference has been found
                  $bibligraphy @bibligraphy| = $_ ;
                 else

   close(BIB);
   #... check for unresolved citations
   foreach(keys(%cite))
      die "could not find bibliography reference for $_" if !$cite $_ ;

#... write LaTeX output file
$_ = $processNumber;
tr/\./\_/;
$fn = outputFile("ms","ms\_$_.tex");
open(OUT,">$fn") || die "could not open $fn for output";
print OUT "%... output generated by $0 on $runtime\n";
print OUT "%...                    component   version\n";
printf OUT "%...        %20s  %s\n",$0,$version $0 ;
delete $version $0 ;
printf OUT "%...        %20s  %s\n","miniSpec",$version "miniSpec" ;
delete $version "miniSpec" ;
foreach $c (sort(keys %version))
   printf OUT "%...        %20s  %s\n",$c,$version $c ;

print OUT "\\documentclass",$formatOptions,"{Minispec}\n";
foreach(keys(%usePackage))
   my($key) = $_;
   my($options) = $usePackage $key ;
   print OUT "\\usepackage",$options,"{$key}\n";

print OUT "\\pagestyle{empty}\n\n";
print OUT "\\setlength\{\\parindent\}\{0em\} ",
          "            \the indent on paragraphs\n";
print OUT "\\setlength\{\\parskip\}\{\\baselineskip\} ",
          " \paragraph skip is one blank line\n";
print OUT "\\productID{$runtime}\n";
print OUT "\\project{$project}\n";
print OUT "\\submodel{$submodel}\n";
print OUT "\\title{$processNumber. $processName}\n";
print OUT "\\author{$author}\n";
print OUT "\\version{$versionId}\n";
print OUT "\\date{$date}\n";
print OUT "\\implementer{$implementer}\n";
print OUT "\\implementDate{$implementDate}\n";
print OUT "\\reviewer{$reviewer}\n"
   if $reviewer ne "reviewer not defined";
print OUT "\\reviewDate{$reviewDate}\n\n"
   if $reviewDate ne "reviewDate not defined";
print OUT "\\begin{document}\n";
print OUT "\\maketitle \\thispagestyle{empty}\n\n";

for($i=0; $i<$nFreeFormBeforeIO; $i++)
   print OUT $freeForm 0| ;
   shift(@freeForm);
```

Figure C.1: Example PERL Module (5 of 9).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
print OUT "\n\n";


print OUT "\\vspace{0.2in}\n";
print OUT "\\begin{tabular}{|l|c|c|c|} \\hline\n";
print OUT "\\multicolumn{1}{|c|}{\\emph{Variable}} & \\emph{Symbol} ".
          " & \\emph{Units} & \\emph{I/O} \\\\ \\hline\n";
for($i=0;$i<$nOutFlow;$i++)
   ($symbol, $units) = getSymbolUnits($outFlow $i|,$project);
   $symbol =~ s/\\\\/\\\_/g;
   $_ = $outFlow $i|; s/\_/\\\_/g; s/\\\\/\_/g;
   printf OUT "%s \& \$%s\$ \& %s \& O \\\\\n", $_, $symbol, $units;

for($i=0;$i<$nInFlow;$i++)
   ($symbol, $units) = getSymbolUnits($inFlow $i|,$project);
   $symbol =~ s/\\\\/\\\_/g;
   $_ = $inFlow $i|; s/\_/\\\_/g; s/\\\\/\_/g;
   printf OUT "%s \& \$%s\$ \& %s \& I \\\\\n", $_, $symbol, $units;

print OUT "   \\hline\n\\end{tabular}\n";
print OUT "\\vspace{0.2in}\n";


#... results of consistency checking
print OUT $parentMsg;
for($i=0; $i<$nCompositeUsed; $i++)
   $_ = get_Label("long", $compositeUsed $i|, $project);
   s/\_/\\\_/g;
   s/\ /\\\ /g;
   s/\ /\\\ /g;
   s/\ /\\\|/g;
   s/\ /\\\|/g;
   printf OUT "\n\$\\textrm{%s} \= \\lbrace", $_;
   my($lineLength) = length($_)+4;
   my($sep) = "";
   $_ = $composite $compositeUsed $i| ;
   my(@subComponent) = split /,/;
   my($subComponent);
   foreach $subComponent (@subComponent)
      print OUT $sep;
      if($lineLength>65)      #... line break if many components
         print OUT "\\\\ ";
         $lineLength = 0;
        else
      $sep = ",";
      if(exists($compositeUsed $compositeUsed $i|.",".$subComponent ))
         $_ = $subComponent;
         s/\_/\\\_/g;
         s/\ /\\\ /g;
         s/\ /\\\ /g;
         s/\ /\\\|/g;
         s/\ /\\\|/g;
         printf OUT "\\textbf{%s}", $_;
        else
         $_ = $subComponent;
         s/\ /\\\ /g;
         s/\ /\\\ /g;
         s/\ /\\\ /g;
         s/\ /\\\|/g;
         s/\ /\\\|/g;
                              printf OUT "\\textrm{%s}", $_;

                           $lineLength += length($_)+1;

                        print OUT "\\rbrace\$\n";

                     print OUT "\\vspace{0.2in}\n" if $nCompositeUsed>0;

                     while(@freeForm)
                        print OUT $freeForm 0|;
                        shift(@freeForm);



                     while(@bibligraphy)
                        print OUT $bibligraphy 0|;
                        shift(@bibligraphy);


                     print OUT "\n\\end{document}\n";
                     close(OUT);

                     ##############################################################################
                     #... subroutine to identify usage of a (potentially) composite flow
                     sub get_composite
                        my $parent  = shift;        #... first argument is the composite to be checked
                        my $child   = shift;        #... first argument is the flow name
                        my $project = shift;        #... second argument is the project

                        my @component = get_components($parent,$project);
                        my($component);
                        my($found) = 0;  #... boolean indicates that composite flow has been found
                        foreach $component (@component)
                           if($component eq $child)
                              my($i);
                              my($alreadyUsed) = 0;
                              for($i=0;$i<$nCompositeUsed;$i++)
                                 if($compositeUsed $i| eq $parent)
                                    $alreadyUsed = 1;
                                    last;


                              if(!$alreadyUsed)   $compositeUsed $nCompositeUsed++| = $parent;
                              $compositeUsed $parent.",".$child  = 1;   #... component is used
                              $found = 1;


                        if(!$found)   #... components could themselves be composite flows
                           foreach $component (@component)
                              if($component ne $parent)
                                 $found = get_composite($component,$child,$project);
                                 if($found)
                                    my($i);
                                    my($alreadyUsed) = 0;
                                    for($i=0;$i<$nCompositeUsed;$i++)
                                       if($compositeUsed $i| eq $parent)
                                          $alreadyUsed = 1;
                                          last;
```

Figure C.1: Example PERL Module (8 of 9).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
                if(!$alreadyUsed) { $compositeUsed[$nCompositeUsed++] = $parent; }
                $compositeUsed{$parent.",".$component} = 1;  #... composite component is used
                last;
            }
        }
    }
    }
    return($found);
}

################################################################################
#... subroutine to identify components of a (potentially) composite flow
sub get_components {
    my $name       = shift;        #... first argument is the flow name
    my $project    = shift;        #... second argument is the project

    my($fn);
    my($nComponents) = 0;
    my(@subComponent);
    my($subComponent);
    my(@retVal);

    #... determine the file name (this will be replaced by a file-hash and RCS/CVS)
    $fn = inputFile("rdd",$name.".rdd");
    $version{$fn} = "noversion";

    open(RDD,"<$fn") || die "could not open $fn for input";
    while(<RDD>) {
        if(/^CompositeOf\s*\:\s*(\S*.*)$/) {
            $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
            $composite{$name} = $1;
            @subComponent = split /,/, $1;
            foreach $subComponent (@subComponent) {
                $retVal[$nComponents++] = $subComponent;
            }
        }
        if(/^Version\s*\:\s*(\S*.*)$/) {
            $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
            $version{$fn} = $1;
        }
        if(/^Project\s*\:\s*(\S*.*)$/) {
            $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
            die "project for $name is $1, but diagram project is $project" if $project ne $1;
        }
    }
    close(RDD);

    $retVal[$nComponents++] = $name;
    return(@retVal);
}

################################################################################
#... subroutine to identify components of a (potentially) composite flow
sub get_all_components {
    my $name       = shift;        #... first argument is the flow name
    my $project    = shift;        #... second argument is the project

    my($fn);
    my($nComponents) = 0;
    my(@subComponent);
    my($subComponent);
    my(@retVal);

    #... determine the file name (this will be replaced by a file-hash and RCS/CVS)
    $fn = inputFile("rdd",$name.".rdd");
    $version{$fn} = "noversion";

    open(RDD,"<$fn") || die "could not open $fn for input";
    while(<RDD>) {
        if(/^CompositeOf\s*\:\s*(\S*.*)$/) {
            $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
            $composite{$name} = $1;
            @subComponent = split /,/, $1;
            foreach $subComponent (@subComponent) {
                $retVal[$nComponents++] = $subComponent;
            }
        }
        if(/^Version\s*\:\s*(\S*.*)$/) {
            $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
            $version{$fn} = $1;
        }
        if(/^Project\s*\:\s*(\S*.*)$/) {
            $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
            die "project for $name is $1, but diagram project is $project" if $project ne $1;
        }
    }
    close(RDD);
    my($i);
    my(@hold);
    my($nHold) = 0;
    for($i=0; $i<$nComponents; $i++) {
        my(@subSubComponent) = get_all_components($retVal[$i],$project);
        my($subSubComponent);
        foreach $subSubComponent (@subSubComponent) {
            if($subSubComponent ne $retVal[$i]) {
                $hold[$nHold++] = $subSubComponent;
            }
        }
    }
    for($i=0; $i<$nHold; $i++) {
        my($j);
        my($found) = 0;
        for($j=0; $j<$nComponents; $j++) {
            if($retVal[$j] eq $hold[$i]) {
                $found = 1;
                last;
            }
        }
        if(!$found) { $retVal[$nComponents++] = $hold[$i]; }
    }
    $retVal[$nComponents++] = $name;
    return(@retVal);
}

################################################################################
#... subroutine to parse the Requirements Data Dictionary file
sub getSymbolUnits {
    my($label)      = shift;        #... first argument is the data name
    my($project)    = shift;        #... second argument is the project
```

Figure C.1: Example PERL Module (7 of 9).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```perl
   my(@retVal); $retVal[0]=""; $retVal[1]="";

   #... determine the file name (this will be replaced by a file hash and RCS/Cvs)
   my($fn) = inputFile("rdd",$label,".rdd");
   $version{$fn} = "noversion";

   if(open(RDD,"<$fn"))
      while(<RDD>)
         if(/^MathName\s*\:\s*(\S*.*)$/)
            $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
            $retVal[0] = $1;

         if(/^PhysicalUnits\s*\:\s*(\S*.*)$/)
            $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
            $retVal[1] = $1;

         if(/^Version\s*\:\s*(\S*.*)$/)
            $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
            $version{$fn} = $1;

         if(/^Project\s*\:\s*(\S*.*)$/)
            $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
            die "project for $label is $1, but minispec project is $project" if $project ne $1;


      close(RDD);


   return(@retVal);

###########################################################################
#... subroutine to parse the Requirements Data Dictionary file
sub get_Label
   my $labelType = shift;       #... first argument is type of label to look for
   my $name      = shift;       #... second argument is the flow name
   my $project   = shift;       #... third argument is the project

   my($fn);
   my($label);
   my($retVal) = $name;

   if($labelType eq "math")
      $label = "MathName";
    elsif($labelType eq "long")
      $label = "LongName";
    else
      $label = "Name";


   #... determine the file name (this will be replaced by a file hash and RCS/Cvs)
   $fn = inputFile("rdd",$name,".rdd");
   $version{$fn} = "noversion";

   open(RDD,"<$fn") || die "could not open $fn for input";
   while(<RDD>)
      if(/^$label\s*\:\s*(\S*.*)$/)
         $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
         $retVal = $1;
         if(/^Version\s*\:\s*(\S*.*)$/)
            $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
            $version{$fn} = $1;

         if(/^Project\s*\:\s*(\S*.*)$/)
            $1 =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
            die "project for $name is $1, but diagram project is $project" if $project ne $1;


      close(RDD);

      return($retVal);


###########################################################################
#... subroutine to preprocess freeform LaTeX
sub notQuiteFreeForm
   my($inputLine) = shift;       #... first argument is the line to be reformatted

   $_ = $inputLine ? $inputLine : "";
   $_ =~ /(.*\S)\s*$/;  #... eliminate trailing spaces
   if($_ && /\\bibliography\s*\{([^\}]+)\}/)     #... BibTex processing
      $bibTeX = 1;
      my($line) = $`."\\bibliography\{";
      my($after) = $';
      my(@bibDB) = split(/\,/,$1);
      my($sep) = "";
      while(@bibDB)
         my($bib) = $bibDB[0];
         $bib =~ s/\s//g;
         my($fnBib)=inputFile("bib",$bib.".bib");
         die "could not find bibTex database \"$bib\"" if ! -e $fnBib;
         $fnBib =~ s/\\/\//g;   #... fix peculiar MS-DOS backslashes
         $fnBib = $` if /\.bb[ll]|bb[\s*$/;
         $line = $line.$sep.$fnBib;
         $sep = "\,";
         shift(@bibDB);

      $_ = $line."\}".$after;
     else
   if($_)
      while(/\\math\s*\{\s*([^\}]+)\s*\}/)
         my($before) = $`;
         my($after) = $';
         my($shortName) = $1;
         my($mathName) = get_Label("math", $shortName, $project);
         $_ = $before.$mathName.$after;


   if($_ && /\\include\s*\{([^\}]+)\}/)     #... traditional LaTeX \include
      my($inc) = $1;
      my($line) = $`."\\include\{";
      my($after) = $';
      $inc =~ s/\s//g;
      $fnInc = inputFile("tex",$inc);
      $fnInc = inputFile("tex",$inc."\.tex") if ! -e $fnInc;
      $fnInc = inputFile("ms",$inc) if ! -e $fnInc;
      $fnInc = inputFile("ms",$inc."\.tex") if ! -e $fnInc;
      die "could not find \\include \"$inc\"" if ! -e $fnInc;
```

Figure C.1: Example PERL Module (8 of 9).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```perl
        $fnInc = $' if /\.[tT][eE][xX]$/;
        $_ = $line.$fnInc."\}".$after;
    } else {}
    if($_ && (/\\cite\s*\{([^\{^\}]+)\}/ || /\\cite\s*\[[^\]]*\]\s*\{([^\{^\}]+)\}/)) {
        my($save) = $_;
        my(@citation) = split(/\,/,$1);
        while(@citation){
            $_ = $citation[0];
            s/\s//g;
            $cite{$_} = 0;
            shift(@citation);
        }
        $_ = $save;
    } else {}
    $usePackage{"apalike"} = "" if ($_ && /\\bibliographystyle\s*\{\s*aecl\s*\}/);
    $usePackage{"graphicx"} = "\[pdftex\]" if ($_ && /\\includegraphics/);
    $usePackage{"hyperref"} = "\[pdftex\]" if ($_ && /\\href/);
    $usePackage{"picinpar"} = "" if ($_ && /\\begin\{figwindow\}/);

    my($retVal) = $_ ? $_ : "";

    return($retVal);
}

##############################################################################
#... read .rc file
sub readRC{

    $_ = $pwd;
    my($delim) = /(\W)\w*$/ ? $1 : "\/";

    my($rcFile) = ".TeXspecrc";
    my($fn) = $rcFile;
    $fn = $ENV{"TEXSPECDIR"}.$rcFile if (( !-e $fn) && exists($ENV{"TEXSPECDIR"}));
    $fn = $ENV{"HOME"}.$rcFile if (( !-e $fn) && exists($ENV{"HOME"}));

    if(-e $fn) {
        open(RC,"<$fn") || die "could not open $fn for input";
        my($in) = "";
        while(<RC>){
            if(/^\.(\w+)\:\s*(\S.*)$/) {        #...  location of file type
                $in = $1;
                $_ = $2;
                $_ = $1 if /(.*\S)\s*$/;
                my($dir) = /\w$/ ? $_.$delim : $_;
                ##die "\<$in\> contains invalid directory \<$dir\>" if (! -d $dir);
                $path{$in} = 0 if (exists($path{$in}));
                $dir =~ s/\\/\//g;    #... TeX chokes on DOSisms
                $path{"$in\|".$path{$in}} = $dir;
                $path{$in} = $path{$in}+1;
            } elsif(/^\s*(\S.*)$/) {        #...  multiple directories (possibly)
                $_ = $1;
                $_ = $1 if /(.*\S)\s*$/;
                my($dir) = /\w$/ ? $_.$delim : $_;
                $dir =~ s/\\/\//g;    #... TeX chokes on DOSisms
                if(-d $dir) {
                    my($key) = "$in\|".($path{$in});
                    $path{$key} = $dir;
                    $path{$in} = $path{$in}+1;
                } else {}
            } else {}
        }
        close(RC);
    }
    return;
}

##############################################################################
#..  subroutine to find an input file
sub inputFile{
    my($typ) = shift;
    my($fn) = shift;

    return $fn if -r $fn;

    if(exists($path{$typ})) {
        my($i);
        for($i=0;$i<$path{$typ};$i++) {
            $_ = $path{"$typ\|$i"}.$fn;
            return $_ if (-r $_);                #... Path.File
            $_ = $fn;
            tr/a-z/A-Z/;
            $_ = $path{"$typ\|$i"}.$_;           #... Path.FILE
            return $_ if (-r $_);
            tr/a-z/A-Z/;
            return $_ if (-r $_);                #... PATH.FILE
            $_ = $fn;
            tr/A-Z/a-z/;
            $_ = $path{"$typ\|$i"}.$_;
            return $_ if (-r $_);                #... Path.file
            tr/A-Z/a-z/;
            return $_ if (-r $_);                #... path.file
        }
    } else {}

    return $fn;
}

##############################################################################
#... subroutine to name an output file
sub outputFile{
    my($typ) = shift;
    my($fn) = shift;

    if(exists($path{"$typ\|0"}))
        return $path{"$typ\|0"}.$fn;
    else
        return $fn;
    }

    return "";
}
##############################################################################
```

Figure C.1: Example PERL Module (9 of 9).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

D Sample Java Module (GUI)

Figure D.1 lists one of the smaller TEXspec Java modules '*DesignSpecificationEditFrame.java*'. This listing is intended to illustrate the coding design and style used in all such modules.

The TEXspec GUI contains are over 32,000 lines of Java over 85 modules.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
//Title:        TeXspec : DesignSpecificationEdit
//Version:      0.
//Copyright:    Copyright (c) 2
//Author:       Steve Oli'
//Company:      University of Mani'
//Description:  Graphical interface for TeXspec CASE

package TeXspecGU;

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

/**
 * Edit/Create a Design Specificat.
 *
 * @author <A HREF="mailto:olivers@aecl.ca">Stephen Oliv
 * @version  0.10,  Mar  3, 2
 */
 public class DesignSpecificationEditFr extends JInternalFram

   //... instance variabl

   /**
   * underlying pane
   */
   private JPanel editPane = new JPanel( );
   /**
   * layout for the underlying par
   */
   private GridBagLayou editPanelLayo = new GridBagLayou( );

   /**
   * panel for common TeXspecComponent cont
   */
   private TeXspecComponentEditPar componentPane;

   /**
   * panel for design spec specific conte
   */
   private DesignSpecificationEditPa designPane;

   /**
   * panel for "SAVE" and "CANCEL" butt
   */
   private SaveCancelPane buttonPane = new SaveCancelPane( );

   /**
   * Design Specification to be edited (origi
   */
   private DesignSpecificati dsOld;

   /**
   * Design Data Dictionary Entry to be edited (modi
   */
   private DesignSpecificati dsNew;

   /**
   * data model for table of design sp
   */
   private TeXspecComponentTableMoc dataMode. ;

   /**
    * Used for generating layout constrai
    *
    */
   private int curRow = 0;
```

Figure D.1: Example Java Module (1 of 5).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
/**
 * Creates an instanc
 *
 * @param title         the String to display in the titl
 * @param resizable     if true, the frame can be r
 * @param closable      if true, the frame can be c
 * @param maximizable   if true, the frame can be max
 * @param iconifiable   if true, the frame can be ico
 * @param d    design Specification to be e
 * @param m    dataModel for Jtable of specifications which may be r
 */
public DesignSpecificationEditFr(String title, boolean resizabl,
boolean closabl, boolean maximizabl, boolean iconifiabl,
DesignSpecificati d, TeXspecComponentTableMoc m)
   super(title, resizabl, closabl, maximizabl, iconifiabl);

   dsOlc = d;
   dsNew = (DesignSpecificati  d.copy  ;
   dataMode = m;

   try
      jbInit();
      pack();

   catch(Exception ex)
      ex.printStackTrac ;



/**
 * Creates an instanc
 *
 * @param p    design Specification to be e
 * @param m    dataModel for Jtable of specifications which may be r
 */
public DesignSpecificationEditFr(DesignSpecificati p,
TeXspecComponentTableMoc m)
   this("Design Specificatio, true, true, true, true, p, m);


/**
 * Creates an instance. Used by JBuilder
 *
 */
public DesignSpecificationEditFr(
   this("", true, true, true, true, null, null);


/**
 * Initiali:
 *
 */
private void jbInit() throws Exceptio

   //... set up common TeXspec component elements for e
   componentPane = new TeXspecComponentEditPar(dsNew);

   //... set up design data dictionary specific elements for e
   designPane = new DesignSpecificationEditPar(dsNew);

   //... set up the listeners for the "SAVE" & "CANCEL" b
   buttonPane.getSaveButto .addActionListene
     new java.awt.event.ActionListene
        public void actionPerforme(ActionEven e)
        saveButton actionPerform(e);

    ;
   buttonPane.getCancelButto .addActionListen
```

Figure D.1: Example Java Module (2 of 5).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
      new javz.awt.event.ActionListene
         public voic actionPerforme ActionEven e:
         cancelButton actionPerforme ;

   ;

   //... set up the main pa:
   getContentPar .setLavou new GridBagLavou  ;
   getContentPar .adc componentPane,
             new GridBagConstrain
                                        0, 0,  // grid pos x.
                                        GridBagConstrain.REMAINDE, 1,
                                        // grid width, heig
                                        1.C, 0.C,
                                        // weight x.
                                        GridBagConstrain.EAST,
                                        GridBagConstrain.HORIZONTA,
                                        // anchor, fil
                                        new Inset: 0, 0, 0, 0 ,
                                        // inset
                                        5, 2
                                        // padx, pad
                                                  ;

   getContentPar .adc designPane,
             new GridBagConstrain
                                        0, 1,  // grid pos x,
                                        GridBagConstrain.REMAINDE, 1, //
                                        grid width, heigl
                                        1.C, 1.C,
                                        // weight x,
                                        GridBagConstrain.EAST,
                                        GridBagConstrain.BOTE,      //
                                        anchor, fil
                                        new Inset: 0, 0, 0, 0 ,
                                        // inset
                                        5, 2
                                        // padx, pad
                                                  ;

   getContentPar .adc buttonPane,
             new GridBagConstrain
                                        0, 2,  // grid pos x,
                                        GridBagConstrain.REMAINDE, 1, //
                                        grid width, heigl
                                        1.C, 0.C,
                                        // weight x.
                                        GridBagConstrain.EAST,
                                        GridBagConstrain.HORIZONTA,
                                        // anchor, fil
                                        new Inset: 0, 0, 0, 0 ,
                                        // inset
                                        5, 2
                                        // padx, pad
                                                  ;


/**
 * Generate lavout constraints for most f.
 *
 * @param width    grid width :<0 = end of row, 0 = remainder
 */
private GridBagConstrain lavout int widtt
   return lavout widtt, GridBagConstrain.CENTEI ;


/**
 * Generate lavout constrai
```

Figure D.1: Example Java Module (3 of 5).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
 *
 * @param width    grid width (<0 = end of row, 0 = remainder
 * @param anchor   see GridBagConstra
 */
private GridBagConstrain layout(int width, int anchor
   GridBagConstrain retVal = null;

   if(width <= 0)
      if( width==0
         width = GridBagConstrain.REMAINDE;
      else
         width = -width;

      retVal = new GridBagConstrain
               GridBagConstrain.RELATIV, curRow++, // grid pos x,
               width, 1,                           // grid width,
               height
               1.0, 0.0,                           // weight x,
               anchor, GridBagConstrain.BOTE,      // anchor, fil
               new Insets(0, 0, 0, 0),             // inset
               0, 2                                // padx, pad
               ;
     else
      retVal = new GridBagConstrain
               GridBagConstrain.RELATIV, curRow,  // grid pos x,
               width, 1,                          // grid width, heig
               1.0, 0.0,                          // weight x,
               anchor, GridBagConstrain.NONE,     // anchor, fil
               new Insets(0, 0, 0, 0),            // inset
               5, 2                               // padx, pad
               ;

   return retVal;



/**
 * Generate layout constrai
 *
 * @param width    grid width (<0 = end of row, 0 = remainder
 * @param anchor   see GridBagConstra
 * @param fill     see GridBagConstra
 */
private GridBagConstrain layout(int width, int anchor, int fill
   GridBagConstrain retVal = null;

   if(width <= 0)
      if( width==0
         width = GridBagConstrain.REMAINDE;
      else
         width = -width;

      retVal = new GridBagConstrain
               GridBagConstrain.RELATIV, curRow++, // grid pos x,
               width, 1,                           // grid width,
               height
               1.0, 0.1,                           // weight x,
               anchor, fill,                       // anchor, resiz
               policy
               new Insets(0, 0, 0, 0),             // inset
               0, 2                                // padx, pad
               ;
     else
      retVal = new GridBagConstrain
               GridBagConstrain.RELATIV, curRow,  // grid pos x,
               width, 1,                          // grid width, heig
               0.1, 0.1,                          // weight x,
               anchor, fill,                      // anchor, resiz
```

Figure D.1: Example Java Module (4 of 5).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
                policy
                new Insets(0, 0, 0, 0),                // inset
                5, 2                                   // padx, pad
              );

   return retVal;
 }

 /**
  * Respond to mouse-click on the "SAVE" pushb
  *
  * @param e event from
  */
 private void saveButton actionPerform ActionEven e)
   componentPane.recordDat();
   designPane.recordDat();
   if( dsOld.getName().equals(dsNew.getName()
      dsNew.copyTo dsOld);
      try
         dsOld.write();
         buttonPane.getCancelButto .setText("Exit");

      catch TeXspecExceptic ex
         ExceptionDialc dlg = new ExceptionDialc Configuratio.
         getDefaultFram ,"TeXspec,true,TeXspecExceptic.error,
         "Could not save DesignDI+"  "+ex.descriptio  );
         Dimensio dlgSize = dlg.getPreferredSi: ;
         Dimensio frmSize = getSize ;
         Point loc = getLocatio ;
         dlg.setLocatio  frmSize.widtt - dlgSize.widtt  / 2 + loc.x,
         frmSize.height - dlgSize.height  / 2 + loc.y);
         dlg.show();

      else
      try
         dsNew.write();
         buttonPane.getCancelButto .setText("Exit");
         if( dataMode != null   dataMode.addComponen dsNew);  //...
         update Jtabl

      catch TeXspecExceptic ex
         ExceptionDialc dlg = new ExceptionDialc Configuratio.
         getDefaultFram  ,"TeXspec,true,TeXspecExceptic.error,
         "Could not create new DesignD+"  "+ex.descriptio  ;
         Dimensio dlgSize = dlg.getPreferredSi: ;
         Dimensio frmSize = getSize  ;
         Point loc = getLocatio  ;
         dlg.setLocatio  frmSize.widtt - dlgSize.widtt  / 2 + loc.x,
         frmSize.height - dlgSize.height  / 2 + loc.y);
         dlg.show();



 /**
  * Respond to mouse-click on the "CANCEL" pushb
  *
  * @param e event from
  */
 private void cancelButton actionPerform ActionEven e
    dispose();

}
```

Figure D.1: Example Java Module (5 of 5).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

E Installation

E.1 Prerequisite Software

TEXspec relies on a number of tools which are available without charge and can be downloaded from various Internet sites. These tools can be installed on various computing platforms. Each of these tools must be installed on a system before TEXspec can be installed. TEXspec should operate on any platform where each of these tools has been installed.

E.1.1 Perl

The main TEXspec processing is performed by modules which have been implemented in PERL [28]. Perl Version 5 was used to develop TEXspec, and earlier versions are unlikely to be compatible.

E.1.2 TEX and LATEX

Various distributions of TEX and LATEX exist for many platforms. TEXspec has been tested on the TeTEX and MikTEX distributions, but should be compatible with any other valid distribution.

Some distributions do not contain the xy-pic package which provides drawing capabilities that TEXspec uses to produce Data Flow Diagrams and Structure Charts, or the vmargin package, which TEXspec uses to control margins. If the selected distribution does not include either of these packages, then the missing package(s) must be downloaded and installed within the TEX installation. Installation of extension packages is detailed in documentation of the TEX distribution.

E.1.3 Noweb

Noweb is a combination of executable programs and a LATEX extension package. Detailed installation instructions are provided for various platforms with the Noweb distribution.

Microsoft Windows-NT users should be aware of the incompatibility of Windows-NT with the Noweb dis-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

tribution binaries (executable images) for other Microsoft Windows systems. Instructions are included with the Noweb distribution for building NT binaries.

E.1.4 JAVA Runtime Environment

Users wishing to run the Graphical User Interface must install a Java Runtime Environment that includes the "Swing" libraries. TEXspec has been tested on Sun Microsystem's JRE version 1.2 and 1.3, but TEXspec should be compatible with any Swing enabled environment.

E.2 TEXspec Specific Installation

The TEXspec distibution includes:

- A number of Perl scripts. If TEXspec is to be run from the command line, then some platforms prefer these to be placed in a particular location. If the GUI is to be used, then the scripts can be placed anywhere provided that the GUI search list is updated to look in that location.
- GUI "batch" files. For each Perl script, a file is required to interface between the GUI and Perl. The TEXspec distribution includes samples for Microsoft Windows environments. These files are only required if the GUI is to be used, and can be placed anywhere provided that the GUI search list is updated to look in that location.
- A Java ARchive (.jar) file containing the executable GUI. This can be placed anywhere, provided that the Java Runtime Environment can access it.
- A class (.cls) file for each publishable product. These must be placed in the LATEX installation. Installation of new class files is detailed in documentation of the TEX distribution.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

References

[1] Advanced Software Technologies *Graphical Designer.* http://www.advancedsw.com

[2] J.J. Bunn. *Floppy and Flow User Manual.* http://vscrna.cern.ch/floppy/contents.html, 1997.

[3] Cadre Technologies, Providence RI. *Teamwork.*

[4] Canadian Standards Association. *Quality Assurance of Analytical, Scientific, and Design Computer Programs for Nuclear Power Plants.* Technical Report N286.7-99, 1999. 178 Rexdale Blvd. Etobicoke, Ontario, Canada M9W 1R3.

[5] T. DeMarco. *Structured Analysis and System Specification.* Yourdon Press/Prentice-Hall, 1978.

[6] Digital Equipment Corporation, Maynard Massachusetts. *Guide to DECdesign.* 1992.

[7] R. Eckstein, D. Wood, and M. Loy. *Java Swing.* O'Reilly & Associates Inc., 1998.

[8] B.W. Goodwin, T.H. Andres, D.C. Donahue, W.C. Hajas, S.B. Keeling, C.I. Kitson, D.M. LeNeveu, T.W. Melnyk, S.E. Oliver, J.G. Szekely, A.G. Wikjord, K. Witzke, and L. Wojciechowski. *The Disposal of Canada's Nuclear Fuel Waste: A Study of Postclosure Safety of In-room Emplacement of Used CANDU Fuel in Copper Containers in Permeable Plutonic Rock. Volume 5: Radiological Assessment.* Technical Report AECL-11494-5,COG-95-552-5, Atomic Energy of Canada Ltd, 1996.

[9] B.W. Goodwin, D.B. McConnell, T.H. Andres, W.C. Hajas, D.M. LeNeveu, T.W. Melnyk, G.R. Sherman, M.E. Stephens, J.G. Szekely, P.C. Bera, C.M. Cosgrove, K.D. Dougan, S.B. Keeling, C.I. Kitson, B.C. Kummen, S.E. Oliver, K. Witzke, L. Wojciechowski, and A.G. Wikjord. *The Disposal of Canada's Nuclear Fuel Waste: Postclosure Assessment of a Reference System.* Technical Report AECL-10717,COG-93-7, Atomic Energy of Canada Ltd, 1994.

[10] E.M. Gurari. *TeX* and LaTeX: Drawing and Literate Programming.* McGraw-Hill, 1994.

[11] Interactive Development Environments, San Francisco, CA. *Software Through Pictures.* 1992.

[12] D.E. Knuth. *Literate Programming.* Center for the Study of Language and Information, 1992.

[13] L. Lamport. *LaTeX: A Document Preparation System.* Addison-Wesley, Reading Massachusetts USA, 1986.

[14] J.W. Leis. *LaTeXcad - a Drawing Package for LaTeX2e.* Communications of the TeX User Group Vol. 21 No. 1, 2000. http://www.eeng.dcu.ie/ csg/latex/latexcad.html

[15] D.M. LeNeveu. *Analysis Specifications for the CC3 Vault Model.* Technical Report AECL-10970,COG-94-100, Atomic Energy of Canada Ltd, 1994.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[16] T.W. Melnyk. *INROC Theory Manual.* Technical Report (unassigned), Ontario Power Generation, 2000. in draft.

[17] S. Oliver. *Computer Program Abstract - INROC 01.* Technical Report 06819-03787.1-T10, Ontario Power Generation, 1999.

[18] S. Oliver, K. Dougan, K. Kersch, C. Kitson, G. Sherman, and L. Wojciechowski. *Unit Testing - a Component of Verification of Scientific Modelling Software.* In T.I. Oren and G.B. Birta, editors, *1995 Summer Computer Simulation Conference*, pages 978–983. The Society for Computer Simulation, 1995.

[19] M. Page-Jones. *The Practical Guide to Structured Systems Design.* Yourdon Press, 1980.

[20] R. Pressman. *Software Engineering: A Practitioner's Approach.* McGraw-Hill, 4th edition, 1996.

[21] N. Ramsey. *Literate Programming Simplified.* IEEE Software, September 1994.

[22] J. Robbins. *ArgoUML Object Oriented Design Tool.* 2001.

[23] K. Rose. *Very High Level 2-Dimensional Graphics.* TeX User Group Conference 1997. http://www.ens-lyon.fr/ krisrose/Xy-pic.html

[24] I. Sommerville. *Software Engineering.* Addison-Wesley, 6th edition, 2000.

[25] W.R. Stevens. *UNIX Network Programming, Volume 1: Networking APIs - Sockets and XTI.* Prentice-Hall, 1997.

[26] W.R. Stevens. *UNIX Network Programming, Volume 2: Interprocess Communications.* Prentice-Hall, 1998.

[27] P.D. Stotts. *Tools Review: 'Software Through Pictures' from IDE.* Journal of Visual Languages and Computing, 4 p201-204, 1993.

[28] L. Wall, T. Christiansen, and R. Schwartz. *Programming Perl.* O'Reilly & Associates, 101 Morris Street, Sebastopol, CA 95472, second edition, 1989.

[29] R.J. Wieringa. *Requirements Engineering: Frameworks for Understanding.* Wiley, 1996.

[30] E. Yourdon. *Modern Structured Analysis.* Yourdon Press/Prentice-Hall, 1989.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Glossary

AECL *Atomic Energy of Canada Ltd*

API *Application Program Interface* A set of routines, protocols, and tools for building software applications. An API facilitates program development by providing pre-defined components.

ASCII *American Standard Code for Information Interchange* A code for representing English characters as numbers, with each letter assigned a number from 0 to 127.

CASE *Computer Aided Software Engineering* A category of software that provides a development environment for software programming. CASE systems offer tools to automate, manage and simplify the development process.

CERN *European Laboratory for Particle Physics* European Organization for Nuclear Research, the world's largest particle physics centre.

CNSC *Canadian Nuclear Safety Commission* Regulator of nuclear energy and materials in Canada.

configuration management system A system to identify and manage change, keeping a record for historical reference.

CP/M *Control Program for Microprocessors* Created by Digital Research Corporation, CP/M was one of the first operating systems for personal computers.

CSA *Canadian Standards Association* A not-for-profit, nonstatutory, voluntary membership association engaged in standards development and certification activities.

Symbolic Debugger A program used to find defects (bugs) in other programs. A debugger allows a programmer to stop a program at a specified point and examine and change the values of variables.

DFD *Data Flow Diagram* A high level abstraction of software requirements showing conceptual processes and the flow of data between them.

DGRTP *Deep Geologic Repository Technology Program*

Design Specification The specification for a single compilable module.

FSF *The Free Software Foundation*

GUI *Graphical User Interface* Pronounced goo-ee. A program interface that takes advantage of the computer's graphics capabilities to make the program easier to use. Well-designed graphical user interfaces can free the user from learning complex command languages.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ISO *International Organization for Standardization* Derived from the greek word iso, which means equal. Founded in 1946, ISO is an international organization composed of national standards bodies from over 75 countries.

Java A general purpose, high-level programming language developed by Sun Microsystems. Java is an object-oriented language similar to C++, but simplified to eliminate language features that cause common programming errors. Java source code files are compiled into a format called bytecode, which can then be executed by a Java interpreter. Compiled Java code can run on most computers because Java interpreters and runtime environments, known as Java Virtual Machines (VMs), exist for most operating systems, including UNIX, the Macintosh OS, and Windows. Bytecode can also be converted directly into machine language instructions by a 'just-in-time' compiler.

LaTeX A typesetting system based on the TeXprogramming language developed by Donald E. Knuth. Most people who use TeXutilize a macro package that provides an easier interface. LaTeX, originally written by Leslie Lamport, is one of the most popular. LaTeXprovides higher-level macros, which makes it easier to format documents but sacrifices some of the flexibility of TeX.

Macintosh A popular model of personal computer made by Apple Computer, featuring a graphical user interface to make it relatively easy for novices to use the computer productively.

MathType An interactive tool for Windows and Macintosh from Design Science Inc that assists in the creation of mathematical notation for word processing, and for TeX& LaTeXand MathML documents

Mini-spec *Process Specification* The description of what is happening in a bottom level, primitive bubble in a dataflow diagram.

MS-DOS *MicroSoft Disk Operating System* Originally developed by Microsoft for IBM, MS-DOS was the standard operating system for IBM-compatible personal computers

MS-Win *Microsoft Windows* A family of operating systems for personal computers owned by Microsoft Inc.

N286.7 CSA Standard for the Quality Assurance of Analytical, Scientific, and Design Computer Programs for Nuclear Power Plants.

OO *Object Oriented* A special type of programming that combines data structures with functions to create re-usable objects.

OPG *Ontario Power Generation* A company owned by the Government of Ontario which operates the majority of Canadian nuclear reactors.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

PC *Personal Computer* The first personal computer produced by IBM was called the PC, and increasingly the term PC came to mean IBM or IBM-compatible personal computers, to the exclusion of other types of personal computers, such as Macintoshes.

PERL *Practical Extraction and Report Language* A programming language developed by Larry Wall, especially designed for processing text. Perl is an interpretive language, which makes it easy to build and test simple programs.

Structure Chart An abstraction of software design showing software modules, usually as a tree, and the flow of data between them.

search list A list of directories to be searched sequentially for a file of a given name. The occurance of the file at a higher level in the list effectively superceeds files of the same name in directories lower in the list.

SGML *Standard Generalized Markup Language* A system for organizing and tagging elements of a document. SGML was developed and standardized by the ISO in 1986. SGML itself does not specify any particular formatting; rather, it specifies the rules for tagging elements. These tags can then be interpreted to format elements in different ways.

TCM *Toolkit for Conceptual Modeling* R.J. Wieringas' collection of software tools to present conceptual models of software systems in the form of diagrams, tables, trees, and the like.

TRADE *Toolkit for Requirements And Design Engineering* R.J. Wieringas' Toolkit for Requirements And Design Engineering.

UNIX Pronounced yoo-niks. A popular multi-user, multitasking operating system developed at Bell Labs in the early 1970s.

W3C *World Wide Web Consortium* An international consortium of companies involved with the Internet and the Web.

XML *Extensible Markup Language* A specification developed by the W3C. XML is a pared-down version of SGML, designed especially for Web documents. It allows designers to create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications and between organizations.

X-Windows A windowing and graphics system developed at the Massachusetts Institute of Technology. Almost all UNIX graphical interfaces are based on X-Window.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Index

AECL, 1, 4, 12, 27
API, 50
ArgoUML, 6, 54

CASE, 2–6
CERN, 28
Chunk, 27, 28, 30–32, 34, 47, 57, 64
Class File, 12, 53, 54
CNSC, 1, 5, 53
Code Chunk, 27
Condition, 15, 16, 32, 64
Configuration File, 11, 39, 40, 54
Configuration Management, 11, 12, 37, 54
Consistency, 2–5, 8, 19, 20, 24, 25, 28, 29, 34, 35
Control Flow, 23
CSA, 1, 2, 4, 5, 37, 53

Data Dictionary, 3, 4, 6, 11, 14–18, 21, 25, 27, 28, 30, 32, 34, 40–45, 47, 55, 64
Data Flow Diagram, 2, 3, 5, 8, 15, 20–25, 34, 40, 43, 45, 46, 54, 55
DecDesign, 2
Design Specification, 3, 6, 8, 11–13, 15, 17, 27–32, 34–36, 47, 48, 51, 57–64
DGRTP, 1, 45

European Laboratory for Particle Physics, 28

Floppy, 10, 28, 51
FSF, 39, 54

Graphical Designer, 2
GUI, 9, 14, 38

INROC, 2

Java, 6, 14, 38, 50, 53, 54, 78–83

LaTeX, 6, 9, 10, 12–14, 16–21, 25, 28, 35, 37, 42, 51, 53, 54, 57, 63, 64
Literate Programming, 6, 12, 27, 28

Mini-Spec, 25
Mini-spec, 2, 3, 8, 11, 17, 20, 24–26, 34, 43–45, 55
MS-windows, 6

Noweb, 6, 10, 13, 27, 28, 32, 47, 51–53, 57, 63

OPG, 1

PERL, 13, 14, 27, 28, 38, 50, 51, 53, 54, 69–77
Postcondition, 15, 16, 30, 32, 47, 64
Precondition, 15, 16, 30, 32, 47, 64
Process Specification, 25

Search List, 11, 39
Structure Chart, 2, 3, 8, 11, 15, 27–30, 34–37, 49, 54

TCM, 5
TRADE, 5

UNIX, 5

Version of Components, 12, 57

Windows, 6

X-windows, 5
XML, 54

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

xypic, 6, 22

Yourdon, 2, 8, 13, 20, 23, 45

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.