Technical Document 2984
September 1997

CMS-2 to Ada
Translator Evaluation
Final Report

Ron Iwamiya
Hans Mumm
Bob Ollerton
Bryan Riegel
NRaD
Currie Colket
SPAWAR

19971128 027

Approved for public release; distribution is unlimited.

Naval Command, Control and Ocean Surveillance Center
RDT&E Division, San Diego, CA 92152–5001
Technical Document 2984
September 1997

CMS-2 to Ada
Translator Evaluation
Final Report

Ron Iwamiya
Hans Mumm
Bob Ollerton
Bryan Riegle
NRaD
Currie Colket
SPAWAR

Approved for public release; distribution is unlimited.

Naval Command, Control and Ocean Surveillance Center
RDT&E Division, San Diego, CA 92152-5001
ADMINISTRATIVE INFORMATION

The work detailed in this document was performed for the Office of Naval Research (ONR 311) by the Naval Command, Control and Ocean Surveillance Center (NCCOSC) RDT&E Division (NRaD). Authors from the following entities contributed to this report: the Systems Branch, Code D4122; the Technology Branch, Code D4123; the Tactical and Battle Cube Systems SSA Branch, Code D871, and SPAWAR PMW 133–2.

Released by
M. W. Morgan, Head
Systems Branch

Under authority of
R. B. Volker, Head
Advanced Concepts and
Technology Division

ACKNOWLEDGEMENTS

The authors wish to thank the reviewers of this document. These reviewers include Mr. Jim Palmer, Johns Hopkins University Applied Physics Laboratory; Dr. Noah Prywes, Computer Command and Control Company; Dr. Charlie Sampson, Computer Sciences Corporation; Dr. Michael Sharpio, NRaD; and Mr. John Bergey, Software Engineering Institute.
EXECUTIVE SUMMARY

OBJECTIVE

The objective of this evaluation was to determine the maturity of the CMS-2 to Ada translators and associated tools, to determine the capabilities of these translators, and to provide information to CMS-2 project managers to assist them in the evaluation of costs and risks of translating CMS-2 to Ada. The evaluation was conducted by NRaD with funding from the Office of Naval Research.

RESULTS

This report contains the results of an in-depth evaluation of three CMS-2 to Ada translators. The translators evaluated were developed by the Johns Hopkins University Applied Physics Laboratory, Computer Command and Control Company, and Computer Sciences Corporation. The evaluation was done in three phases: Quick Look, Stress Testing, and Reengineer Until Ada Code Executes Correctly. The report contains a description of the evaluation process, the detailed results of the three phases of the evaluation, lessons learned, recommendations, an annotated bibliography, a description of relevant translation analysis tools, and an explanation of the metrics collected. Metrics collected included person-hours spent in all aspects of the evaluation, McCabe and Halstead metrics, source lines of code count, conformance of Ada source code to Software Productivity Consortium Guidelines, and metrics that measure the difficulty of conversion. Six projects contributed CMS-2 source code. Source code analysis tools were used to examine the quality of the CMS-2 code and corresponding Ada produced by the translators.

RECOMMENDATIONS

Some of the recommendations contained in this report are:

- Recommendations to CMS-2 project managers when considering translation
  - Do not translate unless expertise is available
  - If seriously considering translation, do it soon
  - Analyze CMS-2 code for suitability for translation
- Recommendations to the Navy for advancing translator technology
  - Before investing resources in improving CMS-2 to Ada translators, managers of deployed CMS-2 systems should be polled to find out their plans regarding translation
  - Support development of Ada quality improvement tools
- Recommendations to translator vendors
  - Minimize global interfaces/declarations
  - Avoid use of nonstandard or proprietary math libraries
  - Produce portable Ada code
- Recommendations to reengineering tool vendors
  - Develop Ada quality improvement tools that remove GO TO statements, remove dead code, convert global objects to local objects, and perform automated information hiding
Contents

1. INTRODUCTION ......................................................................................................................... 1-1
   BACKGROUND ......................................................................................................................... 1-1
   PURPOSE OF THE EVALUATION AND KEY ISSUES ............................................................. 1-2
   USERS OF THE RESULTS ....................................................................................................... 1-3
   PURPOSE OF THIS REPORT .................................................................................................... 1-3
   CONTENTS OF REPORT ........................................................................................................ 1-4

2. OVERVIEW OF THE TRANSLATOR EVALUATION PROCESS .............................................. 2-1
   TRANSLATOR EVALUATION .................................................................................................... 2-1

3. SUMMARY OF TRANSLATOR TRANSLATION RESULTS ...................................................... 3-1
   TRANSLATOR Profiles ........................................................................................................... 3-1
   CONCLUSIONS ....................................................................................................................... 3-1

4. LESSONS LEARNED AND OPINIONS .................................................................................. 4-1
   LESSONS LEARNED .............................................................................................................. 4-1
   OPINIONS ............................................................................................................................. 4-3

5. RECOMMENDATIONS ............................................................................................................ 5-1
   RECOMMENDATIONS TO CMS-2 PROJECT MANAGERS WHEN CONSIDERING
   TRANSLATION ....................................................................................................................... 5-1
   RECOMMENDATIONS TO PROJECT MANAGERS AFTER DECIDING TO USE
   TRANSLATOR TECHNOLOGY ................................................................................................. 5-3
   RECOMMENDATIONS TO THE NAVY FOR ADVANCING TRANSLATOR
   TECHNOLOGY ....................................................................................................................... 5-4
   RECOMMENDATIONS TO TRANSLATOR VENDORS ............................................................ 5-7
   ALL VENDORS ...................................................................................................................... 5-7
   APL ........................................................................................................................................ 5-8
   CCC ....................................................................................................................................... 5-9
   RECOMMENDATIONS TO REENGINEERING TOOL VENDORS ........................................ 5-11
   SUGGESTED TRANSLATION STEPS ................................................................................... 5-12
6. REFERENCES

7. ANNOTATED BIBLIOGRAPHY
   TRANSLATING INTO ADA
   OTHER REENGINEERING PAPERS
APPENDIX M: MK-2 CMS-2L AND ADA SOURCE CODE

SOURCE CODE LINES OF CODE (SLOC) ......................................... M-2
NAMING CONVENTIONS .......................................................... M-3
ELIMINATION OF INTERMEDIATE VARIABLES ............................ M-3
USE OF STANDARD PACKAGES .............................................. M-4
MEMORY MANAGEMENT ...................................................... M-4
PERFORMANCE ...................................................................... M-6
POSITION TO REENGINEER .................................................. M-7
ORIGINAL CMS-2L MK-2 FIRE CONTROL SYSTEM ........................ M-8
ADA TRANSLATION USING APL TRANSLATOR ........................... M-13
APL TRANSLATOR COMMON PACKAGES ................................ M-19
ADA TRANSLATION USING CCCC TRANSLATOR ........................ M-23
CCCC TRANSLATOR COMMON PACKAGE ................................. M-37
ADA REENGINEERING OF MK-2 CODE BY HAND .................... M-49
Figures

1. High-level strategy: translate, reengineer, both, or discontinue .............................................. 5-7
2. QA9 CMS-2 and Translated Ada QA9 Line Counts ................................................................. A-16
3. Halstead Metrics ...................................................................................................................... A-17
4. McCabe Cyclomatic Complexity Metric - 1 ............................................................................ A-18
5. McCabe Complexity versus Percent of Ada QA9 ................................................................. A-21
6. DD-Path graph for paths program ......................................................................................... D-31
7. DD-Path graph for paths program with unreachable code .................................................. D-4
8. Procedure Accessing Global Variables without Renaming and without a 
   “Use Clause”............................................................................................................................... D-9
9. Procedure Accessing Global Variable with a “Use Clause” .................................................... D-9
10. Procedure Accessing Global Variables with a Renamed Addition Operator and 
    without a “Use Clause” .............................................................................................................. D-9
11. Procedure Accessing Global Variables with a Renamed Server Package and 
    Addition Operator and without a “Use Clause” ...................................................................... D-10
12. Ada 95 Procedure Accessing Global Variable with a “Use Type Clause” and no 
    Renaming ................................................................................................................................ D-10
13. Ada 95 Procedure Accessing Global Variables with a “Use Type Clause” and with a 
    Renamed Server Package ........................................................................................................ D-10
14. Ada 95 Procedure Using Access-Subprograms with a “Use Type Clause” and with a 
    Renamed Server Package ........................................................................................................ D-12
15. Class Structure for Target Object ........................................................................................... H-2
16. Class Structure for the Operation Object ............................................................................... H-3
17. Information Structure for the Integer-based Test_Case_Subclasses ........................................ H-4
19. Information Structure for the Fixed-based Test_Case Subclasses Fixed-based 
    Test_Case Subclasses ............................................................................................................... H-6
### Tables

2. Software Products vs. Computer .................................................. 2.5
3. Projects Contributing CMS-2 Source Code .................................... 2-6
4. Key Characteristics of CMS-2 vs. Ada 95 .................................... 2-7
5. Translator Profiles .......................................................................... 3-3
6. Summary of Translator Evaluation Results ..................................... 3-5
7. APL QA9 Package Specification Compilation Error List Using the GNAT Compiler - 1 ........................................................................ A-4
8. APL QA9 Package Body Compilation Error List Using the GNAT Compiler - 1 A-7
9. CCCC QA9 Package Body Compilation Error List Using the GNAT Compiler.. A-12
10. TRADA QA9 Package Specification Compilation Error List Using the GNAT Compiler .................................................................. A-13
11. Total SPC Ada Style Violations of Ada Usage (QA9 Produced by Translators) A-23
17. Translating and Compiling Using Project-Contributed Legacy CMS-2 Source Code - 1 ............................................................... B-37
18. QA9 Source Lines of Code by Translator at Various Stages
   (include Predefined) - 1 ................................................................ C-3
19. QA9 Predefined Utilities Source Lines of Code by Translator ................ C-4
20. QA9 Difficulty of Conversion Person Hours .................................. C-5
21. QA9 Difficulty of Conversion SLOC .............................................. C-6
22. QA9 Weighted McCabe Complexity Metric .................................. C-7
23. QA9 Program Size ........................................................................ C-9
24. Hours Performing Preliminary Tasks - 1 ....................................... G-1
25. Hours Performing Quick Look Inspection Tasks - 1 ........................................... G-5
26. Hours Performing Stress Testing Tasks - 1 ......................................................... G-8
27. Hours Performing Reengineering Tasks - 1 ......................................................... G-10
28. Hours Performing General Tasks and Final Report .............................................. G-12
29. Person-hours by work phase for QA9 translations .............................................. G-14
30. QA9 Person-Hours/100 SLOC Translated .............................................................. G-15
31. Ada Quality and Style Criteria - 1 ........................................................................ I-1
32. Description and POCs for Analysis Tools Applied - 1 ........................................ L-1
33. Description and POCs for Potentially Useful Analysis Tools - 1 ......................... L-2
34. MK-2 Source Lines of Code Counts ...................................................................... M-1
1. INTRODUCTION

BACKGROUND

Over the last three decades the Navy has made a large investment in development of software using Compiler Monitor System-2 (CMS-2). Many of these systems will be required to meet the Navy's needs for at least another decade, and will need periodic upgrades. However, they cannot easily be upgraded to support requirements of the warfighter. The hardware platforms are based on 1960s architecture that is very expensive to maintain. CMS-2 software executes on AN/UYK-7, AN/UYK-20, AN/UYK-43, AN/UYK-44, and AN/AYK-14 Navy standard hardware which is increasingly expensive to maintain. The CMS-2 language is no longer taught and few new programmers are willing to learn and use the language. No commercial support exists for the old hardware environments or the CMS-2 computer language and associated software tools.

Upgrading to satisfy new mission requirements also poses another problem. The vast majority of these systems have already reached their performance and memory limitations. Additionally, the high cost of developing applications for archaic, non-supported environments makes such development very expensive and risky. In such situations, the Navy must migrate or augment these systems using modern technology.

In upgrading, a program manager faces the problem of converting the existing system to a modern system. This means eliminating the operational CMS-2 code, UYK computer, and associated support software. One approach could reengineer at the requirements/design level and develop new code in Ada. This approach involves no code translation. A second approach could capture the legacy system as a starting point. By translating the CMS-2 code into Ada, development and execution of the operational system can move to modern computers. The translated Ada code then serves as the base for upgrading the new system. The new software might be a mix of translated Ada and newly developed Ada for portions of the legacy system that are not suited for translation (for example, IO to special devices, direct code, executive service calls). Besides taking advantage of the existing CMS-2 code, this approach has tremendous potential for cost and schedule savings to satisfy the mission requirements.

Advantages of using modern technology are:
- commercial, modern, faster, very powerful hardware architectures;
- modern programming languages (e.g., Ada 95, C++);
- modern interfacing/networking technologies; and
- modern software engineering environments with powerful tools capable of providing high quality systems with high productivity.

The ONR commissioned NRaD to conduct a hands-on evaluation of existing CMS-2 to Ada translators using controlled experiments. These experiments were performed using representative samples of operational CMS-2 code. This report contains the results of the experiments, lessons learned and recommendations.

In discussing capabilities of software "translator" programs, keep in mind that the three products evaluated (APL, CCCC, TRADA) perform operations much closer to what is sometimes called transliteration rather than complete translation. Transliteration is only the first step in the translation process. In natural language translation, such as from French to English, this first step changes the
words and sentences from the original French to the English equivalents. The process continues by changing the resulting text into good, polished English. Source code translators convert CMS-2 statements to equivalent Ada statements -- from CMS-2 constants, variables, procedure calls and GOTO statements to Ada constants, variables, procedure calls, and GOTO statements. Transliteration produces Ada that mirrors the CMS-2 code in both program structure and complexity, as measured by Halstead and McCabe metrics.

Transliteration does not:
• Reduce code complexity.
• Perform significant code restructuring.
• Produce Ada that conforms to guidelines.
• Produce Ada that makes strong use of information hiding.
• Make source code quality improvements, such as removal of variables that are defined but unused or removal of dead code.
• Take advantage of standard Ada packages (e.g., Ada.Calendar)

Those are additional actions that should be part of a complete translation process. The translation process can also include modifications required for execution on new target hardware (for example, a SPARC rather than a UYK-43), conversion of direct code to Ada, modifications to support different input or output devices, and other changes needed for correct compilation and execution of the Ada code.

PURPOSE OF THE EVALUATION AND KEY ISSUES

The purpose for conducting this evaluation are listed below with associated key issues. These key issues were addressed at the beginning of this study and serve as a guide for the evaluation.

1. To determine the overall maturity of the CMS-2 to Ada translators and associated tools.

Key issues are:
• Are translators at or near "production" quality?
• Are translators usable for very large systems?
• Can translators be easily learned by new users?
• Are translation capabilities lacking that could be provided with new tools (for example, removal of GOTOs and unused variables)?
• How useful are the CMS-2 analysis tools, and the assembler to CMS-2 design extractor in the CMS-2 to Ada translation process?

2. To determine the capabilities of existing CMS-2 to Ada translators.

Key issues are:
• What is the quality (for example, Halstead and McCabe metrics and conformance to Ada guidelines) of the Ada code produced?
• What is the CMS-2 construct coverage provided by the translator?
• Are the CMS-2 constructs translated accurately?
• What is the manpower effort required to translate the code?
• What is the manpower effort required to get the code to compile?
• What is the manpower effort required to get the code to execute correctly?
• What are the computer resources required to translate code?

3. To provide information to project managers to assist them in the evaluation of costs and risks of translating CMS-2 to Ada.

Key issues are:
• What are the dollar, resource, and time costs associated with a translation process?
• How much specialized training is required to support the translation process?
• How much of a schedule reduction is possible with a translation process?
• What is the quality of a system produced using a translation process?
• What is the impact of direct code to the overall translation process?
• What are the technical barriers associated with a translation process?
• What are the risks associated with using a translation process?
• Is it practical to consider a translation process?

The program manager needs information on person-hours, resource costs, risks, technical issues, and feasibility to evaluate the practicality of using a translation approach for the project. In making a decision to reengineer at the specification or design level or to reengineer using a translation process, the answers to the above questions help provide insight towards making the necessary engineering tradeoffs. Depending on the amount of redesign required, a program manager might even use a mixed approach where subsystems requiring significant change are redesigned from scratch and subsystems that are relatively stable are translated. Information throughout this report will assist the CMS-2 project manager in answering these questions for the project scenario. The answers to these questions are prerequisite to making sound reengineering decisions.

USERS OF THE RESULTS

Definite or potential users of the evaluation results include the Office of Naval Research (ONR) to address science and technology deficiencies, managers and software engineers of projects considering transition from CMS-2 to Ada, and developers of the translators and associated tools as feedback on the current state of their products.

PURPOSE OF THIS REPORT

This report provides the results of the translator evaluations and related findings. It is intended primarily for the program manager and their technical representatives.
CONTENTS OF REPORT

This report contains the following:
- An overview of the evaluation process*
- An overview of the results*
- Lessons learned*
- Recommendations*
- Results of quick look inspection
- Results of stress testing
- Results of reengineering until Ada code executes correctly
- An interpretation of the metrics collected
- A discussion of potential follow-on work
- References
- Annotated bibliography
- Other metrics

Throughout this report, when we say that a sample “compiled”, we mean that it ran through the compiler with no compiler detecting errors.

Point of contact for information on this report is:
Hans Mumm
NCCOSC RDT&E DIV D4122
San Diego, CA
92152-5000
mumm@nosc.mil
(619)553-4004
(619)553-4808 (fax)

* The first four sections are key to PM decisions. The remainder is supporting evidence and is included for technical completeness
2. OVERVIEW OF THE TRANSLATOR EVALUATION PROCESS

The CMS-2 programming language is comprised of many dialects. Each is almost a full set of the language. The five principal dialects are CMS-2Y, CMS-2L, CMS-2M, CMS-2A, and CMS-2K. Translators were exercised with CMS-2Y, CMS-2M and CMS-2L source code samples selected to exercise all major CMS-2 constructs. The CMS-2A and CMS-2K dialects only differ from the three dialects exercised in the direct code that they allowed. The CMS-2 to Ada translators do not translate the embedded assembler, but rather bypass it or convert it to Ada comments. The Machine Transferable Support Software (MTASS) CMS-2 User Handbook describes the syntax (structure) and semantics (meaning) of the CMS-2 language.

TRANSLATOR EVALUATION

The translator evaluation was done in three phases. The initial phase was Quick Look Inspection. The purpose of this phase was to ensure that all products and resources were ready for subsequent stress testing phases. During this phase a small CMS-2 sample for CMS-2L, less than 5000 source lines of code (SLOC), was CMS-2 compiled and executed. This executing CMS-2 sample was the baseline for comparisons with executions of equivalent code translated to Ada in the third phase. The Quick Look Inspection sample chosen was the MTASS UYK-43 Quality Assurance 9 (QA9) test. QA9 was developed to examine the MTASS CMS-2 compiler's ability to generate arithmetic code that provides acceptable results when executing on an AN/UYK-43 MIL-STD computer. CMS-2 analysis tools were run on the sample to gather Halstead and McCabe metrics, SLOC counts, and other information. The subject translators were used to convert sample CMS-2 code to Ada which were then compiled with the GNU New York University Ada Translator (GNAT), VAX Ada, and Sun Ada compilers. Ada analysis tools were executed on the translated code to gather SLOC, Halstead, McCabe, and other quality metrics.

The second phase was Stress Testing with large CMS-2 Samples. The purpose of this phase was to collect translator behavior data while rigorously exercising all CMS-2 constructs. 84 files from the CMS-2 UYK-7 test suite were selected for input to the three translators. Additional samples were contributed by project offices from Space and Naval Warfare Systems Command (SPAWAR), Naval Sea Systems Command (NAVSEA), and Naval Air Systems Command (NAVAIR). Stress Testing was taken beyond translation to collect Ada SLOC and compile statistics. All Ada generated by each translator was input to three commonly used Ada compilers (GNAT, VAX, Sun) to determine the percentages that compiled correctly.

The third phase, Reengineer Until Ada Code Executes Correctly, covered the reengineering of each translator's QA9 code, compiling, linking, and executing. The intent of this phase was to continue until the results produced by Ada QA9 coincide with those produced by the CMS-2 QA9 baseline sample. An Ada harness/driver was produced by reengineering the translated CMS-2 test harness. During this phase, we also decided to redesign and rewrite the QA9 functionality in Ada 95 directly to compare the product of a total reengineering effort versus translator based results. This phase included the analysis of translated NAVSEA project code with comparisons to the same set of code reengineered by hand. Table 2-1 lists the computers and software products used by each phase of the evaluation process. Table 2-2 shows the products that reside on each computer.
Additional information on the analysis tools used during this evaluation and other potentially useful analysis tools (but not used in these tests) is found in Appendix L.

**CMS-2 TEST CASES**

Unclassified test cases collected included CMS-2 source code from actual SPAWAR, NAVSEA, and NAVAIR projects and the MTASS CMS-2 Compiler Validation Suite. These test cases are shown in Table 2-3. Test cases were used primarily during stress testing. Projects contributing these test cases and function of the contributed code are listed below. For more information, see Table B-3.

**CMS-2 VERSUS ADA**

Characteristics of the CMS-2 and Ada 95 languages are summarized in Table 2-4.
<table>
<thead>
<tr>
<th>Table 2-1. Computers and Software Products Used by Phase of Evaluation</th>
<th>Quick Look inspection</th>
<th>Stress Testing</th>
<th>Reeng. Until Ada Executes Correctly</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPUTERS &amp; OS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAX 11/785/VMS 5.5-1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>SPARC 10/OS 4.1.3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>PC 486/MS-DOS 6.22</td>
<td></td>
<td>X</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>SOFTWARE PRODUCTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMS-2 Test &amp; Analysis Tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTASS (Machine Transferable Support Software) Ver. 11 Rev. 4.0</td>
<td>X</td>
<td></td>
<td></td>
<td>Stress test translators</td>
</tr>
<tr>
<td>METRC (CMS-2 Source Code Metrics Generator) Rev. 6.2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Produce SLOC, Halstead &amp; McCabe metrics</td>
</tr>
<tr>
<td>DESAN (CMS-2 Source Code Design Analyzer) Rev. 6.1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Examine suitability for translation</td>
</tr>
<tr>
<td>Products Evaluated</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>APL Translator Rev. 2.8</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Translate CMS-2 to Ada</td>
</tr>
<tr>
<td>CCCC TransFormer Ver 6.1 Rev. 071196</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Translate CMS-2 to Ada</td>
</tr>
<tr>
<td>TRADA Translator PBL 1.0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Translate CMS-2 to Ada</td>
</tr>
<tr>
<td>Synetics Assembler Design Extractor (Assembler to CMS-2 Translator)</td>
<td>X</td>
<td></td>
<td></td>
<td>Translate direct code to CMS-2</td>
</tr>
<tr>
<td>Ada Compilers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNAT 3.01 Ada Compiler (Ada 95)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Sun Ada Compiler 1.1 (Ada 83)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>VAX Ada Version 2.2-38 (Ada 83)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Ada Analysis Tools</td>
<td>Quick Look Inspection</td>
<td>Stress Testing</td>
<td>Reeng. Until Ada Executes Correctly</td>
<td>Function</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------</td>
<td>----------------</td>
<td>-----------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>ADA SLOC Counter</td>
<td>X</td>
<td></td>
<td>X</td>
<td>Count SLOC</td>
</tr>
<tr>
<td>Logiscope</td>
<td>X</td>
<td></td>
<td>X</td>
<td>Produce Ada quality metrics</td>
</tr>
<tr>
<td>Ada-ASSURED</td>
<td>X</td>
<td></td>
<td>X</td>
<td>Examine conformance to guidelines</td>
</tr>
<tr>
<td>Software Products</td>
<td>VAX 11/785 VAX VMS</td>
<td>SPARC 10 Sun OS 4.1.3</td>
<td>PC 486 MS-DOS 6.22</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>----------------------</td>
<td>--------------------</td>
<td></td>
</tr>
<tr>
<td><strong>CMS-2 Test &amp; Analysis Tools</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTASS (Machine Transferable Support Software) Ver. 11 Rev. 4.0</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>METRC (CMS-2 Source Code Metrics Generator) Rev. 6.2</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DESAN (CMS-2 Source Code Design Analyzer) Rev. 6.1</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>Products Evaluated</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>APL Translator Rev. 2.8</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CCCC TransFormer Ver. 6.1 Rev. 071196</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRADA Translator PBL 1.0</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synetics Assembly Design Extractor (Assembler to CMS-2 translator) Prototype</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>Ada Compilers</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNAT 3.01 Compiler (Ada 95)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sun Ada Compiler 1.1 (Ada 83)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAX Ada Version 2.2-38 (Ada 83)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Ada Analysis Tools</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADA SLOC Counter</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logiscope</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ada-ASSURED</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project</td>
<td>CMS-2 Dialect</td>
<td>Function</td>
<td>Sponsor</td>
<td>POC</td>
</tr>
<tr>
<td>--------------------------------------------------------------</td>
<td>--------------------------------</td>
<td>-------------------------------------------</td>
<td>-----------</td>
<td>--------------------</td>
</tr>
<tr>
<td>S3 Aircraft Tactical Mission Program</td>
<td>CMS-2Y (with ULTRA-32)</td>
<td>Displays radio frequency (RF) data</td>
<td>NAVAIR</td>
<td>Steve McComas (215) 441-1771</td>
</tr>
<tr>
<td>H60B Helicopter (AOP ECP-267 FLIR/Datalink Upgrade)</td>
<td>CMS-2 (Converted from CMS-2M to CMS-2)</td>
<td>Processes acoustic data</td>
<td>NAVAIR</td>
<td>Charley Booth (607) 751-3408</td>
</tr>
<tr>
<td>AEGIS SPYLOOP</td>
<td>CMS-2L</td>
<td>Captures timing data</td>
<td>NAVSEA</td>
<td>Marv Bomberg (612) 546-7402</td>
</tr>
<tr>
<td>MTASS CMS-2 Compiler Validation Suite</td>
<td>CMS-2Y and CMS-2L</td>
<td>Automated CMS-2 compiler tests</td>
<td>NAVSEA</td>
<td>Bryan Riegel (619) 553-9446</td>
</tr>
<tr>
<td>Combat Control System MK-2 Fire Control System</td>
<td>CMS-2L</td>
<td>Computes target location information</td>
<td>NAVSEA</td>
<td>Dan Juttelstad (401) 624-9615</td>
</tr>
<tr>
<td>Extremely Low Communications (ELF) Transmit Processor Computer</td>
<td>CMS-2M</td>
<td>Modulator IO subprogram</td>
<td>SPAWAR</td>
<td>Bart Brock (803) 974-4595</td>
</tr>
</tbody>
</table>
Table 2-4. Key Characteristics of CMS-2 vs. Ada 95

<table>
<thead>
<tr>
<th>CMS-2</th>
<th>Ada 95</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Address based</td>
<td>• Object-oriented</td>
</tr>
<tr>
<td>• Global variables (COMPOOLS)</td>
<td>• Strong real-time support</td>
</tr>
<tr>
<td>• Overlay memory management</td>
<td>• Support for distribution</td>
</tr>
<tr>
<td>• Source code INCLUDE capability</td>
<td>• Interfaces to other languages (e.g., C, FORTRAN, COBOL)</td>
</tr>
<tr>
<td>• Select source code switching on compilation basis (CSWITCH)</td>
<td>• Strong typing</td>
</tr>
<tr>
<td>• minimal support for reentrancy</td>
<td>• Exception handling</td>
</tr>
<tr>
<td>• Supports limited user defined types with type compatibility rules</td>
<td>• Information hiding capabilities</td>
</tr>
<tr>
<td>• No exception handling, and no data abstraction</td>
<td>• Data abstraction</td>
</tr>
<tr>
<td>• Some information hiding; scoping rules restrict use of data within scope</td>
<td>• Platform independent</td>
</tr>
<tr>
<td>• Supports functional programming</td>
<td>• Standard packages for IO, elementary mathematical functions, and string handling</td>
</tr>
<tr>
<td>• Tied to UYK computers</td>
<td>• Command line interface</td>
</tr>
<tr>
<td></td>
<td>• Supports recursion and reentrancy</td>
</tr>
<tr>
<td></td>
<td>• Supports software engineering principles</td>
</tr>
<tr>
<td></td>
<td>• Supports programming in the large</td>
</tr>
<tr>
<td></td>
<td>• Supports mission-critical and safety-critical applications</td>
</tr>
</tbody>
</table>
3. SUMMARY OF TRANSLATOR/TRANSLATION RESULTS

TRANSLATOR PROFILES

Table 3-1 shows a profile of the three translators. This profile includes the translator points-of-contacts, major characteristics of the translators, and summary of the results of the evaluation. Table 3-2 summarizes translator results.

For details on these results presented and for additional results, we suggest that the reader turn to the results appendices of this report.

CONCLUSIONS

The following are the significant conclusions from the translator evaluation.

1. The overall complexity and the distribution of the complexity across the translator-produced Ada modules was similar to the corresponding CMS-2 code. This suggests that each of the translators took a transliteration approach to translation. The McCabe and Halstead metrics show that the complexity of the translator-produced code mirrors the complexity of the CMS-2 code. The translators do not introduce or reduce complexity.

2. The overall complexity and the distribution of complexity across the translator-produced Ada modules was very similar across translators. This suggests that each of the translators took a similar approach to translation and to the distribution of control and data. The McCabe and Halstead metrics show the similarity in complexity.

3. Most of the programs produced by the translators required manual reengineering to compile and execute successfully.

4. The translators all produced programs that contained many features (e.g., GOTOs, “use clause”, subprogram exceeds 200 SLOC) that conflict with the Software Productivity Consortium (SPC) programming style guidelines (Software Productivity Consortium, 1992). The vast majority of these features appear to reflect characteristics of the CMS-2 ancestor program. The non-compliant code is similar across translators.

5. There was little difference among the translators in the degree of difficulty to perform conversions of CMS-2 to Ada (person-hours and SLOC changed). There were problems with each because Ada 83 does not include standard mathematical functions. (This is not a problem for Ada 95 since mathematical packages are now part of the standard.) There were problems executing the Ada on Suns because the requested range of a floating point type produced exceeded the platform limitations. Changes had to be made to the code produced by each translator. These are described in Appendix A, C, and F.

6. The person-hours and Source Lines of Code (SLOC) changed or added shown in Appendix C, may be useful in making “ball park” estimates of the effort required to translate a CMS-2 application. However, the CMS-2 sample upon which these metrics were based contained no direct code, overlays, or special device IO.
7. The object-oriented features of standard Ada (Ada 95) enhance the potential of a redesign and rewrite of low quality CMS-2 applications in ways that dramatically reduce control complexity and program size. This conclusion is based on an experiment to redesign and manually rewrite QA9 in Ada 95. The quality of the redesigned and rewritten application was far superior to the translated applications as indicated by Halstead and McCabe metrics and the conformance to Software Productivity Consortium style guidelines measured by Logiscope.

8. There were catastrophic failures by all translators during stress testing. The developers were very responsive in fixing these translator deficiencies with an average turnaround of two working days. By the end of testing, only two catastrophic failure conditions remained in final translator revisions for this test set. These were QA7A for CCCC and MK-2 for TRADA. Reference Tables B-1 and B-3.

9. The quality of Ada source code produced by the translators is of low quality and difficult to modify and extend. Many Ada style guidelines were violated because the translated code closely mirrors the CMS-2. Problems included the use of GOTO statements (all), use of “use clause” (APL, CCCC), predefined information that is produced but not needed (APL, CCCC), packaging that is difficult to understand since it was not done by a human (all), excessive use of pointers (CCCC), and others that are described throughout the report.

10. The person-hours per 100 CMS-2 statements (delimiting $s) required to translate and successfully execute the QA9 sample in Ada when using the Sun Ada compiler were: APL, 1.37 person-hours; CCCC, 1.91 person-hours; and TRADA, .62 person-hours. Expect the translation of deployed CMS-2 systems to require a lot more time. The QA9 did not include IO to special devices, direct code, or overlays. For details on how these numbers were calculated see Appendix G: Table G-6, Table G-7, and the discussion of these tables.

11. Translated code, intended to evolve and be maintained, would require significant reengineering. The best translation had about a 2:1 SLOC expansion; the worst translation had about an 8:1 SLOC expansion. A hand reengineering into Ada of the original CMS-2 code had about a .5:1 SLOC expansion. The translated code had serious deficiencies in the use of naming conventions, elimination of intermediate variables, use of standard packages, memory management, performance, and position to reengineer. The comparative analysis along with source code for each system is provided in Appendix M.
<table>
<thead>
<tr>
<th>Characteristics</th>
<th>APL</th>
<th>CCCC</th>
<th>TRADA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Version</td>
<td>Rev. 2.8</td>
<td>Ver 6.1, Rev. 071196</td>
<td>PBL1.0</td>
</tr>
<tr>
<td>Host Computer/OS</td>
<td>Sun OS</td>
<td>VAX VMS</td>
<td>VAX VMS</td>
</tr>
<tr>
<td>User documentation for running translator</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Assistance needed in running translator</td>
<td>Some required</td>
<td>Some required</td>
<td>None required</td>
</tr>
<tr>
<td>Support for translator development/translation assistance</td>
<td>Translator not currently funded by Navy/must be funded by project</td>
<td>Translator not currently funded by Navy/must be funded by project</td>
<td>Translator not currently funded by Navy/must be funded by project</td>
</tr>
<tr>
<td>Developer says CMS-2 construct translates</td>
<td>All</td>
<td>Listed in Section 7 of CCCC user documentation (CCCC, 1996)</td>
<td>Listed in Section 3.8 of TRADA user documentation (CSC, 1994)</td>
</tr>
<tr>
<td>Ada Packaging</td>
<td>Produces one specification and one body</td>
<td>Produces monolithic package with nested packages</td>
<td>Produces multiple specifications and bodies</td>
</tr>
<tr>
<td>Files Produced</td>
<td>Specification in one file and body in second file</td>
<td>All specification and bodies in one file</td>
<td>Each specification and body in separate files</td>
</tr>
<tr>
<td>Availability/Cost to Acquire</td>
<td>Product or Services</td>
<td>Contact Vendor</td>
<td>Contact Vendor</td>
</tr>
</tbody>
</table>

| Vendor Representative                               | James G. Palmer               | Noah Prywes                   | Joe Whalen/Richard Brimson     |
| APL                                                 | APL                           | CCCC                          | CSC                            |
| Room 6-105                                          | Room 6-105                    | Room 6-105                    | Room 6-105                     |
| Laurel, MD 20723                                    | Laurel, MD 20723              | Laurel, MD 20723              | Laurel, MD 20723               |
| (301) 953 6800                                      | (301) 953 6800                | (301) 953 6800                | (301) 953 6800                 |

| CCCC                                                | 2300 Chestnut St.             | 2300 Chestnut St.             | 2300 Chestnut St.              |
|                                                     | Suite 230                     | Suite 230                     | Suite 230                      |
|                                                     | Philadelphia, PA 19103        | Philadelphia, PA 19103        | Philadelphia, PA 19103         |
|                                                     | (215) 854-0555                | (215) 854-0555                | (215) 854-0555                 |

| Division                                            | CSC                            | CSC                            | CSC                            |
| Applied Technology Division                         | Applied Technology Division    | Applied Technology Division    | Applied Technology Division    |
| 4045 Hancock Street                                 | 4045 Hancock Street           | 4045 Hancock Street           | 4045 Hancock Street           |
| (619) 225-8401                                      | (619) 225-8401                | (619) 225-8401                | (619) 225-8401                |

3-3
### Table 3-1. Translator Profiles - 2

<table>
<thead>
<tr>
<th></th>
<th>APL</th>
<th>CCCC</th>
<th>TRADA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Predefineds</strong></td>
<td>Provides predefined package specification and body containing commonly used types and functions (BASIC_DEFNS)</td>
<td>Provides predefined package specification and body containing commonly used types and functions (PREDEFINEDS)</td>
<td>Generates customized predefined package specification containing only types and functions needed (CMS-2 Types)</td>
</tr>
<tr>
<td><strong>Math library used</strong></td>
<td>Sun math library</td>
<td>VAX math library</td>
<td>User must Provide(^1)</td>
</tr>
<tr>
<td><strong>Control of translation process &amp; outputs</strong></td>
<td>Uses switches to control user options</td>
<td>No control (Always produces “use clause”)</td>
<td>Uses script file to control user options (e.g., “use clause” may be on or off)</td>
</tr>
<tr>
<td><strong>Termination and placement of errors</strong></td>
<td>Continues translation regardless of errors, brackets errors and non-translatables in Ada comments</td>
<td>Continues translation regardless of errors, brackets errors and non-translatables in Ada comments</td>
<td>Depending on errors encountered may stop processing &amp; notify user. Some error are annotated in Ada comments, some placed in summary file</td>
</tr>
<tr>
<td><strong>Other Characteristics:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

\(^1\) TRADA generates math functions which return the value of 1.0. It is up to the user to implement the correct functionality of each math function or use the one provided in Ada 95.
Table 3-2. Summary of Translator Evaluation Results

<table>
<thead>
<tr>
<th>Quick Look+Reengineering Results (QA9)</th>
<th>APL</th>
<th>CCCC</th>
<th>TRADA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weighted McCabe cyclomatic complexity for Ada QA9 produced by translators</td>
<td>65</td>
<td>67</td>
<td>66</td>
</tr>
<tr>
<td>Executable statements</td>
<td>3642</td>
<td>3887</td>
<td>3759</td>
</tr>
<tr>
<td>Stress Testing Results</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stress testing catastrophic failures</td>
<td>11</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>Stress testing Ada SLOC produced</td>
<td>468.9K</td>
<td>925.7K</td>
<td>385K(598.9K)</td>
</tr>
<tr>
<td>Wall clock time for running stress tests</td>
<td>4 hr 42 min</td>
<td>31 hr 59 min</td>
<td>6 hr 22 min(9 hr 30 min)</td>
</tr>
<tr>
<td>Percentage of clean compiles out of 84 stress test files</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAX Ada</td>
<td>1% (1/84)</td>
<td>17% (14/84)</td>
<td>29% (24/84)</td>
</tr>
<tr>
<td>GNAT</td>
<td>1% (1/84)</td>
<td>12% (10/84)</td>
<td>29% (24/84)</td>
</tr>
<tr>
<td>Sun</td>
<td>1% (1/84)</td>
<td>12% (10/84)</td>
<td>26% (22/84)</td>
</tr>
<tr>
<td>Percentage of output files produced for 84 input stress test files</td>
<td>100% (84/84)</td>
<td>99% (83/84)</td>
<td>64% (54/84)</td>
</tr>
</tbody>
</table>

1 See Appendix C for more details
2 Catastrophic failures were defined as core dumps, trace backs, infinite loops, and empty Ada output file with no notification.
3 The number is actual. The number in parenthesis is an extrapolation (if Ada code would have been produced for all 84 files).
4 See Table B-2 for more details
5 See Table B-1 for more details.
4. LESSONS LEARNED AND OPINIONS

LESSONS LEARNED

1. Translation from CMS-2 to Ada requires a very strong expertise in CMS-2, the application program being translated, and Ada. Do not attempt it without expertise in all three areas. Training in the use of the translators and tools is desirable.

2. Translation from CMS-2 to the current standard, Ada 95, is easier and faster than to Ada 83 because Ada 95 includes the standard mathematical functions. Ada 83 did not include a floating point exponent which was required by the sample code taken to execution in Ada (QA9). Ada 95 is also preferable because it supports modern software engineering capabilities (e.g. object oriented programming improves interface capabilities, and real time programming enhancements).

3. Translators were advertised (intended) to generate correct compilable Ada code. Trial compiling of generated Ada during translator evaluation showed that this was often not true. (Remember that non-translatables, such as direct code, are bracketed inside Ada comments and will not “dirty” a compile.) During Stress Testing correct compiles occurred no more than 44% of the time (See Table B-2).

4. Translation installation instructions were adequate to good. We needed no help from the Computer Sciences Corporation to install and run the TRADA translator. Some assistance was needed with the APL and CCCC translators. An NRaD software engineer, who participated in the evaluation, was already very familiar with TRADA.

5. Other tools not used in the translator evaluation may also be useful in the translation process. Clue is a reverse engineering tool developed by Mitre that draws flow diagrams from CMS-2 source code. The Design Analyzer calltree feature was not used but may be useful. The Rational Reengineering Toolkit looks promising for restructuring translated Ada source code.

6. After the environment was established for each translator, the translations were easier than expected. The translator’s environment includes logicals, command files, and linking. We did not need any formal training.

7. Catastrophic failures were found in all translators during testing.

8. The Synetics Assembler Design Extractor (direct code to CMS-2 translator) only executed correctly on its demonstration program. It was unsuccessfully executed on samples chosen from the QA tests and project test cases.

9. Halstead and McCabe metrics did not enable us to qualitatively distinguish between translator outputs. This is largely due to the fact that the translator vendors took a "transliteration" approach to translation. As a consequence, source code content and structure was very similar. Halstead and McCabe metrics did show that the complexity of the Ada code produced by the translators mirrored the CMS-2 code. McCabe was very useful in comparing the complexity of translated Ada versus redesigned/rewritten Ada.

10. Comparing SLOC between Ada and CMS-2 indicated that the translators did not raise the level of abstraction during translation. That is, they tended to pick one or more Ada features for each CMS-2 feature. Other than indicating that, SLOC was not a particularly useful metric. It is possible for a module with a smaller SLOC

4-1
count to have more complex expressions than another and be more difficult to understand. It is even possible for a module with a larger SLOC count to be more efficient than one with fewer SLOC. A trivial example is one in which a loop is unrolled and inlined. It is also possible for a module with more comments to have fewer meaningful comments. For example, Ada-ASSURED inserts a line of dashes between subprograms in a package as part of its formatting capability. This raises the "comment count" substantially without adding any meaning whatsoever.

11. SLOC comparisons between Ada and CMS-2 had to be done with care. SLOC was counted several ways: as straight editor lines of code in both CMS-2 and Ada, as delimiting dollar signs ($) in CMS-2 and delimiting semicolons (;) in Ada. Three different kinds of comments were counted in CMS-2 (including the one for compile listing formatting) while in Ada there is only one kind of comment. We also had to figure out how commercial analysis tools, like Logiscope, counted lines so that comparisons of weighted metrics between CMS-2 and Ada source were valid.

12. A project should expect the translated Ada source lines of code to be greater than that for the corresponding CMS-2 code. For example, Table B-4 (last page) shows that for the 84 QA files used in stress testing, the increase in code size is more than 2:1 (Ada:CMS-2) for TRADA, slightly less than 2:1 for APL and almost 4:1 for CCCC. These SLOC counts are lines as counted by an editor and include comments and blank lines. The predefined functions and utilities produced by the translators are included in these line counts. The ratios in SLOC count vary from project to project. The translated Ada SLOC count will always exceed the CMS-2 SLOC count. One might expect the source lines of code for Ada code reengineered by hand to be approximately half of the CMS-2 code.

13. The evaluation process did not address the issue of target platform. For example, the Quick Look sample tested mathematical operations for UYK computers and some of the floating point type declarations reflected this. However, such a test makes less sense if the target is a Sun Workstation. The translators should be "parameterized," for specific targets, or for portability.

14. We found that approximately 90% of the time when translated Ada code compiles with one of the three compilers, it will compile with no changes or with minor changes using the other two compilers (VAX, Sun and GNAT).

15. Metrics used to measure the effort required to take translated code through successful compilation and execution were biased. Person-hour were biased by (1) the order in which QA9 samples taken through compilation and execution and (2) the order in which samples were compiled by the three Ada compilers. The difficulty of conversion metric that counted SLOC modified or added until successful compilation and execution were achieved was biased. Some code changes were much easier to make than others (e.g., finding the cause for a single "program error" was more difficult than making fixes to many lines of code where the translator produced a floating point exponent which is not allowed in Ada 83.) How you count lines of code modified when a segment of code is moved from one location in a program to another can also bias this metric. Future related studies need to be aware of theses biases so that metrics that measure level of effort can be improved.
16. Translated code, intended to evolve and be maintained, would require significant reengineering. The best translation had about a 2:1 SLOC expansion; the worst translation had about 8:1 SLOC expansion. A hand reengineering into Ada of the original CMS-2 code had about a .5:1 SLOC expansion. The translated code had serious deficiencies in the use of naming conventions, elimination of intermediate variable, use of standard packages, memory management, performance, and position to reengineer. See Appendix M for details.

OPINIONS

1. The CMS-2 to Ada translator developers were all very responsive in fixing translator problems with an average repair turnaround of two working days. By the end of testing, only two catastrophic failure conditions remained in final translator revision for this test set. These were QA7A for CCCC and MK-2 for TRADA.

2. Translation is well-suited for stand-alone algorithms free of direct code.

3. The Quick Look and Reengineer Until Ada Code Executes Correctly translation phases demonstrated that automatic translation of general purpose programming constructs from CMS-2 to Ada is feasible. However, if there are plans to maintain the translated code for some time and to extend it, be aware that quality improvements are needed and that translator produced code is more difficult to understand than code produced by humans. Of the three translators, we found the CCCC produced Ada code to be the most difficult to understand because of the extensive use of pointers. Quality improvements that are needed to make translated code easier to understand include less use of access types (CCCC), elimination of GOTOs (all), improved packaging (APL), elimination of “use clauses” not used (APL, CCCC), elimination of variables that are defined but not used (all), and moving declarations and type definitions down to the appropriate level for the purpose of information hiding (all).

4. Correct translation of Ada can be validated more easily when it has not been restructured. We can visually compare the Ada and CMS-2 source code. We believe that many source code quality improvements are best handled following translation. Tools that make these quality improvements have wide application and are certainly useful for more than just translation efforts. Some potential post-translation quality improvements that can be done by tools include the removal of GOTOs and other restructuring, elimination of variables that are declared but not used, elimination of dead code, and automated information hiding (moving declarations and type definitions down to reduce visibility).
5. RECOMMENDATIONS

This section provides recommendations to CMS-2 project managers, to the Navy for advancing translator technology, to translator vendors, and to tool vendors.

RECOMMENDATIONS TO CMS-2 PROJECT MANAGERS WHEN CONSIDERING TRANSLATION

1. Do not translate unless expertise is available.

   Expertise is needed in CMS-2, the application being translated (in the same person), and in Ada. Assistance from translator experts is desirable.

2. If seriously considering translation, do it soon.

   CMS-2 experts are reaching retirement age. CMS-2 analysis tools and some CMS-2 translators are no longer supported. The availability of the translators in the future is uncertain.

3. Expect translation to be difficult and time consuming.

   The effort will probably include the manual translation of some CMS-2 code, the manual translation of direct code, the preparation of new documentation, and learning how to use the translators, and analysis tools. Much will need to be redesigned and rewritten to newer software and hardware technologies. The following examples will require significant program redesign:

   a) Memory - CMS-2 uses memory overlays while modern systems use virtual memory. Conversion of overlays to relocatable objects is error prone. Attempts to use the desired stack memory model will introduce errors when side effects of CMS-2 memory overlays were used (this was frequently done).

   b) System Calls - CMS-2 used Executive Service Routines (ESRs) to interface with the underlying Executive (Operating System). There is not always an easy or correct mapping of ESRs to services in Portable Operating System Interface (POSIX) compliant environments or in the Ada Run Time Executive. Translators do not attempt to replace ESRs with logical modern system services. Instead comments are inserted indicating that the user must do this.

   c) Library Calls - CMS-2 used Common Service Routines (CSRs) for common function such as mathematical functions. Translators do not attempt to replace CSRs with logical modern library services. Instead comments are inserted indicating that the user must do this.

   d) I/O - CMS-2 used very low level primitives to effect I/O. Modern systems have high-level commands and use change of representation clauses to efficiently process data internal to the computer yet transmit/receive data in the format agreed within the interface specification. Practically every I/O mechanism will need to be redesigned in order to be integrated onto hardware and software systems.

5-1

Use analysis tools such as the CMS-2 Source Code Design Analyzer (DESAN) and CMS-2 Source Code and Metrics Generator (METRC). These tools and user documentation are available as freeware from NReD. These tools were developed by the Computer Sciences Corporation with funding from the Ada Technology Insertion Program, Advanced Combat Direction System and other projects.

DESAN was designed to assist in the reengineering of CMS-2 code prior to translation to Ada. It identifies overlays, identifies data units that are defined but not referenced, and identifies data units that are referenced but not set to a value. The tool also examines the scope of variables and makes recommendations to reduce it.

METRC produces source code statistics (e.g., SLOC for CMS-2 and direct code, source statements in DDs and SYSROCS), a keyword report, and Halstead and McCabe complexity metrics.

a) Use these tools to acquire a profile of all code segments for which translation is considered. This includes identifying the quantity of direct code, overlays, bit-level manipulations, dead code, complex code, and IO operations. Dead code should removed. Complex code can be translated but is a strong candidate for redesign. Other categories will have to be dealt with manually.

b) Visually examine the impact of executive and common service routines (e.g., peripheral devices, debugging aids, data extraction capabilities). Calls to service routines will not translate with translators.

5. Determine how to handle replacement or translation of the executive operating system.

Use of ESRs should be evaluated to determine the most appropriate replacements for operating system services or run-time system services.

6. Consider replacing CSRs with common Ada libraries (e.g., math packages).

7. Expect to possibly do some reengineering before translation and to do reengineering afterwards.

Reengineering of CMS-2 can increase the percentage of translatable code. Extraction or isolation of low-level segments and other non-translatables from otherwise translatable segments will facilitate the translation process.

8. View IO as an area that needs complete redesign.

Translators will mark and bypass all low level IO. All CMS-2 IO programming is low-level.

9. Make a cost estimate for translating your CMS-2 system.

10. Evaluate cost-schedule-quality tradeoff for translation versus redesign (See Figure 5-1).

This will involve answering questions such as, do I: use as-is, translate, redesign the project for new technology and a new language, or start an entirely new project at the requirements phase.
11. Do not translate a CMS-2 system that does not execute correctly in CMS-2.

    Problems in the initial system will transfer and will be compounded by translation.

12. If major enhancements are scheduled to the existing software strongly consider redesign.

13. When a substantial amount of new code will be written it probably makes more sense to redesign and rewrite rather than to continue with the legacy design.

14. Do not do translate unless there is strong time and money commitment from the sponsor.

15. Translate stand-alone algorithms.

    Automatic translation is well suited for translating stand-alone algorithms that are free of direct code (e.g., Kalman filters)

16. Be careful about pilot testing on project code for examining translation feasibility

    Results may underestimate the effort. For example, when translated Ada code is compiled, counting the initial set of compilation errors is not an accurate indicator of the magnitude of the effort required to achieve correct compilation. Many compiler errors may be the result of a few problems or after fixing the first set, new ones may appear. Also, obtaining correct compilation is much easier than achieving correct execution in Ada.

RECOMMENDATIONS TO PROJECT MANAGERS AFTER DECIDING TO USE TRANSLATOR TECHNOLOGY

1. Have your experts on board from the start of the translation process.

    This minimally includes your CMS-2 application expert and Ada expert. Also, include in your schedule, time for your software engineers to learn how to use the translators and analysis tools.

2. Translate to Ada 95.

    Use one of the three translators evaluated that translate CMS-2 to Ada. Compile with an Ada 95 compiler because it includes the standard mathematical functions and supports additional software engineering capabilities (e.g. object oriented design).

3. Select a translator based on the translator profiles. See Section 3.

4. Consider CMS-2 reengineering to eliminate overlays, direct code, and to simplify procedures that are overly complex. CMS-2 analysis tools listed in Table 2-1 will be helpful.

    This will improve the quality of the translated Ada and the percentage of CMS-2 that is translatable.

5. Reengineer to eliminate bit manipulation.
Bit manipulation in CMS-2 source code should be analyzed to determine why it is being done. It may be unnecessary on the new target. For example, if the new target platform were the Global Command and Control System (GCCS) the same capability may already be handled by the core services. It may also be unnecessary if it is being done to conserve memory, and the new target is a virtual memory computer or has fewer memory constraints.

6. Use analysis tools:
   a) CMS-2 analysis tools (e.g., CMS-2 Source Code Metrics Generator, CMS-2 Source Code Design Analyzer)
   b) Ada quality analysis tools (e.g., Ada-ASSURED, Logiscope, AdaMat, AdaQuest)
   c) Ada reengineering tools (e.g., Reengineering Toolkit by Rational and Hyperbook by CCCC)

   The Reengineering toolkit and Hyperbook were not used in the translator evaluation.

7. Decide in advance where to recertify.
   If the CMS-2 software is reengineered then the CMS-2 software should be recertified before translation. The Ada must be certified. Doing it this way will reveal any problems more quickly.

RECOMMENDATIONS TO THE NAVY FOR ADVANCING TRANSLATOR TECHNOLOGY

1. Poll managers of deployed CMS-2 systems.

   This will assist decision-making with regard to whether to continue funding CMS-2 translator development and maintenance and whether to fund development of CMS-2 "direct code" translation.

   Ask managers of deployed CMS-2 systems the following questions:

   a) How many lines of CMS-2 code and how many lines of direct code are there in your system?

   b) What are your intentions with your CMS-2 system over the next five years?

   I. Use "as is"?,
   II. Use automatic translation from CMS-2 to Ada 95, to C++ or to another high-level programming language? If so, to which language?
   III. Redesign and rewrite in a Ada 95, C++ or another high-level programming language? If so, which language?
2. Support development of Ada quality improvement tools.

These tools are useful for improving the quality of translated Ada code as well as the quality of legacy Ada code (e.g., removal of GOTO statements, removal of dead code, conversion of global objects to local objects, elimination of subprogram side effects, creation of meaningful types, creation of meaningful names, and repartitioning code into packages). The user community for these capabilities is more than just CMS-2 to Ada projects. These quality improvements are needed by projects that use Ada generated by translators whose input is a language other than CMS-2 as well as projects that use poorly written Ada programs. Most of these improvements are not provided by existing tools.

3. Support translator improvements that improve the quality of Ada produced.

These are improvements that do not hinder the use of existing CMS-2 test designs and test data. The translation approaches used by the three translators was to not make significant structural modifications to the Ada code produced. This allows CMS-2 test designs and test data to be applied to the translator-produced Ada. Hence it easier to demonstrate functional equivalence. Examples of these improvements include, removing unnecessary context clauses, removing the “use clause”, producing code that is target-independent, and other improvements described in recommendations to translator vendors.

4. Perform in-depth analysis of MTASS compilation errors.

During Stress Testing, translated MTASS QA tests were compiled and checked for errors. Time permitted only a high-level examination of the compilation errors. A more in-depth examination is needed to determine the spectrum of errors and the effort required to obtain correct compilations. Information gathered from this analysis will help translators generate higher quality Ada programs.

5. Develop translation cost schedule models.

These are needed to assist the project manager in estimating translation cost and time. Based on parameters such as project size, complexity, and remaining life cycle, a project manager can decide whether to translate or redesign in Ada.

6. Develop methodology to replace CMS-2 overlays and bit manipulations (automated or manual).

Some CMS-2 constructs, such as overlays and bit manipulations, do not translate or translate awkwardly. This methodology will substitute non-translatable CMS-2 with CMS-2 code that is translatable.

7. Consider the cost saving benefits of redeveloping or reengineering a collection of applications as a whole.
When a collection of applications within a domain is to be transported, an opportunity may exist to substantially reduce the transportation cost of the collection as a whole compared to the cost of transporting each application individually. Cost savings may be achieved by reengineering in accordance with different software architecture principles such as client-server or object-oriented if multiple applications can use the products of the effort. Cost savings can also be achieved by developing or using domain-specific components which may be shared by multiple applications.

8. Consider developing a decision-making strategy based on product quality and business value for determining what CMS-2 applications to continue to use “as is” in CMS-2, translate to Ada, discontinue using the product, or redesign/rewrite in Ada.¹

Sneed (1995) suggests a metrics-based approach in which applications are ranked according to their business value and technical quality. Technical quality is related to such things as complexity, modularity, testability, understandability, and availability of meaningful documentation. Business value is importance to the Navy. Technical quality and business value are assigned numerical scores. Figure 5-1 is a visual framework for making reengineering decisions. The following is one high-level decision strategy based on these rankings. The letters below are the quadrant letters in the table.

a) Continue to use CMS-2 “as is” until obsolete (for example, a better product takes its place or UYK computers are no longer used)

b) Redesign and rewrite in Ada

c) Discontinue using product

d) Translate to Ada and reengineer for maintainability

¹ The 84 QA tests used for stress testing, Appendix B, lie in the low quality, high value quadrant. We were able to significantly improve the quality of QA9 with a redesign and rewrite in Ada 95. See Appendix C, Ada 95 QA9: Reengineering a mixed-mode math test in Ada 95.
### Figure 5-1. High-level strategy: translate, reengineer, both, or discontinue

<table>
<thead>
<tr>
<th>Technical quality</th>
<th>Business value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>High quality,</td>
<td>Low quality,</td>
</tr>
<tr>
<td>low value</td>
<td>high value</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>Low quality,</td>
<td>High quality,</td>
</tr>
<tr>
<td>low value</td>
<td>high value</td>
</tr>
</tbody>
</table>

**RECOMMENDATIONS TO TRANSLATOR VENDORS**

**ALL VENDORS**

1. Minimize global interfaces/declarations.

   The only declarations that should appear in the visible part of a package specification are those objects and services that are required for use by clients of the package. In the case of a monolithic package like the APL Qa9qlook package, the only entity required by an external client is "procedure Driver." Qa9qlook is the Ada package produced by the APL translator when translating QA9 during Quick Look (Appendix A). All of the other declarations in the specification of package Qa9qlook are services of other clients in package Qa9qlook. They should not appear in the specification of Qa9qlook. Superfluous visibility is confusing.

2. Avoid use of nonstandard or proprietary math libraries.
The APL and CCCC translators produced source code that relies on nonstandard or proprietary math libraries. The TRADA translator generated completely portable code, but failed 82 tests due to Ada 83's lack of an exponentiation operator with a floating point exponent. Ada 95 contains Ada.Numerics.Generic_Elementary_Functions package (ISO, 1995) which contains the math functions required for the Quick Look tests. The functions in this package should be used to the exclusion of all other math functions when they meet accuracy and efficiency requirements. APL used the Sun math library, CCCC used the VAX math library and TRADA did not use a math library.

3. Consider using unsigned integers with modular types.

Each of the translators defined a number of unsigned integer types or subtypes in their predefined packages. The Ada 83 standard did not support unsigned integers, however, Ada 95 does in the form of modular types (ISO, 1995). The translator developers should consider replacing the existing definitions with definitions using modular types. The following code fragment illustrates this capability.

```plaintext
package Unsigned_Integer is
  type U1 is mod 2**1;
  type U2 is mod 2**2;
  ...
  type U32 is mod 2**32;
end Unsigned_Integer;
```

4. Produce portable Ada code.

The translators should be "parameterized" for specific targets (OS, computer, and compiler) or for portability, and should not necessarily target the UYK architecture. CCCC and TRADA produce UYK-oriented Ada code that will only run unmodified using VAX Ada. For example, for QA9, TRADA produced a floating point number that was too large for a Sun but not for a VAX.

5. Thoroughly test translators using the MTASS test suite

The translator evaluation team found many translator bugs when using MTASS during stress testing. Vendors should translate the entire MTASS test suite and try compiling the Ada produced using an Ada 95 compiler.

**APL**

1. Avoid monolithic packages.
Make better use of Ada's package concept. Among its benefits is its use as a modularization mechanism. Single large packages are more difficult to comprehend and maintain than several smaller compilation units.1

2. Eliminate the “use clause”.

Rather than the “use clause”, a better solution is to make judicious use of package renaming and the Ada 95 “use type clause”.2 We recommend that APL and CCCC include a switch to turn off “use clauses”.

CCCC

1. Avoid access before elaboration.

Avoid calling subprograms before they are elaborated. The module structure generated from the CCCC translator is one in which all of the code for a program which is not included in “PREDEFIN.ADA” is declared somewhere in a single package. This approach imposes limitations with respect to elaboration order and software maintenance. One problem is that variables declared in package specifications cannot be given default values returned from functions implemented in the body of that package.3 This is referred to as access-before-elaboration (ISO, 1995). Ada implementations are required to be able to detect this condition and raise the program_error exception. This problem occurred in two places in the CCCC QA9 program. One simple and straightforward solution is to avoid nested packages, perform variable initializations in the initialization section of the body, and to include "pragma Elaborate_Body;" (ISO, 1995) in the package specification.

It should be kept in mind that the APL and TRADA translators managed to generate a correctly working version of QA9 without resorting to access types, addresses, or unchecked programming. This demonstrates that these questionable techniques were unnecessary.

Additional Thoughts on the Use of Pointers

The CCCC translator uses access types extensively to deal with the overlay problem. In CMS-2, when memory became tight, objects would share memory name space with other objects. This was a very dangerous practice, but necessitated by the severe limits on memory during the 1970s and early 1980s. Programmers could change the value of any of the named objects and the effect would be to change the value of all the named objects. Today memory is very inexpensive and virtual memory models are used by most hardware environment and supported through most computer languages.

---

1 See “Access before elaboration”
2 See Appendix D, section D.4.1.
3 Instantiations of unchecked_conversion do not generate executable code in many cases. In those that do, they do not depend on code implemented in the body of the unit in which they are instantiated.
Ideally, the translation process should resolve names for each of the objects so that each object has a unique name space. In most languages this is achieved using a virtual memory model via the stack. Here the physical address of an object will vary based on its environment at the time the object was placed on the stack. If its value is to be shared with another object, it must be done explicitly via periodic assignment statements. The use of stacks are considered very safe for safety-critical and mission-critical applications.

Most languages also provide a heap memory using pointers (i.e., access types). There are certain operations such as list processing which are facilitated by pointers. The use of heap memory requires additional memory management functions during real-time and is very dangerous as memory can become easily fragmented requiring garbage collection.

Instead of resolving the dangerous consequence of overlays, the CCCC translator converts the object to a pointer (access type) so that the name space of objects are overlaid in the translated environment. This necessitates the use of unchecked conversion as each access type is likely to have a different type with different legal values.

The advantage of using pointers is that object name space resolution does not have to be performed automatically. On occasion a CMS-2 programmer would take advantage of the side-effects of overlays allowing the change of value of one object to also change the value of other objects. This is bad practice, but frequently done. Hence, the use of pointers will provide a correct solution in the face of poor programming practices. Unfortunately, the translated code is not easily understood nor maintained as it continues the legacy of bad programming practices.

Perhaps for those situations where suspected side effects are used, the translators should generate normal Ada objects with a comment to the effect:

“In the CMS-2 program, Object_A and Object_B pointed to the same memory location; please check for side effects.”

2. Avoid monolithic packages.

Make better use of Ada's package concept. Among its benefits is its use as a modularization mechanism. Single large packages are more difficult to comprehend and maintain than several smaller compilation units.¹

3. Eliminate superfluous context clauses.

The presence of superfluous context clauses (e.g., with Package_Name) is confusing because it implies that certain services are required by a client when, in fact, they are not. This places the unnecessary burden on maintenance personnel of proving that such services are irrelevant to their maintenance tasks.

4. Eliminate the “use clause”.

Eliminate the “use clause”. A better solution is to make judicious use of package renaming and the Ada 95 “use type clause”.¹

¹ See “Access before elaboration”
RECOMMENDATIONS TO REENGINEERING TOOL VENDORS

Develop tools that will automatically or semi-automatically improve the quality of legacy Ada or Ada produced by translators. Some examples of these capabilities are listed below. We are not aware of existing tools that perform these operations on the Ada code.

• Remove GOTO statements

  All three translators created Ada source with GOTO statements whenever the corresponding CMS-2 source contained GOTOs. A capability is needed to automatically remove GOTOs by producing functionally equivalent Ada that is maintainable. (METRC should be used to detect the presence of GOTOs in CMS-2, which guarantees their presence in the Ada.)

• Remove dead code

  Programs with dead code are confusing and difficult to maintain. A capability is needed that automatically removes or flags dead code. (DESAN can be used to flag dead CMS-2 code for pre-translation reengineering).

• Convert global objects to local objects

  As the CMS-2 COMPOOL construct is equivalent to the creation of global objects, all translated code should be analyzed for placing objects at the appropriate location. A portion of this should be done automatically. See next item.

• Eliminate subprogram call side effects to global objects

  All subprograms should operate on local objects only. Most CMS-2 procedures and functions operate on global objects making side effect detection a very difficult task. Subprogram calls should pass all affected objects as parameters, eliminating the possibility of dangerous side effects. This conversion could be done automatically. (DESAN can be used to make scope change recommendations in the pre-translation CMS-2.)

• Perform automated information hiding

  A capability is needed to automatically push type definitions, variable declarations, and subprogram declarations down to the appropriate level. Translators do not do a very good job of producing Ada source that takes advantage of information hiding. For example, variables and subprograms are sometimes declared in a package specification when they are only used in the package body. A tool could automatically improve the information hiding.

1 See Appendix D, section D.4.1.
However, there are some valuable Ada reengineering capabilities provided by tools that exist today that were not used during this evaluation. For example, the Rational Reengineering Tool Kit provides a capability for 1) creating meaningful types, 2) creating meaningful object names and 3) for repartitioning code into packages. CCCC's Hyperbook processes Ada source code to produce a collection of hyper-linked graphics and text that is viewable in a web browser. This information helps the programmer to more quickly understand the Ada source code. Proposed research using these tools is discussed in Appendix E.

SUGGESTED TRANSLATION STEPS

We assume that the goal in translation is to produce correctly executing Ada software that is maintainable. The steps of obtaining, installing, and learning to use the tools mentioned are not listed. A description of the Ada analysis tools is found in Appendix E. Some were used in this experiment.

Inspect and Prepare CMS-2 Source Code

1. Determine Feasibility of Translation by following the sub-steps below.
   a) Count lines of CMS-2 and direct code using the CMS-2 Source Code Metrics Generator (METRC). Visually examine code to see if direct code has equivalent CMS-2 functionality in comments.
   b) Gather complexity metrics. METRC produces McCabe Cyclomatic and Halstead Complexity metrics. Analysis can be on SYSPROC, SYSDD, or entire system.
   c) Gather processing flow analysis data. The CMS-2 Source Code Design Analyzer (DESAN) produces both long and short call trees. Analysis can be on SYSPROC, SYSDD, or entire system.
   d) Identify use of dead code, and scoping using DESAN.
   e) Identify use of overlays using METRC.
   f) Examine use of executive and common service routines and other non-translatable aspects. This step is done by visual examination, probably by using a text editor.
   g) If possible, run Logoscope CMS-2 to further examine the quality of the CMS-2 code. NRaD did not use the Logoscope CMS-2 capability. (The CMS-2 analysis capability is an add-on to Logoscope that may be purchased. It produces Halstead, McCabe and other metrics.)
   h) Consider using Clue to help understand CMS-2 code. This prototype CMS-2 reverse engineering tool produces data flow diagrams, control flow diagrams and reports that assist the programmer in understanding CMS-2 source code.

2. Identify CMS-2 Code Segments Suitable for Translation. Select segments based on:
a) Minimal quantity of direct code (where equivalent CMS-2 does not exist in comments)
b) Minimal use of overlays, executive service calls, IO to special devices, and other non-translatable aspects
c) Low McCabe complexity scores (less than 20)
d) Visually examine code that has scores of greater than 20 to verify that it really is not too complex to be maintainable. If translated, the complexity will be equivalent in Ada. For a description of the McCabe Cyclomatic Complexity metric see Appendix D.
e) Stand-alone algorithms
f) Distinguish easy from difficult-to-translate pieces.
g) Consider the costs and benefits of separating direct code and executive calls from otherwise translatable code.

3. Reengineer CMS-2 Source Code
a) Where cost-effective, reengineer CMS-2 to separate direct code and executive calls from otherwise translatable code.
b) Convert direct code to CMS-2 high level in preparation for translation. Manually do this for direct code where equivalent CMS-2 is contained in comments. (All direct code and assembler code that is not converted to high level in preparation for translation will require reengineering of the translated Ada source). A currently unfunded prototype tool, the Synetics Assembler Design Extractor, was developed with the goal of translating 80% of direct code to CMS-2. The tool was proven to be immature and not production ready.
c) Reduce the scope of variables based on information provided by DESAN.
d) Remove dead code identified by DESAN.
e) Decide whether to test/ recertify the reengineered CMS-2 system, or to wait until after translation to certify the Ada system.

Translate and Compile
1. Select a translator (APL, CCC, TRADA) based on the profiles provided in Section 3 and translate candidate segments. Data provided in results appendices of this report may help with translator selection.

2. Compile translated code using an Ada 95 compiler (e.g., GNAT).

3. Make changes required to achieve compilation.

4. See Results of Quick Look Inspection, Appendix A, for typical compilation errors expected for each translator.

Reengineer and Improve the Quality of Ada Source Code
1. Reengineer the Translated Ada
Make changes to Ada source code required to achieve correct execution. For a deployed system, recertification is required. See Appendix F, for typical compilation and execution errors to expect with each translator. Improvements in the use of naming conventions, elimination of intermediate variables, use of standard packages, memory management, and performance should be made. See Appendix M for a discussion as applied to the MK2 CMS-2 code sample for translated Ada source and reengineered Ada source.

2. Improve the Quality of Correctly Executing Ada Code

   a) Examine quality of Ada code by using tools like Ada-ASSURED, Logiscope, Adamat, and AdaQuest.

   b) Bring Ada source code into compliance with established programming style guidelines by using a source code formatter and standards enforcer such as Ada-ASSURED.

   c) Manually make other changes so that code conforms to guidelines (e.g., remove GOTOs).


   The RTK is used to increase the quality of Ada code through restructuring. It is available from Rational. It was not used by NRaD. See Table L-2 for a description.

4. Try using Hyperbook to automatically produce documentation from Ada source code.

Hyperbook was not used by NRaD. See Table L-2 for a description.
6. REFERENCES


Fleet Combat Direction Systems Support Activity (FCDSSA) 1993. “Revision Test Plan and Procedures (RTPP) for MTASS”, (U) MT2Y-TPL-SQA-T5524, R06C0.


---

6-1
7. ANNOTATED BIBLIOGRAPHY

TRANSLATING INTO ADA


This document describes the use of the CMS-2 to Ada Transformer to create Ada code from corresponding CMS-2 code. It includes installation instructions, a description of the transformer, a description of the transformation process, an example, and a list of known problems.


This document includes detailed execution procedures for executing the VAX-based TRADA translator, a list of translator generated error messages, the output summary file produced by TRADA, translation strategies, and a sample translation.


This report describes the results of a study to translate approximately 14,000 source lines of code of CMS-2 and direct code from the Advanced Combat Direction System (ACDS) Block 0 program to Ada using the TRADA translator. The purpose of the study was to determine the effort required to perform the translation, to develop a methodology for conducting translations, and to obtain empirical data that would provide a basis for estimating the translation of other similar code.


This paper is a description of TRADA translator. It emphasizes the translation used and the reasons for using them. It describes the CMS-2 dialects and discusses some of the major translation problems.

OTHER REENGINEERING PAPERS


This paper imparts lessons learned on a legacy-replacement project a not straight forward activity. It contains information valuable to the software manager who is considering the re-engineering of a legacy system.

This paper reports on a framework to reverse engineer selected DoD legacy information systems. The approach was developed to recover business rules, domain information, functional requirements, and data architectures, largely in the form of normalized, logical data models. In a pilot study, the authors reverse engineer the data from diverse systems – ranging from home grown languages and database management systems developed during the late 1960’s to those using high order languages and commercial network database management systems.


This paper describes a method called transformation, used to recover abstractions and design decisions made during implementations.


This paper describes a high-level organizational paradigm for development and maintenance, with it, an organization can learn from development and maintenance tasks and then apply that paradigm to several maintenance process models. Associated with the paradigm is a mechanism for setting measurable goals that let you can evaluate the process and product, and learn from experience.


This paper shows how program slicing techniques can be employed to assist in the comprehension of large software systems. It shows traditional slicing techniques at the statement level, and a new technique, interface slicing, at the module level.


This paper discusses technical and nontechnical challenges with migrating and updating legacy software. Challenges range from justifying the expense, to dealing with offshore contractors, to using program-understanding and visualization techniques. The paper provides a summaries of five articles on legacy systems.

This paper describes the steps of the design recovery process, the properties of design recovery, a model-based design recovery system, and the MCC prototype design recovery system called Desire Version 1.0. The system is intended to explore only that aspect of design recovery that does not depend on the domain model. The paper also discusses commercial reverse engineering tools and related research.


This paper describes how developers at Sandia National Laboratories successfully reengineered a 30 year-old system whose source code and documentation was incomplete, into a client-server application.


This paper describes how IBM Federal Systems Division successfully applied its software engineering principles to modify 100,000 lines of 20 year old Federal Aviation Administration air traffic control system code.


This paper presents a conceptual foundation for software re-engineering. The foundation is composed of properties and principles that underlie re-engineering methods, and assumptions about reengineering. A general model of software re-engineering is established that is useful for examining re-engineering issues such as the re-engineering process and re-engineering strategies.


This paper describes a process model of software re-engineering. This model focuses on the breadth of the process by identifying necessary process phases and possible tasks. Variations within the process are discussed.


This paper describes an approach to reverse engineering that first maps the resource exchange among modules and then derives a hierarchical design description using a system-
restructuring algorithm. The focus is on extracting the structural and, to a lesser degree, functional and dynamic properties of large systems — systems composed of modules and subsystems. This process is equivalent to reverse-engineering a system-level design description.


This paper introduces a method that addresses problems associated with reengineering technology based on program analysis methods such as parsing and data flow analysis. An executable domain model is constructed for understanding the context of a program and an object-oriented framework is used to record that understanding.


This paper describes a method to revalidate modified software while minimizing the time and cost involved in maintenance testing by using a systematic automated approach.


This paper describes how you can understand programs by abstracting program functions. This requires you to determine the precise function of a program or program part, which explains exactly what it does to data in all possible circumstances.


This report describes the development of a set of tools designed to convert a program written in CMS-2 into a program written in Ada having the identical functional performance as the original. The core of the tool set is a group of programs that operate on CMS-2 source code and in a series of passes translate to statements or statement blocks, as well as their associated data elements, into a functionally equivalent set of Ada statements and data. In so doing, the syntactic differences in the two languages are resolved, yielding a code structure which is compilable with relatively minor adjustments. The report includes instructions for running the APL translator.

The paper presents examples from protocol studies of expert programmers, illustrating certain common kinds of comprehension errors that can occur in the reading of code during maintenance. These errors involve programming plans which are delocalized – that is, spread far and wide in the text of the program. Strategies are described for preventing comprehension failures due to delocalization.


This paper presents the status of work being done at Grumman on integrating several development concepts into a single life-cycle. This paper defines an extended software development life-cycle that addresses both forward and reverse software development. This is the first and most crucial step in defining a disciplined and repeatable software development process.


This handbook provides guidance for conducting technical and economic assessment of software reengineering strategies to determine whether reengineering legacy software, retire it, redevelop it, or to continue to maintain it as is. The handbook documents a software reengineering cost/benefit methodology that includes a technical process, economic process, and management decision process.


This paper describes how a partially automation of the process of turning a character based user interface into a graphical interface.

Raglund B. and M. Olsem “Maintain Legacy Software or Reengineer?” CrossTalk, vol. 9, no. 4, pp. 6-10.

This article provides a road map that identifies what an organization needs to reengineer a legacy software system. The road map is a 9-step reengineering process. Definitions for reengineering terms is provided.


This paper describes how a prototype system automatically finds all occurrences of a given set of programming structures (cliché) and builds a hierarchical description of the program in terms of the cliché it finds.

This paper describes how to derive a characterization of design decisions based on the analysis of programming constructs. The characterization underlies a framework for documenting and manipulating design information to facilitate maintenance and reuse activities.


This article reviews current software reengineering tools. It describes a new cognitive approach to system reengineering based on code comprehension tools that provides visual representation of code containing less “cognitive noise.” This approach lets programmers better understand the system design. The approach integrates code comprehension tools with current reengineering methodologies to create an integrated reengineering workbench for converting legacy code into newer languages such as Ada or C/C++.


This paper describes a five-step reengineering planning process, starting with an analysis of the legacy system and ending with contract negotiation. The steps are project justification, portfolio analysis, cost estimation, cost-benefit analysis, and contracting.


This book helps the computer professional produce higher quality Ada programs. Guidelines consist of a concise statement of the principles to be followed and rationale for why the guideline is important. These guidelines are probably the most widely accepted and used Ada guidelines.


A book of specific guidelines helping the computer professionals produce higher quality Ada 95 programs.

This paper describes a method of reverse engineering through redocumentation that promises to extend the useful life of large systems.
APPENDIX A : RESULTS OF QUICK LOOK INSPECTION

The purpose of the Quick Look Inspection was to ensure that software products and resources were ready for subsequent phases. During this phase, a CMS-2 sample program of approximately 5000 lines of code was translated by the three translators. Manual modifications were made to the translated code until compilation was achieved. This phase ensures that required computers are accessible, and required software products including translators are installed and execute correctly.

QA9 SELECTED AS SAMPLE

We chose the CMS-2 QA9 program as our sample program. This program is a large self-checking test program designed to verify the MTASS CMS-2 compiler's ability to generate arithmetic code that provides acceptable results when running in an AN/UYK-43 MIL-STD computer. QA9 heavily uses arithmetic capabilities that are critical to every programming language and are generally fairly comparable between languages. QA9 has 5 sections:

- exponentiation
- multiplication
- division
- addition
- subtraction

Since CMS-2 supports legal arithmetic with mixed types, many mixes are checked by the test (for example, fixed-point * floating-point / integer). If the result is within an acceptable range for the computer, a UYK-43 in this case, the test passes.

We selected QA9 because:

- Ada code after translation could be easily mapped back to the original CMS-2.
- The mathematical functionality is common and critical to each language.
- No translation of direct code (embedded assembly) was involved.
- It contained approximately 5000 lines of code.
- We believed we could achieve successful execution after translation.
- A team member was very familiar with QA9.
OVERVIEW OF STEPS

The Quick Look Inspection phase includes the following steps:

1. Compile, link and execute CMS-2 sample

   CMS-2 QA9 with test harness was compiled, linked and executed on a VAX 11/785 computer using MTASS. This step ensured that the CMS-2 code compiled correctly and the chosen sample would execute. Most important, this step established a baseline to verify valid execution of the translated Ada sample.

2. CMS-2 metrics gathering and analysis

   Two CMS-2 analysis tools were executed: CMS-2 Source Code Metrics Generator (METRC) and CMS-2 Source Code Design Analyzer (DESAN). METRC produced SLOC counts, McCabe cyclomatic complexity and Halstead complexity metrics. DESAN produced metrics related to the suitability for translation.

3. Translation to Ada using three translators

   The CMS-2 QA9 sample was input to the three translators to produce translation listings which included the Ada source and the CMS-2 non-translatables. The TRADA and CCCC translators executed on a VAX 11/785, while the APL translator ran on a Sun Sparcstation.

4. Compilation of translated Ada

   The Ada source produced by the TRADA and CCCC translators was compiled using the VAX Ada compiler and GNAT (Sun) compilers. The Ada source code produced by the APL translator was compiled using Sun Ada and GNAT (Sun) compilers. Compilation errors were recorded and the Ada source was reengineered to achieve successful compilation.

5. Examination of compiled Ada source

   Analysis tools were used to examine the compiled Ada source code. These tools included a SLOC counter, Logiscope, and Ada-ASSURED. Ada-ASSURED was used to examine conformance to Software Productivity Consortium Ada quality and style guidelines. Logiscope produced McCabe and Halstead complexity metrics.

   The remainder of this appendix reports these results.

COMPILATION RESULTS

Compilation was attempted on the translator generated Ada QA9 programs. During this phase, the translator developers were given the opportunity to fix translator problems. The APL translator produced one package specification and body. The CCCC translator produced a
monolithic package containing nested packages. TRADA produced multiple package specifications and bodies (Table 3-1 provides translator profiles). All required some modification to compile. Table A-1 to Table A-4 lists the compilation errors for the Ada code generated by the three translators. Only the GNAT compilation errors are presented, the results for the other compilers are very similar. These tables show the compilation errors produced when the original versions of the translators were used before any translator fixes were made. Included in these figures are the program unit, the problem code, explanation of the problem, the manual changes needed to achieve compilation, and any remedies provided by the translator developers to eliminate compilation errors. The right hand column shows how problems were fixed by the developers. If the column contains a “no”, the problem was not fixed at the time of this writing.

APL

Table A-1 and Table A-2 list the compilation errors for the APL-generated Ada QA9 package specification and package body respectively. Later versions of the translator fixed all of the errors in the package specification and all except two in the package body. The syntax errors associated with the package specification included undeclared variables, undefined types, use of Ada reserved words, constraining strings in the parameter list and others. The errors associated with the package body included undefined variables, use of Ada reserved words, and others.

CCCC

The Ada QA9 produced by the CCC translator required manual modifications of the Ada code to compile. The code initially cleanly compiled with the VAX Ada compiler but porting it to the Sun workstations using the Sun Ada and the GNAT compiler produced errors. Table A-3 lists the compilation errors produced in the CCC QA9 Ada body. The table also shows the manual fix made and whether a later version of the translator corrects the problem. No modifications to the specification were required.

TRADA

The Ada QA9 produced by the TRADA translator required manual modifications to compile. The code initially cleanly compiled with the VAX Ada compiler. Later compilation on the Sun workstations using Sun Ada and the GNAT Ada compiler produced some errors. Table A-4 lists the compilation errors produced in the Ada TRADA QA9 specification. No modifications to the body were required.
<table>
<thead>
<tr>
<th>Program Unit</th>
<th>Problem Code</th>
<th>Problem</th>
<th>Manual Reengineering to Compile</th>
<th>Fixed In Later Versions by Translator Developers</th>
</tr>
</thead>
<tbody>
<tr>
<td>global declaration</td>
<td>vawd1 : integers0</td>
<td>integers0 not declared in the basic_defns package</td>
<td>declare integers0 in the basic_defns package</td>
<td>vawd1 : INTEGERS32;</td>
</tr>
<tr>
<td>global declaration</td>
<td>vawd3 : integers0</td>
<td>integers0 not declared in the basic_defns package</td>
<td>declare integers0 in the basic_defns package</td>
<td>vawd3 : INTEGERS32;</td>
</tr>
<tr>
<td>global declaration</td>
<td>fst : fs5_type</td>
<td>fs5_type is undefined</td>
<td>TYPE fs5_type IS (Dsmssd, Inact, Wait, Run, Crash);</td>
<td>type fs5_tv1a_type is (DSMSSD, INACT, WAIT, RUN, CRASH);</td>
</tr>
<tr>
<td>global declaration</td>
<td>type tv8z_ptr is access tv8z_rec</td>
<td>tv8z_rec is undefined</td>
<td>type TV8Z_REC is record null; end record;</td>
<td>type TV8Z_PTR is access WORD_ARRAY(0 .. (8 - 1));</td>
</tr>
<tr>
<td>global declaration</td>
<td>type tv32z_ptr is access tv32z_rec</td>
<td>tv32z_rec is undefined</td>
<td>type Tpv32z_REC is record null; end record;</td>
<td>type TV8Z_PTR is access WORD_ARRAY(0 .. (8 - 1));</td>
</tr>
<tr>
<td>global declaration</td>
<td>type vs2_type is (ALL, NONE)</td>
<td>All is an Ada reserved word</td>
<td>type vs2_type is (z_ALL, NONE)</td>
<td>type vs2_type is (ALL_D, NONE);</td>
</tr>
<tr>
<td>procedure Qthead</td>
<td>(vhead : in STRING(1 .. 60));</td>
<td>constraint not allowed for string.</td>
<td>(vhead : in STRING);</td>
<td>(vhead2 : in STRING);</td>
</tr>
<tr>
<td>procedure Qttext</td>
<td>(vhead2 : in STRING(1 .. 60))</td>
<td>constraint not allowed for string.</td>
<td>(vhead2 : in STRING);</td>
<td>(vhead2 : in STRING);</td>
</tr>
<tr>
<td>Program Unit</td>
<td>Problem Code</td>
<td>Problem</td>
<td>Manual Reengineering to Compile</td>
<td>Fixed In Later Versions by Translator Developers</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>---------</td>
<td>---------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>procedure Qttext0</td>
<td>(vhead2 : in STRING(1 .. 60))</td>
<td>constraint not allowed for string.</td>
<td>(vhead2 : in STRING);</td>
<td>(vhead2 : in STRING);</td>
</tr>
<tr>
<td>procedure Qttests</td>
<td>vmtestno : in STRING(1 .. 4);</td>
<td>constraint not allowed for string.</td>
<td>vmtestno : in STRING;</td>
<td>vmtestno : in STRING;</td>
</tr>
<tr>
<td>procedure Qtisexph</td>
<td>procedure QTISEXPH (vhisex1 : in STRING(1 .. 60); vhisex2 : in STRING(1 .. 60))</td>
<td>constraint not allowed for string.</td>
<td>procedure QTISEXPH (vhisex1 : in STRING; vhisex2 : in STRING);</td>
<td>procedure QTISEXPH (vhisex1 : in STRING; vhisex2 : in STRING);</td>
</tr>
<tr>
<td>procedure Qa9e</td>
<td>for tv16a use at System.&quot;+&quot;(tv8a'address,8)</td>
<td>not used anywhere so comment line out</td>
<td>--OVERLAY--for tv16a use at System.&quot;+&quot;(tv8a'address,8);</td>
<td></td>
</tr>
<tr>
<td>procedure Qa9e</td>
<td>for tv16ovr use at tv16d'address</td>
<td>not used anywhere so comment line out</td>
<td>--OVERLAY--for tv16ovr use at tv16d'address;</td>
<td></td>
</tr>
<tr>
<td>procedure Qa9e</td>
<td>for tv1a use at tha1'address</td>
<td>not used anywhere so comment line out</td>
<td>--OVERLAY--for tv16ovr use at tv16d'address;</td>
<td></td>
</tr>
<tr>
<td>procedure Qa9e</td>
<td>for tv2a use at System.&quot;+&quot;(System.&quot;+&quot;(tv1a'address,512),2)</td>
<td>not used anywhere so comment line out</td>
<td>--OVERLAY--for tv2a use at System.&quot;+&quot;(tv1a'address,512),2);</td>
<td></td>
</tr>
<tr>
<td>Program Unit</td>
<td>Problem Code</td>
<td>Problem</td>
<td>Manual Reengineering to Compile</td>
<td>Fixed in Later Versions by Translator Developers</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------</td>
<td>--------------------------------------------</td>
<td>------------------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>procedure Qa9e</td>
<td>for tv32a use at System.&quot;+&quot;(tv16a'address,16)</td>
<td>not used anywhere so comment line out</td>
<td>--OVERLAY--for tv2a use at System.&quot;+&quot;(tv1a'address, 512),2;</td>
<td></td>
</tr>
<tr>
<td>procedure Qa9e</td>
<td>for tv4a use at System.&quot;+&quot;(tv2a'address,2)</td>
<td>not used anywhere so comment line out</td>
<td>--OVERLAY--for tv2a use at System.&quot;+&quot;(tv1a'address, 512),2;</td>
<td></td>
</tr>
<tr>
<td>procedure Qa9e</td>
<td>for tv64a use at System.&quot;+&quot;(tv32a'address,32)</td>
<td>not used anywhere so comment line out</td>
<td>--OVERLAY--for tv64a use at System.&quot;+&quot;(tv32a'address,32);</td>
<td></td>
</tr>
<tr>
<td>procedure Qa9e</td>
<td>for tv8a use at System.&quot;+&quot;(tv4a'address,4)</td>
<td>not used anywhere so comment line out</td>
<td>--OVERLAY--for tv8a use at System.&quot;+&quot;(tv4a'address,4);</td>
<td></td>
</tr>
<tr>
<td>Program Unit</td>
<td>Problem Code</td>
<td>Problem</td>
<td>Manual Reengineering to Compile</td>
<td>Fixed in Later Versions by Translator Developers</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------</td>
<td>--------------------------</td>
<td>---------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>global declaration</td>
<td>i1 : TAQR_REC</td>
<td>wrong declaration</td>
<td>i1 : integers32</td>
<td>no</td>
</tr>
<tr>
<td>global declaration</td>
<td>i2 : TAQR_REC</td>
<td>wrong declaration</td>
<td>i2 : integers32</td>
<td>no</td>
</tr>
<tr>
<td>procedure Qthead</td>
<td>procedure QTHEAD(vhead : in STRING) is</td>
<td>unconstrained subtype not allowed</td>
<td>vhead_t : STRING (1..60)</td>
<td>procedure QTHEAD(vhead_t : in STRING) is</td>
</tr>
<tr>
<td></td>
<td>vhead_t := STRING;</td>
<td></td>
<td>begin</td>
<td>begin</td>
</tr>
<tr>
<td></td>
<td>begin</td>
<td></td>
<td>vhead_t := vhead ;</td>
<td>vhead_t := vhead_t ;</td>
</tr>
<tr>
<td></td>
<td>vhead_t := &quot; &quot; ;</td>
<td></td>
<td>return ;</td>
<td>vhead_t := vhead_t ;</td>
</tr>
<tr>
<td>procedure Qthead</td>
<td>end QTHEAD ;</td>
<td></td>
<td>end QTHEAD ;</td>
<td>end QTHEAD ;</td>
</tr>
<tr>
<td></td>
<td>procedure Qthead</td>
<td></td>
<td></td>
<td>removed lx1</td>
</tr>
<tr>
<td></td>
<td>lx1 := lx1__x + 1 ;</td>
<td>lx1 is undefined</td>
<td>lx1, lx2, lx3 : INTEGERS16</td>
<td>Removed lx1</td>
</tr>
<tr>
<td>procedure Qthead</td>
<td></td>
<td>should be a string</td>
<td>taqrct(xx__1).hmtn := &quot;00&quot; ;</td>
<td>taqrct(xx__1).hmtn := (others =&gt; ') ;</td>
</tr>
</tbody>
</table>

\*vhead globally declared
Table A-2. APL QA9 Package Body Compilation Error List Using The GNAT Compiler - 2

<table>
<thead>
<tr>
<th>Program Unit</th>
<th>Problem Code</th>
<th>Problem</th>
<th>Manual Reengineering to Compile</th>
<th>Fixed in Later Versions by Translator Developers</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedure QTSPTSW</td>
<td>procedure QTSPTSW (vx2 : in integers32) is begin</td>
<td>the goto target statements in this procedure resides outside of this program unit. The type is also missing from the parameter.</td>
<td>integrate this procedure inside of procedure QTERRE</td>
<td>procedure QTERRE is begin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>case vx2 is</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>when 14 =&gt; goto QTERRF14 ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>when 15 =&gt; goto QTERRF15 ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>when 16 =&gt; goto QTERRE16 ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>when 17 =&gt; goto QTERRE17 ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>when 18 =&gt; goto QTERRE18 ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>when 19 =&gt; goto QTERRE19 ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>... when others =&gt; raise INDEX_OUT_OF_RANGE ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>end QTSPTSW ;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program Unit</td>
<td>Problem Code</td>
<td>Problem</td>
<td>Manual Reengineering to Compile</td>
<td>Fixed in Later Versions by Translator Developers</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------</td>
<td>------------------------------</td>
<td>--------------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>procedure Qttests</td>
<td>taqrct(xx_1).htmu := 0 ;</td>
<td>should be a string</td>
<td>taqrct(xx_1).htmu := &quot;00&quot;;</td>
<td>taqrct(xx_1).htmn := (others =&gt; ' ') ;</td>
</tr>
<tr>
<td>procedure Qtsynops</td>
<td>28 ( 5 ) ( vhsynhed ) :=</td>
<td>Incorrect translation</td>
<td>vhsynhed(29..32) := vmtstno ;</td>
<td>vhsynhed(29..32) := vmtstno ;</td>
</tr>
<tr>
<td></td>
<td>vmtstno ;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>procedure Qtsynops</td>
<td>&lt;&lt;LOOP&gt;&gt;</td>
<td>Ada reserved word</td>
<td>&lt;&lt;first_LOOP&gt;&gt;</td>
<td>&lt;&lt;LOOP_D&gt;&gt;</td>
</tr>
<tr>
<td>procedure Start</td>
<td>vhtmu := &quot; &quot; ;</td>
<td>wrong number of spaces</td>
<td>vhtmu := &quot; &quot; ;</td>
<td>vhtmu := &quot; &quot; ;</td>
</tr>
<tr>
<td>procedure Qa9a.</td>
<td>z := x ** y ;</td>
<td>y is defined as float. Ada can not handle floating exponents</td>
<td>typecast the floating exponent to integer.</td>
<td>Added the following function in the mathpac package.</td>
</tr>
<tr>
<td>Total of 84</td>
<td></td>
<td></td>
<td>z := x ** integer(y)</td>
<td>function &quot;**&quot;</td>
</tr>
<tr>
<td>occurrences of this error</td>
<td></td>
<td></td>
<td></td>
<td>(X : in INTEGER; Y : in Float)</td>
</tr>
<tr>
<td>procedure Qa9a.</td>
<td>QTISEXP ( INTEGER(vaws90.00.0), 8#33000#)</td>
<td>vaws90.00.0 not valid</td>
<td>QTISEXP ( INTEGER(vaws9), 8#33000#)</td>
<td>QTISEXP ( Float_To_Integer( vaws9 * (2**9) ) , 8#33000# ) ;</td>
</tr>
<tr>
<td>Total of 49</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>occurrences of this error.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program Unit</td>
<td>Problem Code</td>
<td>Problem</td>
<td>Manual Reengineer to Compile</td>
<td>Fixed in Later Versions by Translator Developers</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------------------------------------</td>
<td>-------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>procedure Qtisexph</td>
<td>procedure QTISEXPH (vhisex1 : in STRING; vhisex2 : in STRING) is vhisex1_t : STRING; vhisex2_t : STRING; begin vhisex1_t := vhisex1; vhisex2_t := vhisex2; vhisex1_t := vhospace; vhisex2_t := vhospace; return; end QTISEXPH;</td>
<td>unconstrained subtype not allowed</td>
<td>vhisex1_t : STRING (1..60) := (1..60 =&gt; '');</td>
<td>procedure QTISEXPH(vhisex1_t : in STRING; vhisex2_t : in STRING) is begin vhisex1* := vhisex1_t; vhisex2 := vhisex2_t; vhisex1* := vhospace &amp; c2a_blanks(1..20); vhisex2* := vhospace &amp; c2a_blanks(1..20); return; end QTISEXPH;</td>
</tr>
</tbody>
</table>

* vhisex1 and vhisex2 declared globally
<table>
<thead>
<tr>
<th>Program Unit</th>
<th>Problem Code</th>
<th>Problem</th>
<th>Manual Reengineer to Compile</th>
<th>Fixed in Later Versions by Translator Developers</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedure Qtiexph</td>
<td>procedure QTISEXPH(vhisex1 : in STRING; vhisex2 : in STRING) is vhisex1_t : STRING; vhisex2_t : STRING; begin vhisex1_t := vhisex1; vhisex2_t := vhisex2; vhisex1_t := vhspace; vhisex2_t := vhspace; return; end QTISEXPH;</td>
<td>unconstrained subtype not allowed</td>
<td>vhisex2_t : STRING(1..60) := (1..60 =&gt; '');</td>
<td>procedure QTISEXPH(vhisex1_t : in STRING; vhisex2_t : in STRING) is begin vhisex1* := vhisex1_t; vhisex2* := vhisex2_t; vhisex1* := vhspace &amp; c2a_blanks(1..20); vhisex2* := vhspace &amp; c2a_blanks(1..20); return; end QTISEXPH;</td>
</tr>
<tr>
<td>procedure Qa9b</td>
<td>if vaws9 = -240 then</td>
<td>vaws9 is defined as float</td>
<td>if vaws9 = -240.0</td>
<td>if vaws9 = -240.0 then</td>
</tr>
<tr>
<td>procedure Qa9c</td>
<td>if vaws9 = -7 then</td>
<td>vaws9 is defined as float</td>
<td>if vaws9 = -7.0</td>
<td>if vaws9 = -7.0</td>
</tr>
<tr>
<td>procedure Qa9e</td>
<td>if vfs6 = -17388 then</td>
<td>vfs6 is defined as float</td>
<td>if vfs6 = -17388.0</td>
<td>if vfs6 = -17388.0</td>
</tr>
</tbody>
</table>

* vhisex1 and vhisex2 declared globally
<table>
<thead>
<tr>
<th>Program Unit</th>
<th>Problem Code</th>
<th>Problem</th>
<th>Manual Reengineering to Compile</th>
<th>Fixed in Later Versions by Translator Developers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nquack_Qa9</td>
<td>with Math_Lib_Cms2;</td>
<td>Math_Lib_Cms2 depends on package Math which is a VAX math library.</td>
<td>with Double_Elementary_Functions;</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>use Math_Lib_Cms2;</td>
<td>Math_Lib_Cms2 depends on package Math which is a VAX math library.</td>
<td>use Double_Elementary_Functions;</td>
<td>no</td>
</tr>
<tr>
<td>Program Unit</td>
<td>Problem Code</td>
<td>Problem</td>
<td>Manual Reengineer to Compile</td>
<td>Fixed in Later Versions by Translator Developers</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>---------------</td>
<td>------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>package Cms_2_Types</td>
<td>TYPE Float_s IS DIGITS 7 RANGE -8#0.7777777777806* 2.0 ** 1023.. 8#0.7777777777806* 2.0 ** 1023;</td>
<td>number too big.</td>
<td>TYPE Float_ss IS DIGITS 7; TYPE Float_S is DIGITS 7 RANGE -8#0.7777777777806* 2.0 ** Float_ss'Safe_Emax .. 8#0.7777777777806* 2.0 ** Float_ss'Safe_Emax;</td>
<td>no</td>
</tr>
<tr>
<td>package Cms_2_Types</td>
<td>TYPE Float_d IS DIGITS 16 RANGE -8#0.7777777777777777777806* 2.0 ** 1023.. 8#0.7777777777777777777806* 2.0 ** 1023;</td>
<td>number too big</td>
<td>TYPE Float_d IS DIGITS System.Max_Digits</td>
<td>no</td>
</tr>
</tbody>
</table>
SOURCE LINES OF CODE COMPARISONS

Figure A-1 shows the source lines of code (SLOC) for the translator generated Ada QA9s and CMS-2 QA9 programs. Ada SLOC was counted immediately following translation. The first three sets of bars (left to right) in the graph represent the translated Ada code produced by the TRADA, APL, and CCCC translators, without the predefined utilities that each of the translators provide. The right three sets of bars represent the corresponding code for the entire program.

CMS-2 line counts for the CMS-2 SLOC is the total number of executable statements ending in “;”. Comment lines are statements beginning with the word “comment”. Text counts are total lines as counted by a text editor.

Ada line counts for the SLOC for the Ada source code is computed as the number of statements ending with a “;”, except those occurring in comments and character strings. Comment lines were counted as lines that contain two successive hyphens not embedded in a character string. Text count again are total lines as counted by a text editor.

We do not believe that any meaningful conclusions can be drawn from the SLOC metrics in and of themselves. (See Appendix D for a discussion on problems using SLOC as a metric). However, figures for executable statements support our conclusion that all translators implement a transliterate approach (Appendix C).

HALSTEAD METRICS

Halstead metrics are shown in Figure A-2. The graph shows the overall program length, the vocabulary size, and the actual volume for six program units produced by the translators. These units represent the majority of the QA9 code. As seen from the graph, the translator outputs mirror each other and the CMS-2 code. In other words, the translators produce Ada code that closely resembles the CMS-2 code. QTCON1 vocabulary is very low for TRADA because TRADA moved the complex vocabulary to another subprogram (QTMESSW).

MCCABE CYCLOMATIC COMPLEXITY METRIC

The McCabe cyclomatic complexity metric for the QA9 procedures is shown in Figure A-3. The McCabe cyclomatic complexity metric is based on a graph theoretic interpretation of program control flow and provides an indication of structural complexity. More explanation of this metric is discussed in Appendix D.

As seen by the graph, the translated source code mirrors the CMS-2 code for most of the program units. In units QTCON1 (Figure A-3. McCabe Cyclomatic Complexity Metric - 1), QTSYNOPS and QA9A (Figure A-3. McCabe Cyclomatic Complexity Metric - 3) the CMS-2 code is considerable more complex than the Ada code because the CMS-2 code uses constructs that are considered more complex.

In this table, note that the Ada code for QTCON1 appears to have significantly less complexity than the original CMS-2. This occurs because QTCON1 contains a procedure switch (P-SWITCH) which was translated to an Ada case statement whose complexity is shown under QTMESSW.

---

1 The source listing for the Ada SLOC counter is given in Appendix J.
(Note that only 3 bars are present for QTMESSW) For example, the QTCON1 CMS-2 has a McCabe metric of 13. TRADA resultant Ada has a McCabe of 10 (7 for QTMESSW plus 3 for QTCON1).

Figure A-4 represents the complexity versus the percent of the QA9 source code produced by the three translators. This figure shows that most of QA9 produced by the three translators is very complex. See Appendix D for a detailed explanation of the cyclomatic complexity (V(G)). As seen from the graph, the translator outputs mirror each other with only about eight percent of the code having a V(G) less than 10, about 65 percent of the code having a V(G) between 61 and 70, and about 25 percent of the code having a V(G) over 90. Keep in mind that V(G) greater than 50 usually means the source code is incomprehensible. These results are another indication of the translators producing Ada code that resembles the CMS-2 code.
Figure A-1. QA9 CMS-2 and Translated Ada QA9 Line Counts

1. Ada SLOC is number of delimiting semicolon statements.
   CMS-2 SLOC is number of delimiting $ statements.
2. Text is lines of code counted by an editor (includes comments, blank lines and text).
Figure A-3. McCabe Cyclomatic Complexity Metric - 1

*Note the CMS-2 complexity in QTCON1 was translated into the Ada QTMESSW procedures.
Figure A-3. McCabe Cyclomatic Complexity Metric - 2

* Note that V(G) appears to be dramatically greater for QTSKIP, QTTESTS, and QTISEXPB procedures than other procedures. The differences are not significant since the magnitude of the V(G) scale only ranges from 0 to 3.
Figure A-3. McCabe Cyclomatic Complexity Metric - 3

* Note that the CMS-2 QTSYOPS has a high V(G) because it makes a call to a P-SWITCH. This was translated into a case statement in the Ada QTCONSW procedures. The CMS-2 QA9A and CMS-2 QTSET have a higher V(G) because of a call to a P-SWITCH.
Figure A-4. McCabe Complexity versus Percent of Ada QA9
CONFORMANCE TO SOFTWARE PRODUCTIVITY CONSORTIUM GUIDELINES

The reworked Ada QA9 code produced by the translators was analyzed for conformance to the Software Productivity Consortium (SPC) Ada coding guidelines. The SPC presents a set of specific guidelines for using the features of Ada in a disciplined way intending to produce high quality Ada programs. These guidelines are the most widely accepted Ada guidelines that exist today. Conformance was analyzed by processing the Ada code with the standards enforcement editor of Ada-ASSURED. Ada-ASSURED is a language-sensitive editor for Ada that supports the enforcement of quality and style guidelines and can be set to enforce those guidelines developed by the SPC. All three translators produced Ada code that mirrored the CMS-2 code. Therefore, poor quality CMS-2 code will be translated into poor quality Ada code. Because the CMS-2 QA9 sample violated SPC guidelines, the corresponding Ada code also violated these guidelines. All three translators produced code that had similar coding violations. These included:

- Use of GOTOs
- Non-constant object declarations declared in the visible part of the package specification
- Use of Labels (associated with GOTOs)
- Use of unnamed nested loops
- Subprogram body size exceeds maximum of 200 SLOC

Table A-5 shows the total number of SPC coding violations for Ada QA9 produced by the three translators. These violations were detected by the tool Ada-ASSURED.

Table A-6, Table A-7, and Table A-8 provide detailed information on the coding violations flagged by Ada-ASSURED for Ada QA9 code produced by the APL, CCCC, and TRADA translators.

These tables identify the Ada program unit where the violation occurred, show the problem code (where appropriate) and provide the violation as reported by Ada-ASSURED. When the problem code is many statements long, it is not included in the table. Instead, a brief explanation may be provided in the problem code column.
<table>
<thead>
<tr>
<th>Translator</th>
<th>Use Clause</th>
<th>Named Association</th>
<th>Use of Gotos</th>
<th>Use of Labels</th>
<th>Nested Loops Must be Named</th>
<th>Exit Statements from named loops must be named</th>
<th>Blocks must be named</th>
<th>Non-constant object declarations not allowed in the visible part of the spec</th>
<th>Sub-program body size exceeds 200</th>
<th>Long loops must be named</th>
</tr>
</thead>
<tbody>
<tr>
<td>APL</td>
<td>2</td>
<td>2</td>
<td>403</td>
<td>371</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>101</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>CCCC</td>
<td>2</td>
<td>0</td>
<td>403</td>
<td>394</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>1202&lt;sup&gt;1&lt;/sup&gt;</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>TRADA</td>
<td>0</td>
<td>0</td>
<td>403</td>
<td>391</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>319</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

<sup>1</sup> CCCC produced many objects that are unused in the program. According to the SPC guidelines the use of non-constant object declarations in the package specification should be avoided.
Table A-6. Details on SPC Ada Style Violations: Ada QA9 Produced by APL

<table>
<thead>
<tr>
<th>Program Unit</th>
<th>Problem Code</th>
<th>Coding Violations as Reported by Ada-ASSURED</th>
</tr>
</thead>
<tbody>
<tr>
<td>package spec</td>
<td>use System</td>
<td>The identifier: System is used in context &quot;use clause&quot;</td>
</tr>
<tr>
<td>Qa9qlook</td>
<td>Sx1 : Integeru32</td>
<td>non constant object declarations are not permitted in the visible part of a package specification¹</td>
</tr>
<tr>
<td>package body</td>
<td>use System;</td>
<td>The identifier: System is used in context &quot;use clause&quot;</td>
</tr>
<tr>
<td>Qa9qlook</td>
<td>goto Qterre14</td>
<td>Use of GOTO is not allowed.¹</td>
</tr>
<tr>
<td></td>
<td>&lt;&lt;Qterre14&gt;&gt;</td>
<td>Labels are not allowed since GOTO is not allowed.¹</td>
</tr>
<tr>
<td></td>
<td>(multiple nested un-named loops)</td>
<td>Nested loops must ALL be named.</td>
</tr>
<tr>
<td></td>
<td>(Too many statements within loop)</td>
<td>A loop this long must be named.</td>
</tr>
<tr>
<td>procedure Qa9a</td>
<td>–</td>
<td>Subprogram body size of 885 exceeds maximum of 200</td>
</tr>
<tr>
<td>procedure Qa9b</td>
<td>–</td>
<td>Subprogram body size of 551 exceeds maximum of 200</td>
</tr>
<tr>
<td>procedure Qa9c</td>
<td>–</td>
<td>Subprogram body size of 551 exceeds maximum of 200</td>
</tr>
<tr>
<td>procedure Qa9d</td>
<td>–</td>
<td>Subprogram body size of 551 exceeds maximum of 200</td>
</tr>
<tr>
<td>procedure Qa9e</td>
<td>–</td>
<td>Subprogram body size of 550 exceeds maximum of 200</td>
</tr>
</tbody>
</table>

¹ Occurs many times
<table>
<thead>
<tr>
<th>Problem Code</th>
<th>use System</th>
<th>non-constant object declarations are not permitted in the visible part of a package specification.</th>
<th>The identifier: System is used in context &quot;use clause&quot;</th>
<th>The identifier: System is used in context &quot;use clause&quot;</th>
<th>statement nesting depth of 18 exceeds maximum of</th>
<th>use goto OR use case OR use null statement OR use procedure OR use function OR use interface OR use package OR use subprogram OR use type OR use variable OR use item</th>
<th>labels are not allowed since GOTO is not allowed.</th>
<th>all BLOCKS must be named.</th>
<th>nested loops must ALL be named.</th>
<th>Subprogram body size of 659 exceeds maximum of 200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Unit</td>
<td>use System;</td>
<td>sX1 : integer := 1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>package spec</td>
<td>package body</td>
<td>package spec</td>
<td>package body</td>
<td>procedure Qa9</td>
<td>procedure Qa9</td>
<td>procedure Qa9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 Occurs many times
<table>
<thead>
<tr>
<th>Program Unit</th>
<th>Problem Code</th>
<th>Coding Violations as Reported by Ada-ASSURED</th>
</tr>
</thead>
<tbody>
<tr>
<td>package spec</td>
<td>Vhisex1 : H_60 := (others =&gt; Ascii.Nul);</td>
<td>Non-constant object declarations are not permitted in the visible part of a package specification.¹</td>
</tr>
<tr>
<td>Aqtcon</td>
<td>goto Qterre14</td>
<td>Use of GOTO is not allowed.¹</td>
</tr>
<tr>
<td></td>
<td>&lt;&lt;Qterre14&gt;&gt;</td>
<td>Labels are not allowed since GOTO is not allowed.¹</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nested loops must ALL be named.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A loop this long must be named.</td>
</tr>
<tr>
<td>procedure Qa9a</td>
<td></td>
<td>Subprogram body size of 883 exceeds maximum of 200</td>
</tr>
<tr>
<td>procedure Qa9b</td>
<td></td>
<td>Subprogram body size of 556 exceeds maximum of 200</td>
</tr>
<tr>
<td>procedure Qa9c is</td>
<td></td>
<td>Subprogram body size of 562 exceeds maximum of 200</td>
</tr>
<tr>
<td>procedure Qa9d is</td>
<td></td>
<td>Subprogram body size of 563 exceeds maximum of 200</td>
</tr>
<tr>
<td>procedure Qa9e is</td>
<td></td>
<td>Subprogram body size of 562 exceeds maximum of 200</td>
</tr>
</tbody>
</table>

¹ Occurs many times
CONCLUSIONS

1. The complexity of the Ada code produced by the translators mirrors the complexity of the CMS-2 code. This is shown with the McCabe and Halstead metrics. The translators do not introduce complexity.

2. The complexity of the Ada code by the translators is similar. Complexity is the same across translators. This is shown with the McCabe and Halstead metrics.

3. The Ada produced by the translators all needed some reengineering to compile cleanly. APL fixed a number of bugs that simplified the reengineering of the APL produced Ada code.

4. The translators all produced Ada source that needs to be made compliant with SPC guidelines. The translators have similar problems whose origins are in the CMS-2 code.

5. The variable names produced by the translators usually matched the CMS-2 names. This was extremely useful in comparing the CMS-2 code with the translated Ada code. These names could later be converted to meaningful names during the reengineering process.

6. All translators produced indented Ada source code.

7. The sample selected CMS-2 QA9 was well suited for translators.
APPENDIX B : RESULTS OF STRESS TESTING

The purpose of stress testing is to examine the performance of the APL, CCCC, and TRADA translators when faced with a spectrum of CMS-2 language constructs as seen in today's CMS-2 programs. This phase thoroughly tested the ability of translators to handle all CMS-2 constructs.

TEST CASES

Test cases used for stress testing were:

- The Machine Transferable Support Software (MTASS) CMS-2 Test Suite
- CMS-2 code from NAVAIR, NAVSEA, and SPAWAR projects

The MTASS test suite was specifically designed to test CMS-2 compilers. This collection of CMS-2 test files, containing CMS-2 programs, evolved over a period of 20 years. These files were designed to be more “harmful” than normal because they test variable extremes and compiler weak spots (e.g., rules of arithmetic) largely discovered by user reported errors. A comprehensive list of CMS-2 test files is found in the Machine Transferable Support Software (MTASS) Revision Test Plan Procedures (RTPP) document (FCDSSA, 1993). Those selected for stress testing are shown in Table B-1, and Table B-2. Not all CMS-2 constructs have an associated test file(s). However, where test file(s) existed for a CMS-2 construct, one was selected as a translation candidate. This resulted in a total of 84 files being chosen from the AN/UYK-7 functional Quality Assurance (QA) test suite for translation.

These QA files represented at least one functional test for every translatable CMS-2 construct (e.g., numeric expression) where a test file(s) existed. Sometimes non-translatable constructs (e.g., overlays) were input to examine translator behavior. Several of these files contain forced expected errors. These tests are very appropriate for testing legacy programs because they typically contain non-translatables and other errors.

The CMS-2 source code contributed by NAVAIR, NAVSEA, and SPAWAR included the Extra Low Frequency (ELF) Communications, MK-2 Fire Control System, AEGIS AN/UYK-43 SPYLOOP, S3-Aircraft Tactical Mission Program (TMP), and H60B Helicopter projects. Points-of-contact for these projects are given in Section 2. Results of the stress testing appear in the Table B-3, Translating and Compiling Using Project Contributed Legacy CMS-2 Source Code.

We also selected QA9 from the AN/UYK-43 test suite for testing during this phase as well as in the Quick Look and Reengineer to Execution phases. QA9 performs the most comprehensive numeric testing. QA9 does self-checking (vice manual checking) to compare CMS-2 execution results with expected results.

MTASS STRESS TESTING

Each CMS-2 test file was originally designed to be compiled with a compool (pre-compiled common system data) then linked with a Test Controller (TC). For translation purposes, the compool and test controller, both in source code form, were included directly in the translation run.
stream using the INCLUDE directive. TC CMS-2 code for executive input/output requests, producing test results for self-checking QA files, was strategically commented out. These services were not applicable to stress testing, and would be provided as needed for execution testing in the Ada modified TC via Text_IO, Integer_IO, Float_IO, and other IO packages from the Ada Predefines.

CCCC and TRADA were stress tested on an NRaD VAX 11/785 computer running the VMS 5.5-1 operating system. This was a very lightly loaded system with only this testing and system operator active. The process was automated using command files to submit all 84 test files, 5 to 20 at a time, to all three translators as batch jobs. Grouping was used because translation can be sufficiently time consuming to time-out batch queues. Queues ran sequentially vice concurrently allowing wall clock time collection with little interference from any other jobs. APL was stress tested in a similar manner on a lightly loaded Sun SPARC 10 running OS 4.1.3.

Translation catastrophic failure includes abortive failures such as core dumps and symbolic stack dumps (tracebacks from constraint errors), infinite loops, and cases where all appeared well but no Ada was generated. Several catastrophic failures occurred while running each translator. The overall stress testing translation results, including CMS-2 constructs causing failures, are reported in Table B-1.

Stress testing included the compilation of all translator produced files. (If any code was marked/bypassed during translation, functionality would be lost and correct execution would not be possible, but the remainder needed to compile correctly). The volume of generated Ada provided the perfect opportunity to try many compiles. Overall stress testing compilation results are reported in Table B-2, the Stress Test Using MTASS Test Suite - Compile Information, included in this section.

CONCEPTUAL DIFFERENCES AMONG TRANSLATORS

Five conceptual differences surfaced among translators for:

1. controlling the translation process,
2. termination from translation and placement of errors,
3. construction of packages,
4. providing a utility package that contains type and function declarations, and
5. organizing the translators’ generated Ada code into files.

Each will be discussed.

Controlling The Translation Process

APL provides switches, TRADA provides a script file, and CCCC provides no control over the resultant Ada code. Control over the format and content, such as upper-lower case and indenting of the Ada code is desirable.
Termination and Reporting Errors

CCCC and APL report some classes of errors interactively during translation, place other classes of errors into the generated Ada code inside comments, and always attempt to complete the translation process regardless of errors. TRADA places some classes of errors into its summary file, some classes of errors into the generated Ada code inside of comments, and depending on the real or perceived errors will quit translation as opposed to generating bad Ada. TRADA generated Ada for only 54 of the 84 QA files which is shown at the end of Table B-2.

Construction of Packages

APL produces one package specification and one package body per translation. CCCC and TRADA produce multiple specifications and bodies.

Providing Utility Package

TRADA generates all required Ada from its CMS-2 input, but both APL and CCCC, as part of the translator installation, provide canned Ada packages called BASIC_ DEFNs and PREDEFINEDs which contain some commonly used types and functions. This eliminates the requirement for APL and CCCC translators to generate these. Since their generated Ada might use these types and functions, the predefines must be initially compiled into an Ada library before any other APL or CCCC generated code is compiled.

Creating Files

CCCC puts all generated Ada into one big file, APL puts all Ada into one specification and one body file, and TRADA generates multiple files to accommodate multiple package specifications and bodies, and provides a compilation order in a summary file. TRADA's results were deemed to accommodate changes most easily, and be more amenable to library based configuration management.

BENEFITS OF STRESS TESTING

Stress Testing was of mutual benefit to translator developers and ONR/NRaD. When a catastrophic failure occurred the developer was given supporting CMS-2 source to reproduce and correct the problem. Stress tests provided QA for the developers who, in turn, resubmitted their enhanced products for evaluation. After delivery of a corrected translator, all 84 QA files were input from the beginning to locate failures (regressions) of tests that previously passed. Translator corrections benefited ONR/NRaD, and any future user, by improving a translator's probability of completing its Ada generation, and generating better code in some cases. Results shown in all stress test tables, Tables B-1 through B-3 are based on the final corrected translator revision provided by the developer.
EVALUATION OF TRANSLATION RESULTS

Refer to Table B-1, Stress Testing Using MTASS Test Suite - Translation Information.

The columns titled Test Description, User Handbook Section, and File Name are self-explanatory, e.g. Name (2nd page, 2nd row of table) is defined in MTASS CMS-2 User Handbook section 3.2.4, and tested in file 070QA541. Some files such as 070QA2 test multiple constructs (numeric expression, boolean expression, and others), and appear several places in the checklist. N/A means a specific file is not available to test the construct, but the construct is probably tested non-specifically in other tests. For example, User Handbook section 3.2.1 delimiters are tested throughout the tests. The Test Controller is not in the CMS-2 User Handbook and is included in the table only to provide Source Lines of Code (SLOC) information for later use. (SYSDD and QTCON were INCLUDED in each of the 84 QA files, except for 070DC1 and 070DCER1 which are standalone direct code tests for the translation process.)

Test Type indicates when the CMS-2 construct's file was (M)anual checking, automated and (S)elf-checking, contained (B)oth manual and self-checking parts, was tested (N)on-specifically in other tests, or not tested (-).

Translator Pass, Quit, or Fail and minutes of wall clock time shows all 3 translators' results. When a translator Passed, Ada code was generated followed by normal termination. When a translator Quit, some real or perceived unsatisfactory condition caused a user message(s), no Ada was generated, but termination was normal. When a translator Failed it caused a core dump, traceback, looped infinitely, or quietly generated no Ada. When a translator had a catastrophic failure, the CMS-2 code causing the failure was provided to the developer for translator correction and resubmission to stress testing. A history of failures and corrections can be seen in a sequence such as P,F,F,P which indicates that the translation originally passed, translator changes caused a regressive failure, and, finally, the regression in the translator was corrected. (CMS-2 code in QA files was never modified to correct translator failures.) The total numbers of unique catastrophic failures for all 84 QA tests are shown for each translator on page 14 of this table. The unique failures were: TRADA-6, APL-11, CCC-10. No trends were apparent for CMS-2 constructs causing failures across translators. Note that the unique failures are not a summation of the columns since some files appear several times throughout the table.

The wall clock translation time depended on test file size, CMS-2 constructs encountered, a translator's design/implementation, and host computer. We were the only user on the host computers during the calculation of wall clock time. TRADA and CCC ran on a dedicated VAX/VMS so some comparison between these two is reasonable. APL ran on a dedicated Sun/OS which is faster than the VAX/VMS so time comparison with the other two translators is not reasonable. In most cases where TRADA finished in one minute, it had reported syntactic or semantic problems (real or perceived) needing correction, and then quit.

TRADA generated Ada for 54 of the 84 QA files, APL for all 84, and CCC for 83 of the 84 files. Total translation times for all 84 QA tests are shown on page 14 of the table. The total times were: TRADA - 6 hr. 22 min., APL - 4 hr. 42 min., CCC - 31 hr. 59 min. Based on a 54/84 ratio and adjusting for the 1 minute already spent, we estimate that TRADA could have completed all 84 tests, if they had been in an acceptable condition, in about 9 hr. 30 min. Note that the total times are not a column summation since some files appear several times throughout the table.
We do not believe times, nor time differences between translators, are significant since translators are not used like a compiler which is run repeatedly during project life cycle. Translation will probably involve only a few iterations of reengineering/translation and then be finished.

CMS-2 Source Lines of Code (SLOC) shows the SLOC present in each QA file (before the Test Controller has been INCLUDED for translation). Throughout stress testing, CMS-2 and Ada SLOC is counted as straight lines of text as counted by an editor. A text editor provided these numbers confirmed by the CMS-2 Metrics Generator. For example, the Name test 3.2.4 file 070QA541 is 656 unique SLOC. Table B-1, Stress Testing Using MTASS Test Suite - Translation Information shows only QA file SLOC without the test controller.

Table B-1, shows the combined TC (1543 SLOC) and QA file’s SLOC which in this case would be 1543 plus 656 for 070QA541 totaling 2199 SLOC actually input to a translator.

About 117,700 totally unique SLOC, as shown in the Table B-1, Stress Testing Using MTASS Test Suite - Translation Information page 14, were input to each translator. This sums all 84 QA files, and adds the compool and Test Controller (TC) only once. However, the compool and test controller were INCLUDED in all but two files which means about 242,600 total CMS-2 SLOC were input to each translator, as is shown in the compile information table, Table B-2, Stress Testing Using MTASS Test Suite - Compile Information totals. Considering that data and procedures in TC are used in different contexts by every QA file, each translator processed 242,600 lines of source code. Note that total unique SLOC is not a column summation since some files appear several times throughout the table.

EXAMINATION OF COMPILATION RESULTS

Table B-2 shows results after attempting to compile code generated by each translator for each QA file with three different Ada compilers – VAX, Sun, and GNAT. This required nine compile attempts per CMS-2 QA file.

The columns titled Test Description, User Handbook Section, and File Name are the same as described previously for Table B-1.

Test Number is included in this table only as a cross reference into the stress testing command files. Test number represents the command file alpha/numeric order. The command files (COM) were built in QA test alpha/numeric order, (i.e. QA10, QA11A, QA11B), rather than in CMS-2 User Handbook section numeric order. In User Handbook order a QA test could appear several times. COM file alpha/numeric order ensured each file was invoked once, and only once.

Compiles VAX/Sun/GNAT/ and Ada Source Lines of Code (SLOC) shows compilation results from the three compilers for each translator for each QA file. Results show (C)orrect compile, (U)nsuccessful compile, or X when no Ada was generated by a translator, therefore, no compile attempt was possible. An unsuccessful compile is one containing error messages or informational messages stating that a constraint error will be raised during execution. (3% of errors were informational constraint error messages.) For correct compilation remember that all direct code, non-translatables, and constructs that a translator could not handle appear in comments in a translator’s generated Ada. Therefore, a correct compilation does not give an accurate indication of future correct execution. Unsuccessful compilation implies one or more compilation errors were
encountered across a very wide syntax (format) and semantic (meaning) spectrum. The number following the last slash / is the Ada SLOC generated by the translator, or the word none. The word none, will be preceded by X/X/X/ in all cases. This table allows comparison of the QA test including Test Controller CMS-2 SLOC to the Ada SLOC generated by each translator. For example, the last test in the table, 070QA539D (Table B-2, page 13), shows 2410 CMS-2 SLOC (1543 Test Controller plus 867 for QA539D itself) resulted in 5002 TRADA SLOC, 4414 APL SLOC, and 10252 CCCC SLOC. Remember that both the CMS-2 and Ada SLOCs were counted by editors and include comments and 'white space' (blank lines). Only two tests of the 84, 070DC1 and 070DCER1, did not use/ include TC. Therefore, CMS-2 SLOC numbers for these 2 files are the same in both the translation and compile tables; 4431 and 274 respectively.

EXAMINATION OF SLOC IN COMPILE INFORMATION TABLE

Table B-2 contains the TOTAL SLOC on page 14. 242.6K total CMS-2 SLOC resulted in 385.0K TRADA SLOC (ignore the second numbers for now), 468.3K APL SLOC, and 923.7K CCCC SLOC. Based on the ratio that TRADA generated Ada for only 54 of the 84 files, we estimate that TRADA would have, had all the QA files been acceptable to TRADA, generated the second number of about 598.9K SLOC for all 84 files. The second numbers for APL, 468.9K, and CCCC, 925.7K, simply add 1 time their BASIC_DEFNS and PREDEFINEDs SLOCs, respectively, considering them as part of their overall Ada. This addition is insignificant in both cases.

The Ada SLOCs can be used as a basic indicator of code expansion from the CMS-2, and a comparator among translators. Total SLOCs show that a project may experience an Ada to CMS-2 expansion ratio as high as 4:1 after translating non-reengineered CMS-2. This depends on the translator selected and the CMS-2 constructs. One must consider that the Ada file(s) also contain blank lines for readability (white space), and may contain non-translatables bracketed in comments and error messages. White space is about: TRADA - 10%, APL - 6%, and CCCC - 4%. The original CMS-2 white space was about 3%. Ada reengineering of the non-translatables may result in a size decrease. Removal of error message lines will decrease SLOC. Some error message bloating can be expected in APL and CCCC since most of their error messages appear as Ada comments, whereas, TRADA places many error messages in its summary file. Considering all the above, a project's Ada to CMS-2 expansion ratio will likely be around 2:1. Reengineering the Ada can significantly reduce this ratio. Comparing Ada SLOCs across translators, either by QA file or by totals, shows the code each translator perceived as necessary to solve the problem. Note that total SLOC numbers are not a column summation since some files appear several times throughout the table.

EXPLANATION OF ADA COMPILATIONS

Now continue referring to page 14 of Table B-2. These results are based on QA file translations produced by final translator revisions. Correct compilation percentages are shown for each translator for VAX, Sun, and GNAT compilers, and are discussed in the following three paragraphs. Using multiple compilers showed that when a translator's generated Ada compiled with one compiler, it was over 90% probable to compile correctly, with very minor adjustments, with the
other two compilers. These minor adjustments are mentioned in the next three paragraphs, and are discussed in detail in the **Reengineer Until Ada Executes Correctly** report section, Appendix C.

For TRADA, 24 of the 84 QA files correctly compiled with VAX Ada yielding 29%. But TRADA quit processing for 30 files, producing Ada for only 54 files. The second number, inside parentheses, indicates that 44% of the 54 files produced by TRADA compiled with VAX Ada (24/54). Initially, none of TRADA’s 54 files compiled with either Sun or GNAT. Investigation showed the range defined for floating point single and floating point double types was acceptable to VAX Ada but not by Sun or GNAT compilers on the Sun SPARC. Changing the range values to predefined language attributes of Safe_Emax and Max_Digits provided a workaround for a problem which had guaranteed 100% failure with Sun and GNAT. We believe that this change provided more reasonable/useful compilation statistics. After this change Sun Ada compiled 24 of 54 files yielding 29%, and GNAT compiled 22 of 84 files yielding 21%. Generally, the same files compiled across the 3 compilers.

For APL, 1 of the 84 QA files, 070DCER1, compiled with VAX, Sun, and GNAT yielding 1% each. APL’s low percentage of correct compilations was caused by a high number of syntax errors and extraneous characters appearing in its generated Ada.

For CCCC, 14 of the 84 QA files compiled with VAX yielding 17%. However, the second number inside parentheses, also 17%, is probably a better indicator since CCCC only generated Ada code for 83 QA files (14/83 = 17%). Initially, none of CCCC’s 83 files compiled for either Sun or GNAT. Investigation showed dependency on a proprietary DEC math library, math_lib, available on VAX but not on Sun SPARC. For GNAT substituting the Ada 95 Ada.Numerics.Generic_Elementary_Functions for math_lib corrected a transportability problem which guaranteed 100% failure. For Sun substituting the proprietary math library, math, for math_lib corrected the same transportability problem. We believe these changes provided more reasonable/useful compilation statistics. After this change Sun and GNAT both compiled the same 10 of 83 files with 1 exception, yielding 12%.

**INVESTIGATION OF COMPILATION ERRORS**

Using VAX Ada we looked deeper into the quantity and nature of the syntactic and semantic compilation errors. This information, discussed in the next four paragraphs, is not in a table.

For TRADA, 1003 errors were produced from the VAX compilation of 54 QA files. (30 files produced compilation errors). This averages 33 errors per unsuccessful compile (1003/30). The range was between 1 and 278 errors per compile. About a half dozen syntax errors were reported in the generated Ada code; the rest were semantic errors.

For APL, 2349 errors were produced from the 83 unsuccessful VAX compilations averaging 28 errors per compilation. The range was between 4 and 69 errors per compilation. A high percentage of APL’s errors, about 2/3, were Ada syntactical errors or illegal characters in the source files. These syntax errors guaranteed a high percentage of unsuccessful compilations. These will require either fixing the translator, or reengineering the generated Ada before many of the semantic errors will be exposed.
For CCCC, 1713 errors existed over 69 unsuccessful VAX compilations averaging 25 errors per compilation. The range was from 1 through 178 errors per compile. Less than two percent of errors reported in CCCC's generated code were syntactic; the rest were semantic errors.

Across all three translators the average was 28 errors per unsuccessful compilation. These were usually not 28 separate and distinct errors, but probably about 6 different categories of similar errors meaning that one correction may resolve four or five distinct errors. Due to the nature of compilers, many corrections have potential to expose the next layer of errors. Several correction passes are likely required to achieve a correct compilation at this first level. At the next level non-translatables, bracketed in Ada comments by translators, such as direct code, must be reengineered on either the CMS-2 or Ada side to reach a correct compilation. Final reengineering will probably be necessary to achieve execution that is functionally equivalent to execution of the CMS-2. We consider this observation of multiple level issues very important since considerable time must be spent addressing each and every translation problem.

PROJECT-CONTRIBUTED LEGACY CMS-2 SAMPLES

In addition to using files from the CMS-2 QA test suite, five projects contributed source code for translation/compile research. Results are shown in Table B-3, Translating and Compiling using Project-Contributed Legacy CMS-2 Source Code. This table combines translation and compilation results, and also shows adjustments made to source code before translation, and resultant errors. Each project table entry contains translation pass, quit or catastrophic failure; minutes of wall clock time; Ada compiler results (VAX Ada/Sun Ada/GNAT); Ada SLOC; and descriptive comments.
Table B-1. Stress Testing Using MTASS Test Suite - Translation Information - 1

<table>
<thead>
<tr>
<th>Test Description</th>
<th>MTASS CMS-2 User Handbook Section</th>
<th>File Name from MTASS CMS-2 Test Suite</th>
<th>Test Type</th>
<th>TRADA Pass, Quit, or Fail &amp; min. of wall clock time</th>
<th>APL Pass, Quit, or Fail &amp; min. of wall clock time</th>
<th>CCCC Pass, Quit, or Fail &amp; min. of wall clock time</th>
<th>CMS-2 Lines of Code (SLOC)*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Controller</td>
<td>N/A</td>
<td>SYSDD &amp; QTCON</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A 1543</td>
</tr>
<tr>
<td>Delimiters</td>
<td>3.2.1</td>
<td>N/A</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Digits</td>
<td>3.2.2</td>
<td>N/A</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decimal Digits</td>
<td>3.2.2.1</td>
<td>N/A</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Octal Digits</td>
<td>3.2.2.2</td>
<td>N/A</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hexadecimal Digits</td>
<td>3.2.2.3</td>
<td>N/A</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>–</td>
<td>Not tested</td>
</tr>
<tr>
<td>B</td>
<td>Both self-checking and manual</td>
</tr>
<tr>
<td>M</td>
<td>Manual check</td>
</tr>
<tr>
<td>N</td>
<td>Non-specifically tested in other tests</td>
</tr>
<tr>
<td>S</td>
<td>Automated/ self-checking</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Translation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Catastrophic translator Failure</td>
</tr>
<tr>
<td>F,P</td>
<td>Failed; translator corrected; passed</td>
</tr>
<tr>
<td>N/A</td>
<td>Not applicable</td>
</tr>
<tr>
<td>P</td>
<td>Passed/completed translation - Ada code generated</td>
</tr>
<tr>
<td>Q</td>
<td>Quit translation due to errors - user notified</td>
</tr>
<tr>
<td>*</td>
<td>All SLOC is straight lines of text</td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Letters</td>
<td>3.2.3</td>
</tr>
<tr>
<td>Name</td>
<td>3.2.4</td>
</tr>
<tr>
<td>Name List</td>
<td>3.2.4.1</td>
</tr>
<tr>
<td>Tag (EQUALS)</td>
<td>3.2.5</td>
</tr>
<tr>
<td>Tag Term</td>
<td>3.2.5.1</td>
</tr>
<tr>
<td>Data Unit</td>
<td>3.2.6</td>
</tr>
<tr>
<td>Constant</td>
<td>3.2.7</td>
</tr>
<tr>
<td>Numeric Constant (CMODE)</td>
<td>3.2.7.1</td>
</tr>
<tr>
<td>Octal Constant</td>
<td>3.2.7.1.1</td>
</tr>
<tr>
<td>Decimal Constant</td>
<td>3.2.7.1.2</td>
</tr>
<tr>
<td>Hexadecimal Constant</td>
<td>3.2.7.1.3</td>
</tr>
<tr>
<td>Character Constant</td>
<td>3.2.7.2</td>
</tr>
<tr>
<td>Status Constant</td>
<td>3.2.7.3</td>
</tr>
<tr>
<td>Boolean Constant</td>
<td>3.2.7.4</td>
</tr>
<tr>
<td>Notes (COMMENT)</td>
<td>3.2.7.5</td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Numeric Expression</td>
<td>3.2.8.1</td>
</tr>
<tr>
<td></td>
<td>3.2.8.1</td>
</tr>
<tr>
<td></td>
<td>3.2.8.1</td>
</tr>
<tr>
<td>Numeric Expression (MSCALE)</td>
<td>3.2.8.1</td>
</tr>
<tr>
<td></td>
<td>3.2.8.1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Boolean Expression</td>
<td>3.2.8.2</td>
</tr>
<tr>
<td>Relational Expression</td>
<td>3.2.8.2.1</td>
</tr>
<tr>
<td>Status Expression</td>
<td>3.2.8.3</td>
</tr>
<tr>
<td>Character Expression</td>
<td>3.2.8.4</td>
</tr>
<tr>
<td>Bit String Expression</td>
<td>3.2.8.5</td>
</tr>
<tr>
<td>Conditional Expression</td>
<td>3.2.9.1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Simple Type</td>
<td>3.2.10</td>
</tr>
<tr>
<td>Type Decl</td>
<td>3.2.11</td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>CMS-2 System/End System Decs</td>
<td>3.3.1</td>
</tr>
<tr>
<td>Comments</td>
<td>3.3.2</td>
</tr>
<tr>
<td>CSWITCH</td>
<td>3.3.3</td>
</tr>
<tr>
<td>Header Blocks</td>
<td>3.4.1</td>
</tr>
<tr>
<td>Options Decs</td>
<td>3.4.1</td>
</tr>
<tr>
<td>System Index Decl</td>
<td>3.4.3</td>
</tr>
<tr>
<td>Debug Decl</td>
<td>3.4.4</td>
</tr>
<tr>
<td>Address Counter Separation Decl</td>
<td>3.4.5</td>
</tr>
<tr>
<td></td>
<td>3.4.5</td>
</tr>
<tr>
<td>Equals Decl</td>
<td>3.4.6</td>
</tr>
<tr>
<td></td>
<td>3.4.6</td>
</tr>
<tr>
<td>Substitution Decl</td>
<td>3.4.7</td>
</tr>
<tr>
<td>Constant Mode Decl</td>
<td>3.4.8</td>
</tr>
<tr>
<td>Executive Decl</td>
<td>3.4.9</td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Pooling Decl</td>
<td>3.4.10</td>
</tr>
<tr>
<td></td>
<td>3.4.10</td>
</tr>
<tr>
<td>LTAG Variable Decl</td>
<td>3.4.10</td>
</tr>
<tr>
<td></td>
<td>3.4.11</td>
</tr>
<tr>
<td>Mode Decl</td>
<td>3.4.12</td>
</tr>
<tr>
<td></td>
<td>3.4.12</td>
</tr>
<tr>
<td>Single Precision Decl</td>
<td>3.4.13</td>
</tr>
<tr>
<td>Parameter Passing Decl</td>
<td>3.4.14</td>
</tr>
<tr>
<td></td>
<td>3.4.14</td>
</tr>
<tr>
<td></td>
<td>3.4.14</td>
</tr>
<tr>
<td>CSWITCH Delete Decl</td>
<td>3.4.15</td>
</tr>
<tr>
<td></td>
<td>3.4.15</td>
</tr>
<tr>
<td>Spill Decl</td>
<td>3.4.16</td>
</tr>
<tr>
<td></td>
<td>3.4.16</td>
</tr>
<tr>
<td>Scaling Mode Decl</td>
<td>3.4.17</td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>System Data Design</td>
<td>3.5.1</td>
</tr>
<tr>
<td>Direct Code Block</td>
<td>3.5.2</td>
</tr>
<tr>
<td></td>
<td>3.5.2</td>
</tr>
<tr>
<td></td>
<td>3.5.2</td>
</tr>
<tr>
<td></td>
<td>3.5.2</td>
</tr>
<tr>
<td>Table Decl</td>
<td>3.5.3</td>
</tr>
<tr>
<td>Field Decl</td>
<td>3.5.3.1</td>
</tr>
<tr>
<td></td>
<td>070QA17</td>
</tr>
<tr>
<td></td>
<td>070QA16</td>
</tr>
<tr>
<td></td>
<td>070QA14</td>
</tr>
<tr>
<td>Like-Table Decl</td>
<td>3.5.3.2</td>
</tr>
<tr>
<td>Item-Area Decl</td>
<td>3.5.3.2</td>
</tr>
<tr>
<td>Sub-Table Decl</td>
<td>3.5.3.4</td>
</tr>
<tr>
<td>Field-Overlay Decl</td>
<td>3.5.3.5</td>
</tr>
<tr>
<td>Modifier</td>
<td>3.5.4</td>
</tr>
<tr>
<td>External Program Decl</td>
<td>3.5.5</td>
</tr>
</tbody>
</table>

B-14
<table>
<thead>
<tr>
<th>Test Description</th>
<th>MTASS CMS-2 User Handbook Section</th>
<th>File Name from MTASS CMS-2 Test Suite</th>
<th>Test Type</th>
<th>TRADA Pass, Quit, or Fail &amp; min. of wall clock time</th>
<th>APL Pass, Quit, or Fail &amp; min. of wall clock time</th>
<th>CCCC Pass, Quit, or Fail &amp; min. of wall clock time</th>
<th>CMS-2 Lines of Code (SLOC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Decl</td>
<td>3.5.6</td>
<td>N/A</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Procedure Sw Bl</td>
<td>3.5.7</td>
<td>070QA5</td>
<td>S</td>
<td>F,P (4)</td>
<td>P (2)</td>
<td>P (17)</td>
<td>450</td>
</tr>
<tr>
<td>PINDEX Decl</td>
<td>3.5.7.1</td>
<td>N/A</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PITEM Switch Block</td>
<td>3.5.7.2</td>
<td>N/A</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PDOUBLE Switch</td>
<td>3.5.7.3</td>
<td>N/A</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameter Decl</td>
<td>3.5.8</td>
<td>070QA2</td>
<td>S</td>
<td>Q (1)</td>
<td>F,P (3)</td>
<td>P (25)</td>
<td>2075</td>
</tr>
<tr>
<td>Overlay Decl</td>
<td>3.5.9</td>
<td>070QA3</td>
<td>S</td>
<td>Q (1)</td>
<td>P (2)</td>
<td>P (23)</td>
<td>976</td>
</tr>
<tr>
<td></td>
<td>3.5.9</td>
<td>070QA60</td>
<td>S</td>
<td>F,P (4)</td>
<td>F,P (3)</td>
<td>P (96)</td>
<td>1019</td>
</tr>
<tr>
<td>Data Statement</td>
<td>3.5.10</td>
<td>N/A</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Range Decl</td>
<td>3.5.11</td>
<td>070QA95</td>
<td>S</td>
<td>Q (1)</td>
<td>P (3)</td>
<td>P (17)</td>
<td>477</td>
</tr>
<tr>
<td>Format Decl</td>
<td>3.5.12</td>
<td>070QA8</td>
<td>B</td>
<td>Q (1)</td>
<td>P (2)</td>
<td>P (11)</td>
<td>658</td>
</tr>
<tr>
<td>Nonstandard File Decl</td>
<td>3.5.13.1</td>
<td>070QA64</td>
<td>S</td>
<td>Q (1)</td>
<td>P (2)</td>
<td>F,P (21)</td>
<td>911</td>
</tr>
<tr>
<td>Standard File Decl</td>
<td>3.5.13.2</td>
<td>070QA8</td>
<td>B</td>
<td>Q (1)</td>
<td>P (2)</td>
<td>P (11)</td>
<td>658</td>
</tr>
<tr>
<td>Stringform Decl</td>
<td>3.5.14</td>
<td>N/A</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inputlist Decl</td>
<td>3.5.15</td>
<td>070QA18</td>
<td>S</td>
<td>Q (1)</td>
<td>P (5)</td>
<td>F,P (37)</td>
<td>4074</td>
</tr>
<tr>
<td>Outpilistlist Decl</td>
<td>3.5.16</td>
<td>070QA18</td>
<td>S</td>
<td>Q (1)</td>
<td>P (5)</td>
<td>F,P (37)</td>
<td>4074</td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
<td>File Name from MTASS CMS-2 Test Suite</td>
<td>Test Type</td>
<td>TRADA Pass, Quit, or Fail &amp; min. of wall clock time</td>
<td>APL Pass, Quit, or Fail &amp; min. of wall clock time</td>
<td>CCCC Pass, Quit, or Fail &amp; min. of wall clock time</td>
<td>CMS-2 Lines of Code (SLOC)</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------</td>
<td>--------------------------------------</td>
<td>-----------</td>
<td>----------------------------------------------------</td>
<td>---------------------------------------------------</td>
<td>---------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>System Procedure Stmt</td>
<td>3.6.1</td>
<td>N/A</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local Data Design</td>
<td>3.6.2.1</td>
<td>N/A</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index Switch Block</td>
<td>3.6.2.1.1</td>
<td>N/A</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Item Switch Block</td>
<td>3.6.2.1.2</td>
<td>N/A</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double Switch Block</td>
<td>3.6.2.1.3</td>
<td>N/A</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local Program Decl</td>
<td>3.6.2.1.4</td>
<td>N/A</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Auto Data Design</td>
<td>3.6.2.2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Procedure Block Decl</td>
<td>3.6.3.1</td>
<td>070QA7A</td>
<td>S</td>
<td>P (14)</td>
<td>P (20)</td>
<td>P,F (34)</td>
<td>3907</td>
</tr>
<tr>
<td>Function Block Decl</td>
<td>3.6.3.2</td>
<td>070QA7A</td>
<td>S</td>
<td>P (14)</td>
<td>P (20)</td>
<td>P,F (34)</td>
<td>3907</td>
</tr>
<tr>
<td>Exec Proc Block Decl</td>
<td>3.6.3.3</td>
<td>070QA22</td>
<td>S</td>
<td>P (6)</td>
<td>P (2)</td>
<td>P (18)</td>
<td>804</td>
</tr>
<tr>
<td>Local Index List</td>
<td>3.7.1.1</td>
<td>N/A</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subprogram Data Design</td>
<td>3.7.1.2</td>
<td>070QA542</td>
<td>S</td>
<td>Q (1)</td>
<td>P (1)</td>
<td>P (24)</td>
<td>1468</td>
</tr>
<tr>
<td>Imperative Stmt</td>
<td>3.7.2</td>
<td>N/A</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set Phrase</td>
<td>3.7.3.1</td>
<td>070QA10</td>
<td>S</td>
<td>P (17)</td>
<td>P (14)</td>
<td>P (41)</td>
<td>3672</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA14</td>
<td>S</td>
<td>P (8)</td>
<td>P (9)</td>
<td>P (24)</td>
<td>1880</td>
</tr>
<tr>
<td>Begin Phrase</td>
<td>3.7.3.2</td>
<td>N/A</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return Phrase</td>
<td>3.7.3.3</td>
<td>070QA82A</td>
<td>S</td>
<td>P (3)</td>
<td>F,P (2)</td>
<td>P (15)</td>
<td>91</td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
<td>File Name from MTASS CMS-2 Test Suite</td>
<td>Test Type</td>
<td>TRADA Pass, Quit, or Fail &amp; min. of wall clock time</td>
<td>APL Pass, Quit, or Fail &amp; min. of wall clock time</td>
<td>CCC Pass, Quit, or Fail &amp; min. of wall clock time</td>
<td>CMS-2 Lines of Code (SLOC)</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------</td>
<td>----------------------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------</td>
<td>--------------------------------------------------</td>
<td>--------------------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA82B</td>
<td>S</td>
<td>P (2)</td>
<td>F,P (1)</td>
<td>P (15)</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA82C</td>
<td>S</td>
<td>Q (1)</td>
<td>P (1)</td>
<td>P (15)</td>
<td>114</td>
</tr>
<tr>
<td>Simple Goto Phrase</td>
<td>3.7.3.4.1</td>
<td>070QA83A</td>
<td>S</td>
<td>P (2)</td>
<td>P (1)</td>
<td>P (14)</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA83B</td>
<td>S</td>
<td>P (2)</td>
<td>P (1)</td>
<td>P (14)</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA83C</td>
<td>S</td>
<td>Q (1)</td>
<td>P (1)</td>
<td>P (14)</td>
<td>112</td>
</tr>
<tr>
<td>Index Goto Phrase</td>
<td>3.7.3.4.2</td>
<td>070QA4</td>
<td>S</td>
<td>Q (1)</td>
<td>F,P (2)</td>
<td>P (17)</td>
<td>1342</td>
</tr>
<tr>
<td>Item Goto Phrase</td>
<td>3.7.3.4.3</td>
<td>070QA4</td>
<td>S</td>
<td>Q (1)</td>
<td>F,P (2)</td>
<td>P (17)</td>
<td>1342</td>
</tr>
<tr>
<td>User Proc Call Phrase</td>
<td>3.7.3.5.1</td>
<td>070QA7B</td>
<td>S</td>
<td>P (16)</td>
<td>P (11)</td>
<td>P (48)</td>
<td>4603</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA86</td>
<td>S</td>
<td>P (3)</td>
<td>P (1)</td>
<td>P (19)</td>
<td>439</td>
</tr>
<tr>
<td>Supplied Proc Call Phrase</td>
<td>3.7.3.5.2</td>
<td>070QA538</td>
<td>S</td>
<td>Q (1)</td>
<td>P (2)</td>
<td>P (23)</td>
<td>2998</td>
</tr>
<tr>
<td>PINDEX Switch Call</td>
<td>3.7.3.6.1</td>
<td>070QA4</td>
<td>S</td>
<td>Q (1)</td>
<td>F,P (2)</td>
<td>P (17)</td>
<td>1342</td>
</tr>
<tr>
<td>PITEM Call Phrase</td>
<td>3.7.3.6.2</td>
<td>N/A</td>
<td>N/A</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Vary Block</td>
<td>3.7.3.7</td>
<td>070QA6</td>
<td>S</td>
<td>Q (1)</td>
<td>P (3)</td>
<td>F,P (28)</td>
<td>3294</td>
</tr>
<tr>
<td>Stop Phrase</td>
<td>3.7.3.8</td>
<td>070QA90</td>
<td>S</td>
<td>Q (1)</td>
<td>P (1)</td>
<td>P (15)</td>
<td>134</td>
</tr>
<tr>
<td>Resume Phrase</td>
<td>3.7.3.9</td>
<td>070QA6</td>
<td>S</td>
<td>Q (1)</td>
<td>P (3)</td>
<td>F,P (28)</td>
<td>3294</td>
</tr>
<tr>
<td>For Block</td>
<td>3.7.3.10</td>
<td>070QA6</td>
<td>S</td>
<td>Q (1)</td>
<td>P (3)</td>
<td>F,P (28)</td>
<td>3294</td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
<td>File Name from MTASS CMS-2 Test Suite</td>
<td>Test Type</td>
<td>TRADA Pass, Quit, or Fall &amp; min. of wall clock time</td>
<td>APL Pass, Quit, or Fall &amp; min. of wall clock time</td>
<td>CCCC Pass, Quit, or Fall &amp; min. of wall clock time</td>
<td>CMS-2 Lines of Code (SLOC)</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------</td>
<td>---------------------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------</td>
<td>--------------------------------------------------</td>
<td>--------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Exec Phrase</td>
<td>3.7.3.11</td>
<td>070QA23</td>
<td>S</td>
<td>P (5)</td>
<td>P (2)</td>
<td>P (17)</td>
<td>903</td>
</tr>
<tr>
<td>Shift Phrase</td>
<td>3.7.3.12</td>
<td>070QA2</td>
<td>S</td>
<td>Q (1)</td>
<td>F,P (3)</td>
<td>P (25)</td>
<td>2075</td>
</tr>
<tr>
<td>Display Phrase</td>
<td>3.7.3.13.1</td>
<td>070QA95</td>
<td>M</td>
<td>Q (1)</td>
<td>P (3)</td>
<td>P (17)</td>
<td>477</td>
</tr>
<tr>
<td>Snap Phrase</td>
<td>3.7.3.13.2</td>
<td>070QA95</td>
<td>M</td>
<td>Q (1)</td>
<td>P (3)</td>
<td>P (17)</td>
<td>477</td>
</tr>
<tr>
<td>Trace Phrase</td>
<td>3.7.3.13.3</td>
<td>070QA95</td>
<td>M</td>
<td>Q (1)</td>
<td>P (3)</td>
<td>P (17)</td>
<td>477</td>
</tr>
<tr>
<td>End Trace Phrase</td>
<td>3.7.3.13.4</td>
<td>070QA95</td>
<td>M</td>
<td>Q (1)</td>
<td>P (3)</td>
<td>P (17)</td>
<td>477</td>
</tr>
<tr>
<td>Swap Phrase</td>
<td>3.7.3.14</td>
<td>070QA2</td>
<td>S</td>
<td>Q (1)</td>
<td>F,P (3)</td>
<td>P (25)</td>
<td>2075</td>
</tr>
<tr>
<td>Pack Phrase</td>
<td>3.7.3.15</td>
<td>070QA15</td>
<td>S</td>
<td>P (13)</td>
<td>F,P (5)</td>
<td>P (27)</td>
<td>3358</td>
</tr>
<tr>
<td>Open Phrase</td>
<td>3.7.3.16.1</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Close Phrase</td>
<td>3.7.3.16.2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Endfile Phrase</td>
<td>3.7.3.16.3</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEFID Phrase</td>
<td>3.7.3.16.4</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHKID Phrase</td>
<td>3.7.3.16.5</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIL POS Phrase</td>
<td>3.7.3.16.6</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
<td>File Name from MTASS CMS-2 Test Suite</td>
<td>Test Type</td>
<td>TRADA Pass, Quit, or Fail &amp; min. of wall clock time</td>
<td>APL Pass, Quit, or Fail &amp; min. of wall clock time</td>
<td>CCC Pass, Quit, or Fail &amp; min. of wall clock time</td>
<td>CMS-2 Lines of Code (SLOC)</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------</td>
<td>--------------------------------------</td>
<td>-----------</td>
<td>---------------------------------------------------</td>
<td>--------------------------------------------------</td>
<td>--------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>SET POS Phrase</td>
<td>3.7.3.16.7</td>
<td>N/A</td>
<td></td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output to the Printer</td>
<td>3.7.3.16.8.1</td>
<td>070QA18</td>
<td>S</td>
<td>Q (1)</td>
<td>P (5)</td>
<td>F,P (37)</td>
<td>4074</td>
</tr>
<tr>
<td>Input Phrase</td>
<td>3.7.3.16.9</td>
<td>070QA8</td>
<td>B</td>
<td>Q (1)</td>
<td>P (2)</td>
<td>P (11)</td>
<td>658</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA91</td>
<td>S</td>
<td>P (3)</td>
<td>P (1)</td>
<td>P (15)</td>
<td>249</td>
</tr>
<tr>
<td>Encode Phrase</td>
<td>3.7.3.16.10</td>
<td>N/A</td>
<td></td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decode Phrase</td>
<td>3.7.3.16.11</td>
<td>070QA8</td>
<td>B</td>
<td>Q (1)</td>
<td>P (2)</td>
<td>P (11)</td>
<td>658</td>
</tr>
<tr>
<td>Format Scan</td>
<td>3.7.3.16.12</td>
<td>N/A</td>
<td></td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Null Phrase</td>
<td>3.7.3.17</td>
<td>N/A</td>
<td></td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exit Phrase</td>
<td>3.7.3.18</td>
<td>N/A</td>
<td></td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>If Clause</td>
<td>3.7.4.1</td>
<td>070QA2</td>
<td>S</td>
<td>Q (1)</td>
<td>F,P (3)</td>
<td>P (25)</td>
<td>2075</td>
</tr>
<tr>
<td>Else Clause</td>
<td>3.7.4.2</td>
<td>070QA2</td>
<td>S</td>
<td>Q (1)</td>
<td>F,P (3)</td>
<td>P (25)</td>
<td>2075</td>
</tr>
<tr>
<td>Find Clause</td>
<td>3.7.4.3</td>
<td>070QA2</td>
<td>S</td>
<td>Q (1)</td>
<td>F,P (3)</td>
<td>P (25)</td>
<td>2075</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA6</td>
<td>S</td>
<td>Q (1)</td>
<td>P (3)</td>
<td>F,P (25)</td>
<td>3294</td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
<td>File Name from MTASS CMS-2 Test Suite</td>
<td>Test Type</td>
<td>TRADA Pass, Quit, or Fail &amp; min. of wall clock time</td>
<td>APL Pass, Quit, or Fail &amp; min. of wall clock time</td>
<td>CCCC Pass, Quit, or Fail &amp; min. of wall clock time</td>
<td>CMS-2 Lines of Code (SLOC)</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------</td>
<td>--------------------------------------</td>
<td>-----------</td>
<td>---------------------------------------------------</td>
<td>------------------------------------------------</td>
<td>------------------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>User Function Call</td>
<td>3.8.1</td>
<td>070QA7B</td>
<td>S</td>
<td>P (16)</td>
<td>F,P (11)</td>
<td>P (48)</td>
<td>4603</td>
</tr>
<tr>
<td>Predefined Function Call</td>
<td>3.8.2.1 - 3.8.2.22</td>
<td>070QA2</td>
<td>S</td>
<td>Q (1)</td>
<td>F,P (3)</td>
<td>P (25)</td>
<td>2075</td>
</tr>
<tr>
<td>3.8.2.1-22</td>
<td></td>
<td>070QA12</td>
<td>S</td>
<td>P (11)</td>
<td>P (8)</td>
<td>P (30)</td>
<td>2841</td>
</tr>
<tr>
<td>3.8.2.1-22</td>
<td></td>
<td>070QA11A</td>
<td>S</td>
<td>P (21)</td>
<td>P (7)</td>
<td>P (55)</td>
<td>4798</td>
</tr>
<tr>
<td>3.8.2.1-22</td>
<td></td>
<td>070QA11B</td>
<td>S</td>
<td>P (16)</td>
<td>P (12)</td>
<td>P (53)</td>
<td>4792</td>
</tr>
<tr>
<td>3.8.2.1-22</td>
<td></td>
<td>070QA11C</td>
<td>S</td>
<td>P (16)</td>
<td>P (8)</td>
<td>P (42)</td>
<td>4786</td>
</tr>
<tr>
<td>3.8.2.1 - 3.8.2.22</td>
<td></td>
<td>070QA13A</td>
<td>S</td>
<td>P (16)</td>
<td>P (10)</td>
<td>P (56)</td>
<td>4908</td>
</tr>
<tr>
<td>3.8.2.1-22</td>
<td></td>
<td>070QA13B</td>
<td>S</td>
<td>P (16)</td>
<td>P (10)</td>
<td>P (53)</td>
<td>4908</td>
</tr>
<tr>
<td>3.8.2.1-22</td>
<td></td>
<td>070QA13C</td>
<td>S</td>
<td>P (12)</td>
<td>P (8)</td>
<td>P (34)</td>
<td>4906</td>
</tr>
<tr>
<td>3.8.2.1-22</td>
<td></td>
<td>070QA1</td>
<td>S</td>
<td>F,P (5)</td>
<td>P (2)</td>
<td>F,P (18)</td>
<td>627</td>
</tr>
<tr>
<td>3.8.2.1-22</td>
<td></td>
<td>070QA538</td>
<td>S</td>
<td>Q (1)</td>
<td>P (2)</td>
<td>P (23)</td>
<td>2998</td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
<td>File Name from MTASS CMS-2 Test Suite</td>
<td>Test Type</td>
<td>TRADA Pass, Quit, or Fail &amp; min. of wall clock time</td>
<td>APL Pass, Quit, or Fail &amp; min. of wall clock time</td>
<td>CCC Pass, Quit, or Fail &amp; min. of wall clock time</td>
<td>CMS-2 Lines of Code (SLOC)</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------</td>
<td>---------------------------------------</td>
<td>-----------</td>
<td>----------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Predefined Function Call</td>
<td>3.8.2.1-22</td>
<td>070QA538A</td>
<td>S</td>
<td>P (4)</td>
<td>P (2)</td>
<td>P (17)</td>
<td>674</td>
</tr>
<tr>
<td>(continued)</td>
<td>3.8.2.1-22</td>
<td>070QA538B</td>
<td>S</td>
<td>P (4)</td>
<td>P (2)</td>
<td>P (17)</td>
<td>622</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA538C</td>
<td>S</td>
<td>P (4)</td>
<td>P (2)</td>
<td>P (19)</td>
<td>925</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA538D</td>
<td>S</td>
<td>Q (1)</td>
<td>P (2)</td>
<td>P (18)</td>
<td>979</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA538E</td>
<td>S</td>
<td>Q (1)</td>
<td>P (2)</td>
<td>P (18)</td>
<td>995</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA582</td>
<td>S</td>
<td>Q (1)</td>
<td>F,P (2)</td>
<td>P (17)</td>
<td>595</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>07FQA582F1</td>
<td>S</td>
<td>Q (1)</td>
<td>F,P (2)</td>
<td>P (17)</td>
<td>603</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>07FQA582F2</td>
<td>S</td>
<td>Q (1)</td>
<td>F,P (2)</td>
<td>P (17)</td>
<td>679</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA539</td>
<td>S</td>
<td>P (6)</td>
<td>P (2)</td>
<td>P (22)</td>
<td>2060</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA539A</td>
<td>S</td>
<td>P (3)</td>
<td>P (2)</td>
<td>P (17)</td>
<td>511</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA539B</td>
<td>S</td>
<td>P (4)</td>
<td>P (2)</td>
<td>P (17)</td>
<td>476</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA539C</td>
<td>S</td>
<td>P (6)</td>
<td>P (2)</td>
<td>P (18)</td>
<td>795</td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
<td>File Name from MTASS CMS-2 Test Suite</td>
<td>Test Type</td>
<td>TRADA Pass, Quit, or Fall &amp; min. of wall clock time</td>
<td>APL Pass, Quit, or Fall &amp; min. of wall clock time</td>
<td>CCCC Pass, Quit, or Fall &amp; min. of wall clock time</td>
<td>CMS-2 Lines of Code (SLOC)</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------</td>
<td>--------------------------------------</td>
<td>-----------</td>
<td>--------------------------------------------------</td>
<td>--------------------------------------------------</td>
<td>--------------------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Predefined Function Call</td>
<td>3.8.2.1-22</td>
<td>070QA539D</td>
<td>S</td>
<td>P (4)</td>
<td>P (2)</td>
<td>P (18)</td>
<td>867</td>
</tr>
<tr>
<td>(continued)</td>
<td></td>
<td></td>
<td></td>
<td>6 UNIQUE FAILURES</td>
<td>11 UNIQUE FAILURES</td>
<td>10 UNIQUE FAILURES</td>
<td></td>
</tr>
<tr>
<td>TOTAL FAILS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL UNIQUE SLOC</td>
<td></td>
<td></td>
<td></td>
<td>06 HOURS</td>
<td>04 HOURS</td>
<td>31 HOURS</td>
<td>117.7K</td>
</tr>
<tr>
<td>INPUT TO TRANSLATORS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL TIMES</td>
<td></td>
<td></td>
<td></td>
<td>22 MINS. / 09 HOURS 30 MINS.</td>
<td>42 MINS.</td>
<td>59 MINS.</td>
<td></td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
<td>File Name from MTASS CMS-2 Test Suite</td>
<td>Test Num</td>
<td>TRADA</td>
<td>APL</td>
<td>CCCC</td>
<td>CMS-2 SLOC w/SY SDD&amp; QTCON</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------</td>
<td>--------------------------------------</td>
<td>----------</td>
<td>-------</td>
<td>-----</td>
<td>------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Test Controller</td>
<td>N/A</td>
<td>SYSDD &amp; QTCON</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>1543</td>
</tr>
<tr>
<td>Delimiters</td>
<td>3.2.1</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Digits</td>
<td>3.2.2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decimal Digits</td>
<td>3.2.2.1</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Octal Digits</td>
<td>3.2.2.2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hexadecimal Digits</td>
<td>3.2.2.3</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Letters</td>
<td>3.2.3</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>3.2.4</td>
<td>070QA541</td>
<td>T51</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 4191</td>
<td>U/U/U/ 8969</td>
<td>2199</td>
</tr>
<tr>
<td>Name List</td>
<td>3.2.4.1</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tag (EQUALS)</td>
<td>3.2.5</td>
<td>070QA20</td>
<td>T18</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 7531</td>
<td>U/U/U/ 14643</td>
<td>4352</td>
</tr>
</tbody>
</table>

**Compiles VAX/Sun/GNT Meaning**

- **C** Correct compilation of generated Ada code
- **X** No Ada code generated by translator - no compile possible
- **U** Unsuccessful Ada compilation - errors present or informational message states that a constraint error will be raised during execution

For example, U/C/U/ 5000 means 5000 source lines of Ada code (SLOC) produced by the translator unsuccessfully compiled with VAX Ada, correct with Sun Ada, and unsuccessful with GNAT Ada.

B-23
<table>
<thead>
<tr>
<th>MTASS CMS-2 User Handbook Section</th>
<th>Test Description</th>
<th>File Name from Test Suite</th>
<th>Test Num</th>
<th>CCC</th>
<th>CMODE</th>
<th>APL SLOC w/</th>
<th>SLOC w/</th>
<th>QtyCon</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.5.1</td>
<td>Tag Term</td>
<td>N/A</td>
<td>3.2.7.1</td>
<td>N/A</td>
<td>N/A</td>
<td>T62</td>
<td>T62</td>
<td>2201</td>
</tr>
<tr>
<td>3.2.6</td>
<td>Data Unit</td>
<td>N/A</td>
<td>3.2.7.1.1</td>
<td>N/A</td>
<td>070QA8</td>
<td>070QA8</td>
<td>070QA8</td>
<td>2201</td>
</tr>
<tr>
<td>3.2.7</td>
<td>Constant</td>
<td>N/A</td>
<td>3.2.7.1.2</td>
<td>N/A</td>
<td>N/A</td>
<td>T53</td>
<td>T53</td>
<td>2138</td>
</tr>
<tr>
<td></td>
<td>Numeric Constant</td>
<td>N/A</td>
<td>3.2.7.1.3</td>
<td>N/A</td>
<td>N/A</td>
<td>T53</td>
<td>T53</td>
<td>2138</td>
</tr>
<tr>
<td></td>
<td>Octal Constant</td>
<td>N/A</td>
<td>3.2.7.2</td>
<td>N/A</td>
<td>N/A</td>
<td>T53</td>
<td>T53</td>
<td>2138</td>
</tr>
<tr>
<td></td>
<td>Decimal Constant</td>
<td>N/A</td>
<td>3.2.7.3</td>
<td>N/A</td>
<td>N/A</td>
<td>T53</td>
<td>T53</td>
<td>2138</td>
</tr>
<tr>
<td></td>
<td>Hexadecimal Constant</td>
<td>N/A</td>
<td>3.2.7.4</td>
<td>N/A</td>
<td>N/A</td>
<td>T53</td>
<td>T53</td>
<td>2138</td>
</tr>
<tr>
<td></td>
<td>Character Constant</td>
<td>N/A</td>
<td>3.2.8.1</td>
<td>N/A</td>
<td>N/A</td>
<td>T53</td>
<td>T53</td>
<td>2138</td>
</tr>
<tr>
<td></td>
<td>Status Constant</td>
<td>N/A</td>
<td>3.2.8.1</td>
<td>N/A</td>
<td>N/A</td>
<td>T53</td>
<td>T53</td>
<td>2138</td>
</tr>
<tr>
<td></td>
<td>Boolean Constant</td>
<td>N/A</td>
<td>3.2.8.1</td>
<td>N/A</td>
<td>N/A</td>
<td>T53</td>
<td>T53</td>
<td>2138</td>
</tr>
<tr>
<td></td>
<td>Notes (COMMENT)</td>
<td>N/A</td>
<td>3.2.8.1</td>
<td>N/A</td>
<td>N/A</td>
<td>T53</td>
<td>T53</td>
<td>2138</td>
</tr>
<tr>
<td></td>
<td>Numeric Expression</td>
<td>N/A</td>
<td>3.2.8.1</td>
<td>N/A</td>
<td>N/A</td>
<td>T53</td>
<td>T53</td>
<td>2138</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.2.8.1</td>
<td>N/A</td>
<td>N/A</td>
<td>T53</td>
<td>T53</td>
<td>2138</td>
</tr>
</tbody>
</table>

Note: CMODE is represented as CMODE | APL SLOC w/ | SLOC w/ | QtyCon
<table>
<thead>
<tr>
<th>Test Description</th>
<th>MTASS CMS-2 User Handbook Section</th>
<th>File Name from MTASS CMS-2 Test Suite</th>
<th>Test Num</th>
<th>TRADA</th>
<th>APL</th>
<th>CCCC</th>
<th>CMS-2 SLOC w/ SYSDD&amp; QTCN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric Expression (MSCALE)</td>
<td>3.2.8.1</td>
<td>070MS1</td>
<td>T82</td>
<td>U/C/C/ 6848</td>
<td>U/U/U/ 4721</td>
<td>U/U/U/ 10739</td>
<td>2622</td>
</tr>
<tr>
<td></td>
<td>3.2.8.1</td>
<td>070MS2</td>
<td>T83</td>
<td>U/U/U/ 6823</td>
<td>U/U/U/ 4361</td>
<td>U/U/U/ 11502</td>
<td>2650</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070MS1F1</td>
<td>T84</td>
<td>U/C/U/ 6848</td>
<td>U/U/U/ 4722</td>
<td>U/U/U/ 10746</td>
<td>2647</td>
</tr>
<tr>
<td>Boolean Expression</td>
<td>3.2.8.2</td>
<td>070QA2</td>
<td>T17</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 7275</td>
<td>U/U/U/ 12616</td>
<td>3618</td>
</tr>
<tr>
<td>Relational Expression</td>
<td>3.2.8.2.1</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status Expression</td>
<td>3.2.8.3</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Character Expression</td>
<td>3.2.8.4</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit String Expression</td>
<td>3.2.8.5</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conditional Expression</td>
<td>3.2.9.1</td>
<td>070QA2.SCL</td>
<td>T17</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 7275</td>
<td>U/U/U/ 12616</td>
<td>3618</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA16</td>
<td>T13</td>
<td>U/U/U/ 8076</td>
<td>U/U/U/ 6389</td>
<td>U/U/U/ 12025</td>
<td>3496</td>
</tr>
<tr>
<td>Simple Type</td>
<td>3.2.10</td>
<td>070QA584</td>
<td>T56</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 5637</td>
<td>U/U/U/ 11673</td>
<td>2997</td>
</tr>
<tr>
<td>Type Decl</td>
<td>3.2.11</td>
<td>070QA584</td>
<td>T56</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 5637</td>
<td>U/U/U/ 11673</td>
<td>2997</td>
</tr>
<tr>
<td>CMS-2 System/End System Decals</td>
<td>3.3.1</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.3.2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>3.3.3</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSWITH</td>
<td>3.3.4</td>
<td>070QA26</td>
<td>T24</td>
<td>C/C/C/ 3935</td>
<td>U/U/U/ 5054</td>
<td>C/C/C/ 9247</td>
<td>2210</td>
</tr>
</tbody>
</table>

B-25
<table>
<thead>
<tr>
<th>Test Description</th>
<th>Header Blocks</th>
<th>Options Decs</th>
<th>System Index Decl</th>
<th>Debug Decl</th>
<th>Address Counter Separation Decl</th>
<th>Equals Decl</th>
<th>Substitution Decl</th>
<th>Constant Mode Decl</th>
<th>Executive Decl</th>
<th>Pooling Decl</th>
<th>LTAG Variable Decl</th>
<th>Mode Decl</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.1</td>
<td>N/A</td>
<td>N/A</td>
<td>070QA95</td>
<td>T3</td>
<td>X/X/none</td>
<td>U/UI/19902</td>
<td>U/UI/11272</td>
<td>U/UI/11300</td>
<td>U/UI/4347</td>
<td>U/UI/6525</td>
<td>U/UI/5183</td>
<td>2020</td>
</tr>
<tr>
<td>3.4.3</td>
<td>N/A</td>
<td>N/A</td>
<td>070AC1</td>
<td>T4</td>
<td>C/C/4533</td>
<td>U/UI/4732</td>
<td>U/UI/11300</td>
<td>U/UI/9007</td>
<td>U/UI/9807</td>
<td>C/C/9047</td>
<td>U/UI/1292</td>
<td>1925</td>
</tr>
<tr>
<td>3.4.4</td>
<td>070AD1</td>
<td>070AD2</td>
<td>T5</td>
<td>T27</td>
<td>C/C/4519</td>
<td>U/UI/4347</td>
<td>U/UI/11300</td>
<td>U/UI/9007</td>
<td>U/UI/9807</td>
<td>C/C/9047</td>
<td>U/UI/1292</td>
<td>1933</td>
</tr>
<tr>
<td>3.4.5</td>
<td>070AD3</td>
<td>070AD4</td>
<td>T28</td>
<td>T29</td>
<td>X/X/none</td>
<td>U/UI/3552</td>
<td>U/UI/9250</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9696</td>
<td>U/UI/1292</td>
<td>1971</td>
</tr>
<tr>
<td>3.4.6</td>
<td>070QA32A</td>
<td>070QA32B</td>
<td>T30</td>
<td>T30</td>
<td>X/X/none</td>
<td>U/UI/3613</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/1292</td>
<td>1827</td>
</tr>
<tr>
<td>3.4.7</td>
<td>070QA33</td>
<td>070QA34</td>
<td>T31</td>
<td>T31</td>
<td>X/X/none</td>
<td>U/UI/3613</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/1292</td>
<td>1850</td>
</tr>
<tr>
<td>3.4.8</td>
<td>N/A</td>
<td>N/A</td>
<td>070QA34</td>
<td>T31</td>
<td>X/X/none</td>
<td>U/UI/3613</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/1292</td>
<td>1850</td>
</tr>
<tr>
<td>3.4.9</td>
<td>N/A</td>
<td>N/A</td>
<td>070QA35</td>
<td>T31</td>
<td>X/X/none</td>
<td>U/UI/3613</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/1292</td>
<td>1850</td>
</tr>
<tr>
<td>3.4.10</td>
<td>070AD1</td>
<td>070AD2</td>
<td>T30</td>
<td>T30</td>
<td>X/X/none</td>
<td>U/UI/3613</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/1292</td>
<td>1933</td>
</tr>
<tr>
<td>3.4.11</td>
<td>070AD3</td>
<td>070AD4</td>
<td>T30</td>
<td>T30</td>
<td>X/X/none</td>
<td>U/UI/3613</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/1292</td>
<td>1920</td>
</tr>
<tr>
<td>3.4.12</td>
<td>070QA32A</td>
<td>070QA32B</td>
<td>T30</td>
<td>T30</td>
<td>X/X/none</td>
<td>U/UI/3613</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/1292</td>
<td>1920</td>
</tr>
<tr>
<td>3.4.13</td>
<td>070QA33</td>
<td>070QA34</td>
<td>T30</td>
<td>T30</td>
<td>X/X/none</td>
<td>U/UI/3613</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/1292</td>
<td>1920</td>
</tr>
<tr>
<td>3.4.14</td>
<td>N/A</td>
<td>N/A</td>
<td>070QA34</td>
<td>T30</td>
<td>X/X/none</td>
<td>U/UI/3613</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/9352</td>
<td>U/UI/1292</td>
<td>1920</td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
<td>File Name from MTASS CMS-2 Test Suite</td>
<td>Test Num</td>
<td>TRADA</td>
<td>APL</td>
<td>CCCC</td>
<td>CMS-2 SLOC w/ SYSDDDD &amp; QTCON</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------</td>
<td>--------------------------------------</td>
<td>----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single Precision Decl</td>
<td>3.4.13</td>
<td>070QA39</td>
<td>T32</td>
<td>U/U/U/ 4111</td>
<td>U/U/U/ 3575</td>
<td>C/C/C/ 9267</td>
<td>1693</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameter Passing Decl</td>
<td>3.4.14</td>
<td>070QA21C</td>
<td>T19</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 7453</td>
<td>U/U/U/ 13801</td>
<td>3310</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.4.14</td>
<td>070QA21D</td>
<td>T20</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 7460</td>
<td>U/U/U/ 13802</td>
<td>3307</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.4.14</td>
<td>070QA21R</td>
<td>T21</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 7459</td>
<td>U/U/U/ 13804</td>
<td>3308</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSWITCH Delete Decl</td>
<td>3.4.15</td>
<td>070QA43A</td>
<td>T34</td>
<td>C/C/C/ 4018</td>
<td>U/U/U/ 3914</td>
<td>C/C/C/ 9263</td>
<td>2220</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.4.15</td>
<td>070QA43B</td>
<td>T35</td>
<td>C/C/C/ 4068</td>
<td>U/U/U/ 4248</td>
<td>C/C/C/ 9282</td>
<td>2169</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spill Decl</td>
<td>3.4.16</td>
<td>070QA44A</td>
<td>T36</td>
<td>C/U/U/ 3646</td>
<td>U/U/U/ 3512</td>
<td>U/U/U/ 9166</td>
<td>1602</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.4.16</td>
<td>070QA44B</td>
<td>T37</td>
<td>C/U/U/ 3755</td>
<td>U/U/U/ 3616</td>
<td>U/U/U/ 9491</td>
<td>1651</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scaling Mode Decl</td>
<td>3.4.17</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Data Design</td>
<td>3.5.1</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Direct Code Block</td>
<td>3.5.2</td>
<td>070DC1</td>
<td>T76</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 4535</td>
<td>U/C/U/ 688</td>
<td>4431</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>070DCER1</td>
<td>T77</td>
<td>C/C/C/ 315</td>
<td>C/C/C/ 250</td>
<td>U/U/U/ 91</td>
<td>274</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>070DCER1</td>
<td>T77</td>
<td>C/C/C/ 315</td>
<td>C/C/C/ 250</td>
<td>U/U/U/ 91</td>
<td>274</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>070DCER1</td>
<td>T78</td>
<td>C/C/C/ 3678</td>
<td>U/U/U/ 3433</td>
<td>C/C/C/ 8912</td>
<td>1633</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>070DCER1</td>
<td>T79</td>
<td>C/C/C/ 3627</td>
<td>U/U/U/ 3445</td>
<td>C/C/C/ 8922</td>
<td>1614</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Table Decl</td>
<td>3.5.3</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
<td>File Name from MTASS CMS-2 Test Suite</td>
<td>Test Num</td>
<td>TRADA</td>
<td>APL</td>
<td>CCCC</td>
<td>CMS-2 SLOC w/ SYSDD&amp; QTCON</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------</td>
<td>--------------------------------------</td>
<td>----------</td>
<td>-------------</td>
<td>-----------</td>
<td>------------</td>
<td>---------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field Decl</td>
<td>3.5.3.1</td>
<td>07FQA10F2</td>
<td>T3</td>
<td>U/U/U/ 11411</td>
<td>U/U/U/ 8316</td>
<td>C/U/U/ 14018</td>
<td>5349</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA17</td>
<td>T14</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 7366</td>
<td>U/U/U/ 13760</td>
<td>3467</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA16</td>
<td>T13</td>
<td>U/U/U/ 8076</td>
<td>U/U/U/ 6389</td>
<td>U/U/U/ 12025</td>
<td>3496</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA14</td>
<td>T11</td>
<td>U/U/U/ 7167</td>
<td>U/U/U/ 7684</td>
<td>U/U/U/ 11910</td>
<td>3423</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Like-Table Decl</td>
<td>3.5.3.2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Item-Area Decl</td>
<td>3.5.3.2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sub-Table Decl</td>
<td>3.5.3.4</td>
<td>070QA19</td>
<td>T16</td>
<td>C/C/C/ 11593</td>
<td>U/U/U/ 7584</td>
<td>U/U/U/ 15551</td>
<td>5255</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field-Overlay Decl</td>
<td>3.5.3.5</td>
<td>070QA3</td>
<td>T26</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 4896</td>
<td>U/U/U/ 11333</td>
<td>2519</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modifier</td>
<td>3.5.4</td>
<td>070QA53</td>
<td>T39</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 3885</td>
<td>U/U/U/ 9838</td>
<td>1864</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>External Program Decl</td>
<td>3.5.5</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variable Decl</td>
<td>3.5.6</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Procedure Sw Bl</td>
<td>3.5.7</td>
<td>070QA5</td>
<td>T38</td>
<td>U/U/U/ 4618</td>
<td>U/U/U/ 4216</td>
<td>U/U/U/ 9850</td>
<td>1993</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PINDEX Decl</td>
<td>3.5.7.1</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PITEM Switch Block</td>
<td>3.5.7.2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PDOUBLE Switch</td>
<td>3.5.7.3</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameter Decl</td>
<td>3.5.8</td>
<td>070QA2</td>
<td>T17</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 7275</td>
<td>U/U/U/ 12616</td>
<td>3618</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B-28
<table>
<thead>
<tr>
<th>Test Description</th>
<th>MTASS CMS-2 User Handbook Section</th>
<th>File Name from MTASS CMS-2 Test Suite</th>
<th>Test Num</th>
<th>TRADA</th>
<th>APL</th>
<th>CCCC</th>
<th>CMS-2 SLOC w/ SYSDD&amp; QTC0N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overlay Decl</td>
<td>3.5.9</td>
<td>070QA3</td>
<td>T26</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 4896</td>
<td>U/U/U/ 11333</td>
<td>2519</td>
</tr>
<tr>
<td></td>
<td>3.5.9</td>
<td>070QA60</td>
<td>T58</td>
<td>U/U/U/ 5838</td>
<td>U/U/U/ 5038</td>
<td>U/U/U/ 16954</td>
<td>2562</td>
</tr>
<tr>
<td>Data Statement</td>
<td>3.5.10</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Range Decl</td>
<td>3.5.11</td>
<td>070QA95</td>
<td>T73</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 5183</td>
<td>U/U/U/ 9902</td>
<td>2020</td>
</tr>
<tr>
<td>Format Decl</td>
<td>3.5.12</td>
<td>070QA8</td>
<td>T62</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 4535</td>
<td>U/U/U/ 11648</td>
<td>2201</td>
</tr>
<tr>
<td>Nonstandard File Decl</td>
<td>3.5.13.1</td>
<td>070QA64</td>
<td>T59</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 4795</td>
<td>U/U/U/ 11219</td>
<td>2454</td>
</tr>
<tr>
<td>Standard File Decl</td>
<td>3.5.13.2</td>
<td>070QA8</td>
<td>T62</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 4535</td>
<td>U/U/U/ 11648</td>
<td>2201</td>
</tr>
<tr>
<td>Stringform Decl</td>
<td>3.5.14</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inputlist Decl</td>
<td>3.5.15</td>
<td>070QA18</td>
<td>T15</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 9317</td>
<td>U/U/U/ 15664</td>
<td>5617</td>
</tr>
<tr>
<td>Outpilstlist Decl</td>
<td>3.5.16</td>
<td>070QA18</td>
<td>T15</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 9317</td>
<td>U/U/U/ 15664</td>
<td>5617</td>
</tr>
<tr>
<td>System Procedure Stmt</td>
<td>3.6.1</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local Data Design</td>
<td>3.6.2.1</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index Switch Block</td>
<td>3.6.2.1.1</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Item Switch Block</td>
<td>3.6.2.1.2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double Switch Block</td>
<td>3.6.2.1.3</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local Program Decl</td>
<td>3.6.2.1.4</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
<td>File Name from MTASS CMS-2 Test Suite</td>
<td>Test Num</td>
<td>TRADA</td>
<td>APL</td>
<td>CCCC</td>
<td>CMS-2 SLOC w/ SYSDD&amp; QTCON</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------</td>
<td>---------------------------------------</td>
<td>----------</td>
<td>--------</td>
<td>------</td>
<td>------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Auto Data Design</td>
<td>3.6.2.2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Procedure Block Decl</td>
<td>3.6.3.1</td>
<td>070QA7A</td>
<td>T60</td>
<td>U/U/U</td>
<td>U/U/</td>
<td>X/X/ none</td>
<td>5450</td>
</tr>
<tr>
<td>Function Block Decl</td>
<td>3.6.3.2</td>
<td>070QA7A</td>
<td>T60</td>
<td>U/U/U</td>
<td>U/U/</td>
<td>X/X/ none</td>
<td>5450</td>
</tr>
<tr>
<td>Exec Proc Block Decl</td>
<td>3.6.3.3</td>
<td>070QA22</td>
<td>T22</td>
<td>U/U/U</td>
<td>U/U/</td>
<td>4601</td>
<td>U/U/U/ 10089</td>
</tr>
<tr>
<td>Local Index List</td>
<td>3.7.1.1</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subprogram Data Design</td>
<td>3.7.1.2</td>
<td>070QA542</td>
<td>T52</td>
<td>X/X/X</td>
<td>U/U/</td>
<td>5355</td>
<td>U/U/U/ 12966</td>
</tr>
<tr>
<td>Imperative Stmt</td>
<td>3.7.2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set Phrase</td>
<td>3.7.3.1</td>
<td>070QA10</td>
<td>T2</td>
<td>U/U/U</td>
<td>U/U/</td>
<td>10951</td>
<td>C/U/U/ 13249</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA14</td>
<td>T11</td>
<td>U/U/U</td>
<td>U/U/</td>
<td>7167</td>
<td>U/U/U/ 11910</td>
</tr>
<tr>
<td>Begin Phrase</td>
<td>3.7.3.2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return Phrase</td>
<td>3.7.3.3</td>
<td>070QA82A</td>
<td>T63</td>
<td>C/C/C</td>
<td>U/U/</td>
<td>4015</td>
<td>U/U/U/ 8994</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA82B</td>
<td>T64</td>
<td>C/C/C</td>
<td>U/U/</td>
<td>3648</td>
<td>U/U/U/ 8998</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA82C</td>
<td>T65</td>
<td>X/X/X</td>
<td>U/U/</td>
<td>3245</td>
<td>U/U/U/ 9030</td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
<td>File Name from MTASS CMS-2 Test Suite</td>
<td>Test Num</td>
<td>TRADA</td>
<td>APL</td>
<td>CCCC</td>
<td>CMS-2 SLOC w/ SYSDD&amp; QTCN</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------</td>
<td>--------------------------------------</td>
<td>----------</td>
<td>-------</td>
<td>-------</td>
<td>-----------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Simple Goto Phrase</td>
<td>3.7.3.4.1</td>
<td>070QA83A</td>
<td>T66</td>
<td>C/C/C/ 3607</td>
<td>U/U/U/ 3117</td>
<td>U/U/U/ 8923</td>
<td>1590</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA83B</td>
<td>T67</td>
<td>C/C/C/ 3608</td>
<td>U/U/U/ 3118</td>
<td>U/U/U/ 8924</td>
<td>1591</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA83C</td>
<td>T68</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 3242</td>
<td>U/U/U/ 9053</td>
<td>1655</td>
</tr>
<tr>
<td>Index Goto Phrase</td>
<td>3.7.3.4.2</td>
<td>070QA4</td>
<td>T33</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 4640</td>
<td>U/U/U/ 9834</td>
<td>2885</td>
</tr>
<tr>
<td>Item Goto Phrase</td>
<td>3.7.3.4.3</td>
<td>070QA4</td>
<td>T33</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 4640</td>
<td>U/U/U/ 9834</td>
<td>2885</td>
</tr>
<tr>
<td>User Proc Call Phrase</td>
<td>3.7.3.5.1</td>
<td>070QA7B</td>
<td>T61</td>
<td>U/U/U/ 17738</td>
<td>U/U/U/13013</td>
<td>U/U/U/17462</td>
<td>6146</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA86</td>
<td>T69</td>
<td>C/C/C/ 4545</td>
<td>U/U/U/ 3702</td>
<td>U/U/U/ 10724</td>
<td>1982</td>
</tr>
<tr>
<td>Supplied Proc Call Phrase</td>
<td>3.7.3.5.2</td>
<td>070QA538</td>
<td>T40</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 6976</td>
<td>U/U/U/ 12246</td>
<td>4541</td>
</tr>
<tr>
<td>PINDEX Switch Call</td>
<td>3.7.3.6.1</td>
<td>070QA4</td>
<td>T33</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 4640</td>
<td>U/U/U/ 9834</td>
<td>2885</td>
</tr>
<tr>
<td>PITEM Call Phrase</td>
<td>3.7.3.6.2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vary Block</td>
<td>3.7.3.7</td>
<td>070QA6</td>
<td>T57</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 8340</td>
<td>U/U/U/ 14626</td>
<td>4837</td>
</tr>
<tr>
<td>Stop Phrase</td>
<td>3.7.3.8</td>
<td>070QA90</td>
<td>T71</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 3266</td>
<td>U/U/U/ 9058</td>
<td>1677</td>
</tr>
<tr>
<td>Resume Phrase</td>
<td>3.7.3.9</td>
<td>070QA6</td>
<td>T57</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 8340</td>
<td>U/U/U/ 14626</td>
<td>4837</td>
</tr>
<tr>
<td>For Block</td>
<td>3.7.3.10</td>
<td>070QA6</td>
<td>T57</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 8340</td>
<td>U/U/U/ 14626</td>
<td>4837</td>
</tr>
<tr>
<td>Exec Phrase</td>
<td>3.7.3.11</td>
<td>070QA23</td>
<td>T23</td>
<td>U/U/U/ 5652</td>
<td>U/U/U/ 5030</td>
<td>U/U/U/ 9895</td>
<td>2446</td>
</tr>
<tr>
<td>Shift Phrase</td>
<td>3.7.3.12</td>
<td>070QA2</td>
<td>T17</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 7275</td>
<td>U/U/U/ 12616</td>
<td>3618</td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
<td>File Name from MTASS CMS-2 Test Suite</td>
<td>Test Num</td>
<td>TRADA</td>
<td>APL</td>
<td>CCCC</td>
<td>CMS-2 SLOC w/ SYSDD&amp; QTCN</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------</td>
<td>--------------------------------------</td>
<td>----------</td>
<td>-------------</td>
<td>-------------</td>
<td>--------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Display Phrase</td>
<td>3.7.3.13.1</td>
<td>070QA95</td>
<td>T73</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 5183</td>
<td>U/U/U/ 9902</td>
<td>2020</td>
</tr>
<tr>
<td>Snap Phrase</td>
<td>3.7.3.13.2</td>
<td>070QA95</td>
<td>T73</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 5183</td>
<td>U/U/U/ 9902</td>
<td>2020</td>
</tr>
<tr>
<td>Trace Phrase</td>
<td>3.7.3.13.3</td>
<td>070QA95</td>
<td>T73</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 5183</td>
<td>U/U/U/ 9902</td>
<td>2020</td>
</tr>
<tr>
<td>End Trace Phase</td>
<td>3.7.3.13.4</td>
<td>070QA95</td>
<td>T73</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 5183</td>
<td>U/U/U/ 9902</td>
<td>2020</td>
</tr>
<tr>
<td>Swap Phrase</td>
<td>3.7.3.14</td>
<td>070QA2</td>
<td>T17</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 7275</td>
<td>U/U/U/ 12616</td>
<td>3618</td>
</tr>
<tr>
<td></td>
<td>3.7.3.14</td>
<td>070QA15</td>
<td>T12</td>
<td>U/U/U/ 14714</td>
<td>U/U/U/ 9089</td>
<td>U/U/U/ 13018</td>
<td>4901</td>
</tr>
<tr>
<td>Pack Phrase</td>
<td>3.7.3.15</td>
<td>070QA2</td>
<td>T17</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 7275</td>
<td>U/U/U/ 12616</td>
<td>3618</td>
</tr>
<tr>
<td>Open Phrase</td>
<td>3.7.3.16.1</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Close Phrase</td>
<td>3.7.3.16.2</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Endfile Phrase</td>
<td>3.7.3.16.3</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEFID Phrase</td>
<td>3.7.3.16.4</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHKID Phrase</td>
<td>3.7.3.16.5</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIL POS Phrase</td>
<td>3.7.3.16.6</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SET POS Phrase</td>
<td>3.7.3.16.7</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output to the Printer</td>
<td>3.7.3.16.8.1</td>
<td>070QA18</td>
<td>T15</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 9317</td>
<td>U/U/U/ 15664</td>
<td>5617</td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
<td>File Name from MTASS CMS-2 Test Suite</td>
<td>Test Num</td>
<td>TRADA</td>
<td>APL</td>
<td>CCCC</td>
<td>CMS-2 SLOC w/ SYSDD&amp;D QTCN</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------</td>
<td>--------------------------------------</td>
<td>----------</td>
<td>---------------</td>
<td>--------------</td>
<td>-------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Input Phrase</td>
<td>3.7.3.16.9</td>
<td>070QA8</td>
<td>T62</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 4535</td>
<td>U/U/U/ 11648</td>
<td>2201</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA91</td>
<td>T72</td>
<td>C/C/C/ 4140</td>
<td>U/U/U/ 3367</td>
<td>U/U/U/ 9288</td>
<td>1792</td>
</tr>
<tr>
<td>Encode Phrase</td>
<td>3.7.3.16.10</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decode Phrase</td>
<td>3.7.3.16.11</td>
<td>070QA8</td>
<td>T62</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 4535</td>
<td>U/U/U/ 11648</td>
<td>2201</td>
</tr>
<tr>
<td>Format Scan</td>
<td>3.7.3.16.12</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Null Phrase</td>
<td>3.7.3.17</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exit Phrase</td>
<td>3.7.3.18</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>If Clause</td>
<td>3.7.4.1</td>
<td>070QA2</td>
<td>T17</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 7275</td>
<td>U/U/U/ 12616</td>
<td>3618</td>
</tr>
<tr>
<td>Else Clause</td>
<td>3.7.4.2</td>
<td>070QA2</td>
<td>T17</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 7275</td>
<td>U/U/U/ 12616</td>
<td>3618</td>
</tr>
<tr>
<td>Find Clause</td>
<td>3.7.4.3</td>
<td>070QA2</td>
<td>T17</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 7275</td>
<td>U/U/U/ 12616</td>
<td>3618</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA6</td>
<td>T57</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 8340</td>
<td>U/U/U/ 14626</td>
<td>4837</td>
</tr>
<tr>
<td>Predefined Function Call</td>
<td>3.8.2.1 - 3.8.2.22</td>
<td>070QA2</td>
<td>T17</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 7275</td>
<td>U/U/U/ 12616</td>
<td>3618</td>
</tr>
<tr>
<td></td>
<td></td>
<td>070QA12</td>
<td>T7</td>
<td>U/U/U/ 11858</td>
<td>U/U/U/ 8489</td>
<td>U/U/U/ 13765</td>
<td>4384</td>
</tr>
</tbody>
</table>

B-33
<table>
<thead>
<tr>
<th>Test Description</th>
<th>MTASS CMS-2 User Handbook Section</th>
<th>File Name from MTASS CMS-2 Test Suite</th>
<th>Test Num</th>
<th>TRADA</th>
<th>APL</th>
<th>CCCC</th>
<th>CMS-2 SLOC w/ SYSDD &amp; QTCON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predefined Function Call (continued)</td>
<td>3.8.2.1-22</td>
<td>070QA11A</td>
<td>T4</td>
<td>U/U/U/ 17417</td>
<td>U/U/U/ 10817</td>
<td>U/U/U/ 16486</td>
<td>6341</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA11B</td>
<td>T5</td>
<td>U/U/U/ 17413</td>
<td>U/U/U/ 10815</td>
<td>U/U/U/ 16471</td>
<td>6335</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1 - 3.8.2.22</td>
<td>070QA13A</td>
<td>T8</td>
<td>U/U/U/ 17187</td>
<td>U/U/U/ 11198</td>
<td>U/U/U/ 16649</td>
<td>6451</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA1</td>
<td>T1</td>
<td>U/U/U/ 5723</td>
<td>U/U/U/ 1876</td>
<td>U/U/U/ 10130</td>
<td>2170</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA538</td>
<td>T40</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 6976</td>
<td>U/U/U/ 12246</td>
<td>4541</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA538C</td>
<td>T43</td>
<td>U/U/U/ 5593</td>
<td>U/U/U/ 4507</td>
<td>U/U/U/ 10386</td>
<td>2468</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA538D</td>
<td>T44</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 4382</td>
<td>U/U/U/ 10390</td>
<td>2522</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA538E</td>
<td>T45</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 4408</td>
<td>U/U/U/ 10273</td>
<td>2538</td>
</tr>
<tr>
<td>Test Description</td>
<td>MTASS CMS-2 User Handbook Section</td>
<td>File Name from MTASS CMS-2 Test Suite</td>
<td>Test Num</td>
<td>TRADA</td>
<td>APL</td>
<td>CCCC</td>
<td>CMS-2 SLOC w/ SYSDD&amp; QTCON</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------</td>
<td>--------------------------------------</td>
<td>----------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Predefined Function Call (continued)</td>
<td>3.8.2.1-22</td>
<td>070QA582</td>
<td>T53</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 4319</td>
<td>U/U/U/ 9650</td>
<td>2138</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>07FQA582F1</td>
<td>T54</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 4324</td>
<td>U/U/U/ 9635</td>
<td>2146</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>07FQA582F2</td>
<td>T55</td>
<td>X/X/X/ none</td>
<td>U/U/U/ 4319</td>
<td>U/U/U/ 9605</td>
<td>2222</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA539</td>
<td>T46</td>
<td>C/C/C/ 6489</td>
<td>U/U/U/ 6022</td>
<td>U/U/U/ 11463</td>
<td>3603</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA539A</td>
<td>T47</td>
<td>C/C/C/ 4381</td>
<td>U/U/U/ 3084</td>
<td>U/U/U/ 9904</td>
<td>2054</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA539B</td>
<td>T48</td>
<td>C/C/C/ 4398</td>
<td>U/U/U/ 3938</td>
<td>C/U/C/ 9922</td>
<td>2019</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA539C</td>
<td>T49</td>
<td>C/C/C/ 4858</td>
<td>U/U/U/ 4165</td>
<td>C/U/U/ 10185</td>
<td>2238</td>
</tr>
<tr>
<td></td>
<td>3.8.2.1-22</td>
<td>070QA539D</td>
<td>T50</td>
<td>C/C/C/ 5002</td>
<td>U/U/U/ 4414</td>
<td>C/U/U/ 10252</td>
<td>2410</td>
</tr>
<tr>
<td>Test Description</td>
<td>TRADA</td>
<td>APL</td>
<td>CCCC</td>
<td>CMS-2 SLOC w/ SYSDD &amp; QTCON</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>--------------</td>
<td>---------------</td>
<td>-----------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>385.0K / 598.9K¹</td>
<td>468.3K / 468.9K²</td>
<td>923.7K / 925.7K²</td>
<td>242.6K</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL LINES of CODE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAX CORRECT</td>
<td>24 of 84</td>
<td>1 of 84</td>
<td>14 of 84</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPILATION %</td>
<td>compilable – 29%</td>
<td>compilable – 1%</td>
<td>compilable – 17 %</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(24 of 54</td>
<td></td>
<td>(14 of 83</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>compilable – 44 %³</td>
<td></td>
<td>compilable – 17 %³</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sun CORRECT</td>
<td>24 of 54</td>
<td>1 of 84</td>
<td>10 of 83</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPILATION %</td>
<td>compilable – 44%</td>
<td>compilable – 1%</td>
<td>compilable – 12%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNAT CORRECT</td>
<td>22 of 54</td>
<td>1 of 84</td>
<td>10 of 83</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPILATION %</td>
<td>compilable – 41%</td>
<td>compilable – 1%</td>
<td>compilable – 12%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

¹ Estimated SLOC assuming all files translated
² For these numbers, the predefined packages are counted once.
³ Percentage compilable based on actual number of files produced by translator.
<table>
<thead>
<tr>
<th>Project and CMS-2 Lines of Code (SLOC)</th>
<th>TRADA Translator on VAX/VMS</th>
<th>APL Translator on Sun/OS</th>
<th>CCCC Transformer on VAX/VMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extra Low Frequency Communications (ELF)</td>
<td>P (6 minutes)</td>
<td>P (1 minute)</td>
<td>P (14 minutes)</td>
</tr>
<tr>
<td>9,988 SLOC</td>
<td>U/U/U/ 17,870 SLOC</td>
<td>U/U/U/ 3,534 SLOC</td>
<td>U/U/U/ 11,226 SLOC</td>
</tr>
<tr>
<td></td>
<td>Two parameters on OPTIONS statement, TAPE and ASM, were unrecognizable by TRADA and had to be removed to continue</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The errors for all three compilers for all three ELF translations resulted mainly from Use clauses for missing Ada package AMTO02, whose CMS-2 source code was not available.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combat Control System MK-2 Fire Control System</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>421 SLOC</td>
<td>F (1 minute — aborted)</td>
<td>P (1 minute)</td>
<td>P (2 minutes)</td>
</tr>
<tr>
<td></td>
<td>X/X/X/ no Ada generated</td>
<td>U/U/U/ 370 SLOC</td>
<td>C/U/U/ 936 SLOC</td>
</tr>
<tr>
<td></td>
<td>TRADA constraint error</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>An END-HEAD statement was added to the MK2 major header to provide syntactic correctness.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Compile Results**

- **C**
  - Correctly compiled of generated Ada code
- **U**
  - Unsuccessful Ada compile — errors present
- **X**
  - No Ada code generated by translator — no compile possible
<table>
<thead>
<tr>
<th>Project and CMS-2 Lines of Code (SLOC)</th>
<th>TRADA Translator on VAX/VMS</th>
<th>APL Translator on Sun/OS</th>
<th>CCCC Transformer on VAX/VMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>S3-Aircraft Tactical Mission Program (TMP)</td>
<td>Q (1 minute) X/X/X/ no Ada generated</td>
<td>P (1 minute) U/U/U 1,183 SLOC</td>
<td>P (7 minutes) U/U/U 4,148 SLOC</td>
</tr>
<tr>
<td>1,391 SLOC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRADA reported 279 occurrences of missing identifiers, and terminated</td>
<td></td>
<td>An incomplete CMS-2 compile time system was built from the S3-TMP code pieces provided in order to attempt translation. The code was included into one SYS-DD and two SYS-PROC-RENs. More code would be needed for a viable translation.</td>
<td></td>
</tr>
<tr>
<td>AEGIS SPY UYK-43 Timing Loop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2,841 SLOC</td>
<td>P (3 minutes) C/C/C/ 3,965 SLOC</td>
<td>P (3 minutes) U/U/U 2,447 SLOC</td>
<td>P (5 minutes) U/U/U 3,640 SLOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The two SYS-PROCs provided were combined into one CMS-2 compile time system for input to the translators.</td>
</tr>
<tr>
<td>Project and CMS-2 Lines of Code (SLOC)</td>
<td>TRADA Translator on VAX/VMS</td>
<td>APL Translator on Sun/OS</td>
<td>CCCC Transformer on VAX/VMS</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-----------------------------</td>
<td>--------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>H60B Helo DataLink Upgrade-ACASS module</td>
<td>P (4 minutes)</td>
<td>P (1 minute)</td>
<td>P (11 minutes)</td>
</tr>
<tr>
<td>4,725 SLOC</td>
<td>U/U/U/ 6,534 SLOC</td>
<td>U/U/U/ 1,448 SLOC</td>
<td>U/U/U/ 5,987 SLOC</td>
</tr>
</tbody>
</table>

All ACASS multiple nested includes (MTASS/M form) were combined into a single CMS-2 compile time system for input to the translators.
CONCLUSIONS

1. All translators had catastrophic failures during stress testing. The developers were very responsive in fixing these translator deficiencies with an average turnaround of two working days.

2. Most source code produced by the translators did not compile correctly without manual changes.

3. When using the translators a project will see an increase in the ratio of Ada to CMS-2 SLOC counts from approximately 2:1 to 4:1 depending on the translator selected and the CMS-2 constructs being translated (See Table B-2, 14). The code expansion is due not only because of differences in the two languages but also because during translation blank lines are inserted for readability, in some cases error messages are generated as comments, and predefined packages are produced.

4. Only the CCCC translator translated overlay. The correct execution of the translated overlays was not verified.
APPENDIX C: RESULTS OF REENGINEER UNTIL ADA CODE EXECUTES CORRECTLY

OVERVIEW

This section presents results of the Reengineer Until Ada Code Executes Correctly phase of the evaluation. Versions of the translators used were the developers final revisions delivered after problems causing translator failure were corrected. In this phase, the effort to take a CMS-2-based program from translation to correct functional execution of the generated Ada version was measured for each of the translators. These data were recorded as person-hours devoted to each stage of the process and number of source lines of code added and modified.

It was noted that there was no baseline against which to compare the properties of translated code and the effort required to reengineer it to execute correctly. A decision was made to generate such a baseline and the resulting metrics were included with those of the translator-generated code. The Reengineer Until Ada Code Executes Correctly phase of evaluation constitutes a small case study of CMS-2 to Ada translation. The metrics obtained will assist CMS-2 project managers in generating cost and schedule estimates for using automated CMS-2 to Ada translation.

The initial phase, Conduct Quick Look Inspection Using Small CMS-2 Sample, paved the way for execution testing described in this appendix. Under Quick Look Inspection QA9 CMS-2 source code was compiled, linked with a test harness, and executed to provide baseline execution results. Then QA9 CMS-2 was translated by each translator and the generated Ada was repeatedly submitted to the Ada compilers and reworked until it compiled.

Translator evaluation continued in the Reengineer Until Ada Code Executes Correctly phase. The Ada QA9 source code was compiled, linked with the test harness, and executed. The Ada harness was produced by reengineering the CMS-2 test harness, translating, and reengineering in Ada. The Ada generated for QA9 was reengineered until execution produced results at least as accurate as the CMS-2 execution results.

The QA9 program was taken from CMS-2 translation to correct execution in Ada for the seven combinations of translators and compilers listed below. The APL and CCCC QA9 translations were not taken to correct execution when compiled with VAX Ada due to a lack of time.

1. APL translation compiled with GNAT,
2. APL translation compiled with SunAda,
3. CCCC translation compiled with GNAT,
4. CCCC translation compiled with SunAda,
5. TRADA translation compiled with GNAT,
6. TRADA translation compiled with SunAda, and
7. TRADA translation compiled with VAX Ada.

The QA9 program contains self checking arithmetic tests that compare computed with expected results. Informational messages are printed when results do not match and summary information is
printed at the conclusion of program execution. Translators bracketed QA9 harness related direct code inside Ada comments. No direct code was required for execution.

This appendix presents a high-level summary of the results of this phase. The section is intended for managers considering translation as an aid to program generation.

Results include:
- Tables that show quantity of source lines of code at different stages of the reengineering process
- Table that indicates the difficulty in conversion as measured by person-hours
- Table that indicates difficulty in conversion as measured by Ada source code modifications required to achieve correct execution
- Discussion of redesign/rewrite of QA9 in Ada 95
- Tables that compare weighted McCabe cyclomatic complexity and program size for CMS-2 and Ada versions of QA9.

Appendix F is a log containing details of the steps followed to achieve correct execution in Ada. The intended audience is software engineers considering translation as a code generation method. Appendix F includes a description of the source code corrections made for compilation and correct execution.

LINE COUNT COMPARISONS

Table C-1 contains line counts for QA9 as translated, compiled, and executed by the APL, CCCC, and TRADA translators. Line counts include the predefined utilities which were produced or provided by the translators and are required by all translated programs. The second row from the bottom shows the line count for Ada 95 QA9, the redeveloped equivalent to QA9. There was a substantial reduction in the number of lines of source code for Ada 95 QA9. QA9 CMS-2 line counts are included for comparison purposes.

Table C-2 shows the line counts for the predefined utilities for QA9. The predefined utilities are Ada packages that contain type declarations and functions used by the translated code. These line counts are constant for all translations when using the APL and CCCC translators. The counts are different for TRADA, since only what was required was produced.
Table C-1. QA9 Source Lines of Code by Translator at Various Stages (Includes Predefined)-1

<table>
<thead>
<tr>
<th>QA9 Translated by APL</th>
<th>Delimiting semicolons</th>
<th>Comments</th>
<th>Statements of text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translated</td>
<td>4650</td>
<td>5855</td>
<td>7570</td>
</tr>
<tr>
<td>Compilation with GNAT(^1)</td>
<td>4856</td>
<td>6061</td>
<td>7776</td>
</tr>
<tr>
<td>Compilation with Sun Ada(^1)</td>
<td>4856</td>
<td>6061</td>
<td>7776</td>
</tr>
<tr>
<td>Correct execution GNAT</td>
<td>4875</td>
<td>6484</td>
<td>8496</td>
</tr>
<tr>
<td>Correct execution Sun Ada</td>
<td>4874</td>
<td>6487</td>
<td>8498</td>
</tr>
<tr>
<td>QA9 Translated by CCCC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Translated</td>
<td>9632</td>
<td>1667</td>
<td>15657</td>
</tr>
<tr>
<td>Compilation with GNAT</td>
<td>9634</td>
<td>1669</td>
<td>15660</td>
</tr>
<tr>
<td>Compilation with Sun Ada(^2)</td>
<td>9660</td>
<td>1675</td>
<td>15720</td>
</tr>
<tr>
<td>Compilation with VAX Ada</td>
<td>9631</td>
<td>1661</td>
<td>15653</td>
</tr>
<tr>
<td>Correct execution GNAT(^3)</td>
<td>9653</td>
<td>1675</td>
<td>15712</td>
</tr>
<tr>
<td>Correct execution Sun Ada(^2)</td>
<td>9660</td>
<td>1675</td>
<td>15720</td>
</tr>
<tr>
<td>QA9 Translated by TRADA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Translated</td>
<td>4725</td>
<td>2700</td>
<td>10227</td>
</tr>
<tr>
<td>Compilation with GNAT</td>
<td>4726</td>
<td>2719</td>
<td>10245</td>
</tr>
<tr>
<td>Compilation with Sun Ada</td>
<td>4726</td>
<td>2719</td>
<td>10245</td>
</tr>
<tr>
<td>Compilation with VAX Ada</td>
<td>4952</td>
<td>2866</td>
<td>10378</td>
</tr>
<tr>
<td>Correct execution GNAT</td>
<td>4948</td>
<td>3388</td>
<td>11348</td>
</tr>
<tr>
<td>Correct execution Sun Ada</td>
<td>4948</td>
<td>3388</td>
<td>11348</td>
</tr>
<tr>
<td>Correct execution VAX Ada</td>
<td>4952</td>
<td>2866</td>
<td>10245</td>
</tr>
<tr>
<td>QA9 Redesigned &amp; Rewritten in Ada 95(^4)</td>
<td>1675</td>
<td>438</td>
<td>5879</td>
</tr>
<tr>
<td>QA9 CMS-2</td>
<td>3568</td>
<td>785</td>
<td>4326</td>
</tr>
</tbody>
</table>

\(^1\) Estimated counts because actual numbers were not kept

\(^2\) Includes modifications due to Sun Ada compiler bug (3 delimiting ; & 2 text statements)

\(^3\) Includes statements for debugging purposes (17 delimiting ; & 33 text statements)

\(^4\) Because of the design and evolution of this test code, great improvements could be made in code efficiency. Reengineering of most legacy code is likely to result in substantial improvements, but perhaps not as dramatic as achieved here.
Table C-2. QA9 Predefined Utilities Source Lines of Code by Translator

<table>
<thead>
<tr>
<th></th>
<th>Delimiting semicolons</th>
<th>Comments</th>
<th>Statements of text</th>
</tr>
</thead>
<tbody>
<tr>
<td>APL (BASIC_DEFNS)</td>
<td>317</td>
<td>165</td>
<td>642</td>
</tr>
<tr>
<td>CCCC (PREDEFINEDS)</td>
<td>1203</td>
<td>432</td>
<td>2022</td>
</tr>
<tr>
<td>TRADA (CMS-2 TYPES)</td>
<td>225</td>
<td>29</td>
<td>459</td>
</tr>
</tbody>
</table>

DIFFICULTY OF CONVERSION METRICS

Table C-3 shows the Difficulty of Conversion Hours metric for the APL, CCCC, and TRADA translators. For each translator QA9 was taken from generation to correct execution using the compilers indicated in this table. Difficulty of Conversion Hours is the sum of person-hours spent to achieve compilation plus person-hours spent to achieve correct execution.

The authors had to decide whether to perform the conversion for each compiler from the original translated code or to take the product of conversion using one compiler as input into the process of conversion by the other. The thoroughness of the Ada standard makes it likely that a program compiled by one compiler will compile with little or no modification by another. Following the first approach would mean that the learning that would have taken place during conversion using one compiler would shorten the time taken in the conversion process for another. This is because most of the required corrections for the second conversion effort would be known ahead of time. Following the second approach would mean that the second conversion would measure only the incremental effort to get a correctly executing program to compile and execute using another compiler. Since the first approach would be biased and would require duplicate effort, the second approach using SLOC was followed.

Table C-4 shows the Difficulty of Conversion SLOC metric for the three translators. The method used for computing SLOC and some problems involved in comparing SLOC metrics are described in appendix D. The issue of how to count lines of code that are moved from one location to another was resolved as counting each line moved as one change.

The APL translator had numerous Ada syntax and semantic errors. The most common error encountered was with APL producing Ada code that contained floating point exponents. Type casting these exponents to integer solved those problems but upon running QA9, 82 execution errors were reported similar to the TRADA translator. This was because Ada 83 does not have sufficient precision to pass the exponentiation test suite. The program was modified using Ada 95 which can handle floating point exponents so later executions reported no errors.
Table C-3. QA9 Difficulty of Conversion Person Hours

<table>
<thead>
<tr>
<th></th>
<th>Hours to achieve compilation</th>
<th>Hours to achieve correct execution</th>
<th>Difficulty of Conversion Hours (Total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>APL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNAT</td>
<td>9</td>
<td>18</td>
<td>27</td>
</tr>
<tr>
<td>Sun Ada</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CCCC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNAT</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Sun Ada</td>
<td>1</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>TRADA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNAT</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sun Ada</td>
<td>1</td>
<td>0 (6)(^1)</td>
<td>1 (7)(^1)</td>
</tr>
<tr>
<td>VAX Ada</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

The CCCC translator assumed the existence of package “Math_Lib” which was presumed to contain the appropriate exponentiation operator, but “Math_Lib” was not contained in the generated code. Therefore, access to the an appropriate mathematical library was sufficient to remedy that problem. The APL translator also relied on the existence of an exponentiation operator for a floating point exponent but did not provide the operator. Although both the CCCC and APL implementations were incomplete with respect to exponentiation, the assumption of a different exponentiation operator, and the consequent difference in execution behavior is not incorrect.

The CCCC-generated code also presented an access-before-elaboration problem (see Section 5, Recommendations to Translator Vendors) which was relatively difficult to analyze and represents the majority of time consumed in converting the CCCC code.

Table C-4 indirectly reflects a ambiguity in the definition of “correct execution.” The modifications made to the TRADA-generated code to achieve execution with no errors reported by the executing program were of two kinds. The first kind of modification was made to achieve compilation on Sun SPARC platforms. Sun SPARC apparently does not support the specification of a floating point type that was presumably supported on the CMS-2 targeted platform. The modification was not required for execution on DEC VAXes and was not one of having generated incorrect code. It was a portability problem. The second kind of modification was made because TRADA generated code that only used Ada 83 standard mathematical functions. The QA9 test suite was designed to detect errors in mathematical precision. Therefore, TRADA-generated code executed correctly when it reported 82 execution errors because it correctly indicated that the Ada 83

\(^1\) The number in parenthesis is the time required to fully implement exponentiation with a floating point exponent. These additional hours would not be required for conversion to Ada 95.

C-5
does not have sufficient precision to pass the exponentiation test suite. One can legitimately state that the TRADA code was correct "as generated" and was also the most portable of the three generated samples. Nevertheless, the program was modified to the point that when executed, it reported no errors. Those difficulty of conversion data appear in parentheses in tables C-3 and C-4.

Access to an exponentiation operator for a floating point exponent was required for the TRADA-generated code to achieve execution with no reported errors. This required 98 SLOC modifications and was made by accessing package Ada.Numerics.Generic_Elementary_Functions for GNAT compilation and by accessing the Sun Ada standard math library for the Sun Ada compiler.

The difficulty of conversion metrics, while meaningful, cannot simply be extrapolated on the basis of SLOC to achieve a level-of-effort estimate for a legacy system. QA9, including harness, contained no direct code or low-level operations necessary for execution, and was selected for this study because its translation was thought to be feasible. It also has relatively simple requirements. As a result, it is probably not representative of many legacy systems.

### Table C-4. QA9 Difficulty of Conversion SLOC

<table>
<thead>
<tr>
<th></th>
<th>SLOC added or modified for compile</th>
<th>SLOC added or modified for correct execution</th>
<th>Difficulty of Conversion SLOC (Total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>APL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNAT</td>
<td>206</td>
<td>225</td>
<td>431</td>
</tr>
<tr>
<td>Sun Ada</td>
<td>206</td>
<td>224</td>
<td>430</td>
</tr>
<tr>
<td>CCCC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNAT</td>
<td>2</td>
<td>28(^1)</td>
<td>30</td>
</tr>
<tr>
<td>Sun Ada</td>
<td>9(^2)</td>
<td>28(^1)</td>
<td>37</td>
</tr>
<tr>
<td>TRADA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNAT</td>
<td>6</td>
<td>0 (98)^3</td>
<td>6 (104)^3</td>
</tr>
<tr>
<td>Sun Ada</td>
<td>4</td>
<td>0 (98)^3</td>
<td>4 (102)^3</td>
</tr>
<tr>
<td>VAX Ada</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**WEIGHTED MCCABE AND PROGRAM SIZE METRICS**

Table C-5 shows the weighted McCabe cyclomatic complexity \(((\Sigma_i=1..n(SLOC_i*V(G)_i))/(\Sigma_i=1..n(SLOC_i)))\) for the CMS-2 QA9 and the translator-generated Ada

1 17 lines were added for debugging purposes
2 3 lines were added to compensate for a bug in the Sun Ada compiler
3 The number in parenthesis is the SLOC required to fully implement exponentiation with a floating point exponent
QA9 programs. A discussion of this metric is found in Appendix D. The information in this table and the information in Figure A-3 combine to yield important insight into the differences in amount and distribution of control complexity between the three translators. As can be seen in Table C-5, each translator-generated value for weighted V(G) is within 2% of the others. Figure A-3 shows that the distribution of V(G) across subprograms is also very similar among translator-based QA9 programs. However, Table C-5 also indicates that the CMS-2 QA9 has substantially more complexity than the translator-based QA9 programs. This difference is present because of a CMS-2 construct, procedure switch, that is counted as having higher complexity than its Ada counterpart, the case statement. When this section of CMS-2 code was visually compared to its Ada counterpart, its control structure appeared to be very similar.

<table>
<thead>
<tr>
<th>QA9 Version</th>
<th>Weighted McCabe Complexity Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMS-2 QA9</td>
<td>92 (343143/3733)1</td>
</tr>
<tr>
<td>Ada QA9 produced by APL</td>
<td>65 (235132/3594)2</td>
</tr>
<tr>
<td>Ada QA9 produced by CCCC</td>
<td>67 (234126/3500)</td>
</tr>
<tr>
<td>Ada QA9 produced by TRADA</td>
<td>66 (236813/3572)</td>
</tr>
<tr>
<td>Ada 95 QA9 Redesigned/Rewritten3</td>
<td>1.1 (1802/1677)</td>
</tr>
</tbody>
</table>

Table C-6, Program Size, shows another revealing aspect of the QA9 programs. This shows the number of executable statements as measured by the CMS-2 source code Metrics Generator and by Logiscopic. In this case, the Ada version of QA9 with the largest number of executable statements has fewer than 19% (3887-3297)/3297) more executable statements than the CMS-2 version. There is more variability in Halstead program length than in executable statements, however, average statement complexity (program length/executable statements) is relatively similar, with the Ada programs at the extremes.

The data in Table C-5 and C-6, and in Figure A-2 and A-3 indicate that the CMS-2 ancestor and the translator-generated Ada versions of QA9 are very similar in structure, content, and size. This leads to the unremarkable but important implication that translator output will be very similar to translator input in structure, content, and size.

---

1 SLOC counts used in CMS-2 calculation are straight lines of text. CMS-2 complexity is due to a large extend because of a complex "if statement" in QA9A (QA9A V(G) = 194).

2 SLOC counts used in Ada are counted by Logiscopic.

3 Because of the design and evolution of this test code, great improvements could be made in code efficiency. Reengineering of most legacy code is likely to result in substantial improvements, but perhaps not as dramatic as achieved here.
ADA 95 QA9: REENGINEERING A MIXED-MODE MATH TEST IN ADA 95

The decision to generate a baseline against which to compare the properties of translator-produced code and the effort required to use translation was based primarily on three considerations. The requirements were relatively simple and well-understood. The program, Ada 95 QA9, could also be produced in a relatively short amount of time. Finally, the resulting program metrics would provide an objective measure of the potential differences between redevelopment and translation.

Application redevelopment affords many opportunities for improvement during legacy system migration via requirement-level reengineering, exploiting modern language features, and design for reuse. Requirement-level reengineering in this case means reconsidering functionality in a CMS-2 application and generating a design and implementation that meets the requirements provided by that functionality. Additional requirements may be put in place such as reducing potential maintenance cost or improving performance. In this exercise an artificially-imposed new requirement was to reduce potential maintenance costs as indicated by $V(G)$ (McCabe cyclomatic complexity) and to enhance reusability.

The CMS-2 QA9 program tests accuracy of certain mathematical operations and places an emphasis on mixed-mode arithmetic. It tests various combinations of integer, real, and fixed point operands and targets. Ada 95 QA9 framed the solution as the repetitive application of the pattern $op_1 = op_2 \text{ infix-op } op_3$ using three numeric types and five kinds of infix operations. Since there are three different numeric types for each of the operands $op_1$, $op_2$, and $op_3$, and five different values for $\text{infix-op}$ (i.e., $+, -, /, *, **$), the number of basic kinds of test cases is 135 ($3 \times 3 \times 5 \times 3$). However, since there is no available exponentiation ($**$) operator for fixed point types, 9 must be subtracted from 135 to yield a total of 126 basic kinds of test cases. There must also be an accuracy constraint on the result so that the pattern $\text{lower-bound} \leq op_1 \leq \text{upper-bound}$ must also be a part of the solution. Appendix H contains a more detailed explanation of the Ada 95 QA9 design.

As seen in Table C-5, the weighted McCabe complexity ($V(G)$) for the Ada 95 QA9 (1.1) was less than 2% of the values for the translator-generated QA9 programs (65-67). Keep in mind that a McCabe complexity greater than 50 is considered to be incomprehensible and less than 5 are considered simple and easy to understand. The dramatic reduction was due to the approach taken for test case selection and execution. The translator-generated QA9s used conventional if-then-else and goto semantics. However, Ada 95 QA9 defined separate test cases as subclasses (using Ada 95 Object-Oriented capabilities) and relied on the Ada 95 run-time dispatcher for polymorphic operations to select the appropriate subprogram (i.e., method) to execute for each test case. Ada-ASSURED was also invoked to check conformance to Software Productivity Consortium (SPC) Ada guidelines. There was 100% conformance with SPC guidelines.

Table C-6 also indicates a dramatic reduction in the number of executable statements required to perform the test. An executable statement is statement between a “begin” and “end” that is not in a declarative block. While the other QA9 programs did execute more test cases the comparison of number of executable statements is still valid. This is because in Ada 95 QA9, the number of executable statements is independent of the number of test cases executed. Halstead program length and average statement complexity (executable statements/Halstead program length) is also given in the table. Appendix D explains Halstead program length.

Thirty hours were required to develop Ada 95 QA9. This includes the time required for an experienced Ada 83 developer to gain a sufficient understanding of the object-oriented features of
Ada 95. The Ada 95 QA9 experiment shows that significant improvements in certain indicators of software maintenance cost can be obtained through redevelopment. However, many factors must be taken into account when deciding what course of action to take with respect to a legacy system. Redevelopment may be an appropriate choice under certain circumstances.

### Table C-6. QA9 Program Size

<table>
<thead>
<tr>
<th></th>
<th>Executable Statements</th>
<th>Halstead Program Length</th>
<th>Avg. Statement Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMS-2 QA9</td>
<td>3297</td>
<td>15609</td>
<td>4.73</td>
</tr>
<tr>
<td>APL</td>
<td>3642</td>
<td>14710</td>
<td>4.04</td>
</tr>
<tr>
<td>CCCC</td>
<td>3887</td>
<td>19547</td>
<td>5.03</td>
</tr>
<tr>
<td>TRADA</td>
<td>3759</td>
<td>22037</td>
<td>5.86</td>
</tr>
<tr>
<td>Ada 95 QA9</td>
<td>391</td>
<td>-¹</td>
<td>-¹</td>
</tr>
</tbody>
</table>

### CONCLUSIONS

1. The three translators studied are capable or nearly capable of generating Ada programs that compile and execute correctly.²

2. All three translators produced versions of QA9 that were very similar in complexity, content, and program size (executable statements, Halstead program length, average statement length).

3. The CMS-2 QA9 was very similar in complexity, content, and program size to the translator-generated Ada versions.

4. The quality of generated output will be approximately the same as the CMS-2 input.

5. Only use effort metrics for making “ballpark” estimates of the effort required to translate a CMS-2 system. This is true because of the small sample size (1), questions about the representativeness of the QA9 application, and the uniqueness of each application. Person hours must be adjusted upward to account for direct code, overlays, device dependent IO, and other differences.

6. No significant difference in the difficulty to convert code was found between the three translators.

¹ Logiscoped does not calculate Halstead metrics on Ada 95 source code.

² This assessment did not address the difficulty of converting direct code, overlays, or device-dependent IO.
7. Ada 95 is a better translation target than Ada 83 for many reasons, one of which is the availability of more mathematical functions.

8. Dramatic improvements in quality indicators through redevelopment are a possibility. This option should be given serious consideration when maintenance cost is a significant concern.
APPENDIX D : METRICS INTERPRETATION

The purpose of this appendix is to provide an explanation of the metrics maintained during the translator evaluation process. The outline below shows the metrics collected. Metrics are grouped by intended use. Tools used to calculate metrics are included in parentheses.

- Characterize the CMS-2 Source Code
  - McCabe Cyclomatic Complexity (METRC)
  - Halstead Metrics (METRC)
  - Source lines of code (METRC)
- Examine the quality of the Ada source code produced
  - McCabe Cyclomatic Complexity (Logiscope)
  - Halstead Metrics (Logiscope)
  - Software Productivity Consortium Ada quality and style guidelines (Ada-ASSURED)
  - Source Lines of Code (ASLOC)
- Compare level of correspondence between the CMS-2 source and translated Ada,
  - McCabe Cyclomatic Complexity (METRC, Logiscope)
  - Halstead Metrics (METRC, Logiscope)
  - Source Lines of Code (METRC, ASLOC)
  - Translation Source Lines of Code Ratio
- Examine effort
  - Person-hours
  - Difficulty of Conversion Hours
  - Difficulty of Conversion Source Lines of Code

This appendix provides an explanation of these metrics in the following order:

- McCabe Cyclomatic Complexity
- Halstead Metrics
- Source Lines of Code
- Software Productivity Consortium Ada Quality and Style Metrics
- Person-hours
- Difficulty of Conversion Hours
- Difficulty of Conversion Source Lines of Code
- Translation Source Lines of Code Ratio
MCCABE CYCLOMATIC COMPLEXITY

McCabe’s cyclomatic complexity, $V(G)$, is based on a graph theoretic interpretation of program control flow and provides an indication of structural complexity. The graph of interest is the decision-to-decision path or DD-Path graph (Jorgenson, 1995). A DD-Path graph depicts the paths between decision points in a module or program. The formula for cyclomatic complexity is $V(G) = e - n + 2p$, where $e$ is the number of edges (arcs), $n$ is the number of nodes, and $p$ is the number of connected regions in the graph. $V(G)$ is equal to the number of linearly independent circuits, or “basis paths,” in a DD-Path graph. Figure D-1 contains a short program, “paths,” in which $V(G) = 4$. The four basis paths depicted in the graph can be traced by visiting each of the listed nodes in the stated order.

\{1,2,3,4,1,5\}
\{1,2,3,1,5\}
\{1,2,1,5\}
\{1,5\}

$V(G)$ has important implications for effort required in path testing since all DD-Paths will be tested if all the “basis paths” are covered. Since at least one test case must be constructed for each basis path to be tested, path testing effort will be proportional to $V(G)$ and “testing level.” Two examples of testing level are $C_1$, or DD-path testing, and $C_{1k}$, where each program path containing up to $k$ repetitions of each loop is tested (Jorgenson, 1995). For the program in Figure D-1, $C_1$ testing would require generation of a minimum or four test cases. The total number of paths in zero to five iterations of the loop in program “paths” is $\sum_{j=0}^{5} (V(G)-1)^j = 1074$. It is also the number of test cases that must be generated to meet a $C_{1k}$ test requirement for $k=5$.

---

1 (Jorgenson 1995) notes that there is some confusion about the formula for $V(G)$. The alternative formula substitutes $1p$ for the $2p$ term used here. However, that method adds an edge from the terminal node to the start node, so, both versions yield the same result.

2 The formula only applies to this graph and is not a general equation for computing the number of cases for a particular test requirement.
with procl, proc2, proc3, proc4;
with Set Values;
procedure Paths is
  A, B, C: Boolean;
begin
  Set Values(A, B, C);
  while A loop -- node 1
    proc1;
    if B then -- node 2
      proc2;
      if C then -- node 3
        proc3;
        else -- node 4
          proc4;
          end if;
        proc5;
      end if;
    end loop;
  end Paths; -- node 5

Figure D-1. DD-Path graph for paths program

$V(G)$ is not without problems. $V(G)$ would still be 4 for program "paths" even if the loop statement were replaced by an if statement. The number of possible paths for the if statement version would be 4, but the number of possible paths for the loop statement version would be $\sum_{j=0}^{\infty} (3)^j$ for up to $j$ iterations of the loop. $V(G)$ is related to, but not equal to the number of paths in a program. Another problem with cyclomatic complexity is that it does not take data dependence into consideration in the calculation of number of paths. If the following version of procedure "Set_Values" were used by program "paths," all basis paths in the program would be feasible.

with Random;
procedure Set Values
  (A: out Boolean;
   B: out Boolean;
   C: out Boolean) is
  K : Float := Random;
begin
  A := Boolean\'val(K > 0.0 and K < 100.0);
  B := Boolean\'val(K > -1.0 and K < 1.0);
  C := Boolean\'val(K = 0.5);
end Set_Values;
However, if the following version of procedure “Set_Values” were used, basis paths \{1,2,3,4,1,5\} and \{1,2,3,1,5\} would be unreachable and would constitute sections of “dead code.” The graph depicting reachable sections of code is shown in Figure D-2.

```plaintext
with Random;
procedure Set_Values
  (A : out Boolean;
   B : out Boolean;
   C : out Boolean) is
  K : Float := Random;
begin
  A := Boolean'val(K > 0.0);
  B := Boolean'val(K = 0.0);
  C := Boolean'val(K < 0.0);
end Set_Values;
```

Empirical studies reveal that programs with cyclomatic complexities less than 5 are generally considered simple and easy to understand (Jones, 1991). A good rule of thumb for software development projects is that modules with cyclomatic complexities greater than 10 should be reexamined for possible simplification and that values greater than 20 indicate that serious scrutiny of the source is required. Modules with cyclomatic complexities greater than 50 are generally considered to be incomprehensible. However, these are only guidelines and there are exceptions. For example, long case statements yielding large values of $V(G)$ can be simple to understand because of the inherent mutual exclusivity of the cases. However, a comparable sequence of if statements may be harder to comprehend because successive if statements are not inherently mutually exclusive. Mutual exclusivity for if statements is data dependent. Such data dependencies may not be understandable through examination of the local structure. In these cases cyclomatic complexity serves as a “red flag” for potential understandability problems.

Per-module $V(G)$ may be misleading when used to assess total program complexity. This is because there may be many small modules with low values of $V(G)$. The sum of $V(G)$ for all modules in a program is not a good indication of $V(G)$ since a program with 100 modules of $V(G) = 1$ has much simpler control-flow complexity than a program with a single module with $V(G) = 100$. In addition, average $V(G)$ computed as $V(G)_{avg} = \frac{\sum_{k=1}^{N}V(G)_k}{n}$ is also slightly misleading. Programs with many small modules of low cyclomatic complexity but with few large modules with relatively high values of $V(G)$ will yield a relatively small value for $V(G)_{avg}$, perhaps giving the impression that the program is relatively simple. Consider the example of a program containing 25 modules of one statement each with $V(G) = 1$, and one module with 250 statements with $V(G) = 25$. For this program, $V(G)_{avg} = \frac{(25*1 + 1*25)}{26} \approx 2$. This value is well within the normally acceptable range. $V(G)_{avg}$ considered in isolation obscures the fact that the majority of the source code statements in this program are located in an area of high cyclomatic complexity.
Logiscope computes $V(G)_{avg}$. Average cyclomatic complexity weighted by lines of source code is a more meaningful indication of program $V(G)$. For example, let $C_k$ be source lines of code for module $k$ and $C_T$ be total source lines of code in a program. A weighted $V(G)$ such as $V(G)_{wavg} = \sum_{k=1}^{n} (V(G)_k * C_k) / (C_T * n)$ would give a better indication of the total complexity in the program. In the example above

$$V(G)_{wavg} = (25*(1*1)+1*(25*250))/275 = 6275/275 \approx 23.$$  

This report uses the weighted average McCabe metric rather than average.

The McCabe cyclomatic complexity metric addresses the following questions:

- What is the level of cyclomatic complexity of the CMS-2 source?
- Can CMS-2 source code with high cyclomatic complexity be translated into Ada?
- Is there a similar distribution of cyclomatic complexity between the CMS-2 input and the generated Ada?
- How different or similar are the cyclomatic complexities of the outputs of the various translators?
- How understandable is the generated Ada on the basis of cyclomatic complexity?
HALSTEAD METRICS

Three of the Halstead metrics are of use in comparing the input and output of the CMS-2 translators. They are program (or module) vocabulary size, program length, and volume (Halstead, 1977).

Vocabulary size, $\eta$ (Greek eta), is total number of unique operators and operands in a program.

$\eta_1$: the number of unique operators
$\eta_2$: the number of unique operands
$\eta = \eta_1 + \eta_2$

Program length, $N$, is the total number of occurrences of operators and operands.

$N_1$: the total usage, or count of all occurrences of operators
$N_2$: the total usage, or count of all occurrences of operands
$N = N_1 + N_2$

Program volume, $V$, can be thought of as the number of bits needed to represent a given program in the main memory of a special-purpose computer designed to execute that program (Halstead & Schneider, 1980). This is based on the observation that $\log_2 \eta$ is the minimum number of bits required to represent all of the individual elements of a program.

$V = N \log_2(\eta_1 + \eta_2) = N \log_2 \eta$

Halstead developed other equations to predict such things as programming effort and number of errors. However, those aspects of the theory are not particularly relevant to this evaluation. The Halstead metrics used here describe the textual content and complexity of a program on a per-subprogram basis. That is, comparisons based on these Halstead metrics between translator input and translator output, and between translator outputs give a high level description of the textual similarities between the various versions of the same program.

SOURCE LINES OF CODE (SLOC)

SLOC has been used historically as a means to understand program size. It has been valuable for estimating complexity, costs, productivity, and many other programming metrics. There are a number of problems with the "source lines of code" (SLOC) metric. No standards exist for counting SLOC in any programming language. That makes it difficult to compare programs written in different programming languages on the basis of SLOC. In addition, the amount of code produced for the same specification written in the same programming language can differ by a factor of five between programmers due to individual programming style (Jones, 1991). It is not clear that a smaller or larger program is preferable. A smaller program may be more terse and have more statement complexity. A larger program may be more readable, or may be less efficient. The SLOC metric does not distinguish degrees of complexity, efficiency or understandability.

The CMS-2 SLOC is a count of three things: lines ending in '!', comment lines, and total lines of text. The lines reported as "LOC" in the CMS-2 SLOC count were computed as the total number of lines ending in ‘!’ minus the number of comment lines. Comment lines were counted as lines in
which the word “comment” occupied character positions 11 through 17. The UNIX “grep” and “vi” programs were used to count CMS-2 SLOC.

The Ada line counter also counts three things: non-embedded semicolons, comments, and lines of text. The number of non-embedded semicolons is the count of all semicolons except those occurring in comments and character strings. Comment lines were counted as lines which contained two successive hyphens not embedded in a character string. SLOC counting in CMS-2 sample was line-oriented in that each line of text was interpreted to be either a comment, an executable statement, or a blank line. This was verified upon visual inspection of the Quick Look CMS-2 sample. Multiple non-embedded semicolons may occur on the same line in Ada. In addition, comments and terminal semicolons may be located on the same line of text in an Ada program. It is possible in Ada to have the sum of the number of comments and SLOC exceed the total number of lines of text in a file of Ada source code. The Ada line counter, ASLOC, that was written and used to count SLOC for this translator evaluation is found in Appendix J.

SOFTWARE PRODUCTIVITY CONSORTIUM (SPC) METRICS

The SPC has developed a set of guidelines for Ada programmers to support the development of high-quality, reliable, reusable, and portable software (Software Productivity Consortium, 1992). Ada-ASSURED is an Ada source code processor that is a language-sensitive editor, programming standards enforcer, and pretty-printer (GrammaTech, 1995). In the default configuration, its standards enforcement capability is strongly related to the SPC guidelines. It takes Ada source code as input and generates a new listing, formatted according to SPC guidelines, and including in-line diagnostics that map to SPC guidelines. There is a many-to-many relationship between the Ada-ASSURED diagnostics and the SPC guidelines. This is due to the fact that Ada-ASSURED operates at the syntactic level and there is a many-to-many relationship between Ada syntax and SPC guidelines.

The Quick Look Ada QA9 samples were processed with Ada-ASSURED. A number of diagnostics relating to Ada-ASSURED violations were produced. In general, it probably is desirable to change the offending sections of code associated with Ada-ASSURED violations so that they comply with the SPC guidelines. However, this is not necessarily the case for translated code. In general, the closer the translator output is to the input, the easier it is to verify correct translation. There are two primary reasons for this. First, it is easier to understand the relationships between two similarly structured programs. Second, there may also be test programs in the original language that are candidates for translation. The closer the translated code is to the original code, the more likely it is that the original test cases and procedures will be useful in testing the translated code. Once the translated code is verified and tested, much can be gained by reengineering the code and applying the SPC guidelines.

This section provides a discussion of the meaning of the Ada-ASSURED violations that were encountered on the translator-produced Ada QA9 samples. (The reader is referred to Tables A-5 through A-8 for the number of occurrences of these errors and for the exact statements that were flagged.)

Ada-ASSURED violations are designated with “V” for violation and a number, n, which identifies the violation. The violations produced for the Quick Look sample are discussed in the
following sections. Each violation is discussed in the context of SPC guidelines and implications for testing and certification.

- V0: “The identifier/keyword `<id>` is used in context `<context>`” (GrammaTech, 1995). Each occurrence of V0 was due to the use of a “use clause”. The presence or absence of “use clauses” has no effect on source code structure. The SPC guideline from (SPC92 sec. 5.7.1) is

Minimize using the “use clause”

Consider using the “use clause” in the following situations:
1. Infix operators are needed
2. Standard packages are needed and no ambiguous references are introduced
3. References to enumeration literals are needed

Consider the `renames` clause to avoid the “use clause”

Localize the effect of all “use clauses”.

In the absence of a “use clause”, qualified naming must be used to refer to all entities declared outside the current scope. For example, if main procedure `Z`, a client of `package X`, invokes `procedure Y of package X`, all references to `Y` in `Z` must appear as “X.Y.” In the presence of a “use clause”, references to `Y` in `Z` may appear simply as “Y.” Qualified naming makes the source of the identifier (e.g., `Y`) obvious (e.g., `X.Y` implies that `Y` is declared in `X`). The presence of the “use clause” decreases program understanding because it obscures the origin of identifiers. This is why many projects ban the “use clause” and may be why the SPC guidelines advise minimizing its use.

However, the “use clause” can eliminate a certain amount of clutter and unwieldiness in writing and maintaining programs with server packages having long names. This is particularly true for mathematically oriented programs. Ada provides programmers the capability to declare derived versions of standard numeric types. Such declarations may be used to prevent errors such as adding a variable for voltage to a variable for longitude. The operations on a derived type defined in a server package, are not, by default, visible to clients of the package. In the absence of a “use clause” for the server package, the required syntax for an infix operation for such a type is the same as for a function call. The following infix operators for floating point types are affected: `<`, `<=`, `=`, `/=`, `=`, `>`, `>=`, `>`, `+`, `-`, `*`, `/`, and `**`.

Figure D-3 depicts the case in which no “use clause” is used. It is quite cluttered in comparison to Figure D-4 which has a “use clause”. However, use of qualified naming in Figure D-4 makes the origin of the declarations clear whereas the “use clause” has introduced ambiguity with respect to the origins of the variables in Figure D-4.
with First_Long Package_Name;
with Second_Long Package_Name
procedure A83 Nu Nr is
begin
    First_Long Package_Name.Sum := "First_Long Package_Name."+"(First_Long Package_Name.G1, First_Long Package_Name.G2);
end A83 Nu Nr;

Figure D-3. Procedure Accessing Global Variables without Renaming and without a “Use Clause”

with First_Long Package_Name;
use First_Long Package_Name;
with Second_Long Package_Name;
use Second_Long Package_Name;
procedure A83 U Nr is
begin
    Sum := G1 + G2;
end A83 U Nr;

Figure D-4. Procedure Accessing Global Variables with a “Use Clause”

The SPC recommendation to use renaming, presumably to allow normal infix format of expression, has been obviated by the introduction of the Ada 95 “use type clause”. Figure D-5 shows an Ada 83 example of renaming the “+” operator. This gives the addition statement a more familiar appearance and requires a rather lengthy renaming statement to achieve that effect. The addition statement is still relatively cluttered due to the length of name of the server package. Figure D-6 shows an Ada 83 example of renaming the long server package name in addition to the “+” operator. This results in a much simpler and unambiguous statement syntax through the addition of four words.

with First_Long Package_Name;
procedure A83 Nu_Ro is
    function "i" (Left, Right : in First_Long Package_Name.Real)
        return First_Long Package_Name.Real
            renames First_Long Package_Name."+";
begin
    First_Long Package_Name.Sum := First_Long Package_Name.G1 + First_Long Package_Name.G2;
end A83 Nu_Ro;

Figure D-5. Procedure Accessing Global Variables with a Renamed Addition Operator and without a “Use Clause”
with First_Long_Package_Name;
procedure A83_Ro_Rc is
  package Flpn renames First_Long_Package_Name;
  function "+" (Left, Right : in Flpn.Real)
    return Flpn.Real renames Flpn."+";
begin
  Flpn.Sum := Flpn.G1 + Flpn.G2;
end A83_Ro_Rc;

Figure D-6. Procedure Accessing Global Variables with a Renamed Server Package and Addition Operator and without a “Use Clause”

Figure D-7 illustrates use of the Ada 95 “use type clause” which provides direct visibility of a type’s operators. This has the same affect as renaming the “+” operator as depicted in Figure D-5. Figure D-8 shows use of the “use type clause” in conjunction with package renaming. While it is not as brief as Figure D-4 which uses the “use clause” it is unambiguous. However, it is relatively brief and uncluttered compared to the other alternatives.

with First_Long_Package_Name;
procedure A95_Ut_Nr is
  use type First_Long_Package_Name.Real;
begin
  First_Long_Package_Name.Sum := First_Long_Package_Name.G1 + First_Long_Package_Name.G2;
end A95_Ut_Nr;

Figure D-7. Ada 95 Procedure Accessing Global Variables with a “Use Type Clause” and no Renaming

with First_Long_Package_Name;
procedure A95_Ut_Rc is
  package Flpn renames First_Long_Package_Name;
  use type Flpn.Real;
begin
  Flpn.Sum := Flpn.G1 + Flpn.G2;
end A95_Rc;

Figure D-8. Ada 95 Procedure Accessing Global Variables with a “Use Type Clause” and with a Renamed Server Package

Use of the “use clause” can decrease that part of the maintainer’s cognitive load pertaining to cluttered source code. This amount of the decrease is related to the length of the names of the server packages. On the other hand, the “use clause” increases the part of the maintainer’s cognitive load pertaining to correct comprehension of the roles and relationships of the various packages comprising a program. During maintenance, it is not sufficient to just correct, enhance, or add functionality. It must be done without introducing unknown side effects to any other part of the program. Use of the “use clause” makes this more difficult because it obscures the origins of identifiers.
• V1: “A list with this many items must be a named association list.” (GrammaTech, 1995). There is no difference in code structure resulting from use of either positional or named association. Each occurrence of V1 was due to the use of an array aggregate. The SPC guidelines referenced by V1 are related to named association. (Software Productivity Consortium, 1992) and aggregates (Software Productivity Consortium, 1992). The SPC guidelines for named association do not mention aggregates. However, one of the guidelines for the aggregates states “Use positional association only when there is a conventional ordering of the arguments” (Software Productivity Consortium, 1992). There is also reference to named association in the rationale section for aggregates which states:

Aggregates can also be a real convenience in combining data items into a record or array structure required for passing the information as a parameter. Named component association makes aggregates more readable.

In this case, the Ada-ASSURED violation does not seem to indicate noncompliance with SPC guidelines. The aggregates in question are array aggregates with integer indexes. As such, the applicable guideline should probably be the one cited above applying to “conventional ordering of arguments.”

• V4: “Use of GOTO not allowed.” V5: “Labels are not allowed” (GrammaTech, 1995). Both of these violations reference (Software Productivity Consortium, 1992) “Do not use goto statements.” Loop, if, and case statements are what must be used to replace GOTO..<label> pairs. There are combinations of GOTO...<label> pairs for which there is no simple equivalent in goto-less programming. Eliminating GOTO statements in translated code could increase required testing effort due to significant changes in code structure.

• V7: “Nested loops must all be named.” V8: “Exit statements from named loops must be named.” V10: “All BLOCKS must be named.” V25: “A loop this long must be named.” There is no difference in code structure resulting from use or lack of use of loop, exit, or block statement names. The applicable guidelines and portions of the rationales follow:

1. (Software Productivity Consortium, 1992): Associate names with loops when they are nested.

When you associate a name with a loop, you must include that name with the associated end for that loop (Department of Defense, 1983). This helps readers find the associated end for any given loop ... The choice of a good name for the loop documents its purpose.

2. (Software Productivity Consortium, 1992): Associate names with blocks when they are nested.

When there is a nested block structure, it can be difficult to determine which end corresponds to which block. Naming blocks alleviates this confusion.

3. (Software Productivity Consortium, 1992): Use loop names on all exit statements from nested loops.
An exit statement is an implicit goto. It should specify its source explicitly. When there is a nested loop structure and an exit statement is used, it can be difficult to determine which loop is being exited. Also, future changes which may introduce a nested loop are likely to introduce an error, with the exit accidentally exiting from the wrong loop. Naming loops and their exits alleviates this confusion.

- V12: “Non-constant object declarations are not permitted in the visible part of a package specification.” The applicable guideline is “Avoid declaring variables in package specifications” (Software Productivity Consortium, 1992).

There can be a significant difference in source code structure between programs with and without non-constant object declarations in package specifications. Moreover, it is unclear that any significant benefit would be obtained by simply declaring access-subprograms for variables formerly declared in a package specification. Compare Figure D-9 with Figure D-8 to see the stylistic difference.

```ada
with First_Long_Package_Name;
procedure A95_UC_Rc is
  package Flpn Renames First_Long_Package_Name;
  use type Flpn.Real;
begin
  Flpn.Put_Sum(Flpn.Get_G1 + Flpn.Get_G2);
end A95_Rc;
```

**Figure D-9. Ada 95 Procedure Using Access-Subprograms with a “Use Type Clause” and with a Renamed Server Package**

The guideline against declaring variables in package specifications is more meaningful in the context of type and object managers. In those cases the operations on the type are carefully crafted so that the objects can only be accessed in prescribed ways. Cohen (1996) has an example of a type manager for “Length_Type” such that the multiplication operation returns a value of type “Area_Type,” not “Length_Type.” In his example, a variable of type “Length_Type” cannot be the result type of a multiplication operation with operands of type “Length_Type.” The constraints imposed by this package design preclude certain types of programming errors. However, in the context of translated code, conversion from the standard arithmetic approach to the type and object manager approach constitutes a reengineering effort with potentially significant maintenance consequences for the rest of the program.

- V17: “Subprogram body size of <n> exceeds maximum of <m>.” There is no SPC reference for this violation. However, a review by Banker (1993) of several studies the optimum values of SLOC/module indicate that it is below the DoD’s proposed standard of 200 SLOC/module. Nevertheless, placing an upper limit on module (subprogram) size for translator output could result in programs that were structurally dissimilar to the original CMS-2 programs.

**PERSON-HOURS**

Person-hours metrics were kept to assist others who are considering translating project code. This information may be useful in estimating the time and dollars required to perform translations. Detailed person-hours were kept for the steps of the three phases of the translator evaluation process,
the steps of the preliminary work, as well as for general tasks. General tasks included metrics collection, preparing and giving presentations, and writing the reports.

DIFFICULTY OF CONVERSION HOURS (DOCH)

This metric is calculated as

\[ \text{DOCH} = \text{HCC} + \text{HEC} \]

Where HCC is hours spent modifying translated code until compiles correctly and HEC is hours spent reengineering Ada code until executes correctly.

This metric was included for comparing the reengineering effort needed to move the translated code to correct execution. It was intended primarily for comparing translators, but could also be used for comparisons across compilers.

DIFFICULTY OF CONVERSION SLOC (DOCS)

This metric is calculated as

\[ \text{DOCS} = \text{SCC} + \text{SEC} \]

Where SCC is SLOC added or modified until translated Ada code compiles correctly and SEC is SLOC added or modified to reengineer Ada code until executes correctly.

This metric is very similar to DOCH. It was collected for the same purpose. This metric was kept because of potential bias problems with DOCH. We felt that the software engineer would be learning as he/she takes the translated Ada code produced by the three translators through the Reengineer Until Ada Code Executes Correctly phase. The second set of translated Ada may be completed faster than the first and the third faster than the second because of the learning experience. We believe that DOCS is less biased.

TRANSLATION SOURCE LINES OF CODE RATIO

This metric is calculated as

\[ \text{Translation SLOC ratio} = \frac{\text{Ada SLOC}}{\text{CMS-2 SLOC}} \]

It is used for comparing the size of the translator-produced Ada source with the corresponding CMS-2 code.
APPENDIX E: POTENTIAL FOLLOW-ON WORK

This appendix describes several translator evaluation tasks that could be done if additional time and funding were available.

IMPROVE QUALITY OF TRANSLATED ADA SOURCE

This task would address methodologies, tools, and effort to convert correctly executing Ada code to high quality, maintainable, Ada code. A key research activity could be to identify specific reengineering tool requirements that would facilitate the use of translated Ada code. The current research project has already identified some reengineering capabilities needed. Tool vendors may be responsive to incorporating these requirements into their products once they are identified. Initial requirements to support translation not normally satisfied by Ada reengineering tools include:

- Remove GOTO statements
- Remove dead code
- Convert global objects to local objects
- Eliminate subprogram call side effects to global variables
- Move type definitions and subprogram declarations to package bodies where appropriate for information hiding
- Create meaningful types and object names
- Reposition code into packages

This task could begin at the completion of the third phase, Reengineer Until Ada Code Executes Correctly. The quality of the translated Ada source code would be improved by using tools and by making manual changes. Ada source code produced by translators mirrors the CMS-2 code and does not take advantage of Ada typing, packaging, exception handling, and useful software engineering capabilities offered by Ada and Ada 95. The source code produced needs to be brought into conformance with the “Ada Quality and Style Guidelines for Professional Programmers,” (Software Productivity Consortium, 1992).

Tools that would assist in the quality improvement of the Ada source code need to be identified, obtained, and installed. Some of these tools identify problems and others can automatically fix them. Some of these tools were already used during the evaluation to assess quality (Table L-1).

Other potentially useful tools to be considered for this task are described in Table L-2. Others need to be identified.

This source code quality improvement task includes the steps listed below. This task could start with an Ada version of QA9 or another translated sample.

- Examine the quality of translated and correctly executing Ada/Ada 95 sample using tools Candidate tools include: Ada-ASSURED, AdaMat, and Logiscope. Much of this has already been done under the translator evaluation.

- Experiment with existing Ada quality improvement tools
Tools include: Rational’s Reengineering Toolkit, Xinotech’s Composer and Xinotech’s prototype Object Extractor, and Ada-ASSURED. Feedback would be provided to tool developers for improvements.

- Make manual code improvement changes that existing tools cannot handle
  We expect that these changes would include removal of GOTO statements, elimination of dead code, pushing scoping to appropriate level, partitioning code into packages, replace translated identifiers that are usually related to the eight character CMS-2 names, by more meaningful identifiers, and others. A product of this step would be specific recommendations to tool developers for new automated capabilities for Ada source code quality improvement.

- Experiment with new Ada documentation tools
  These tools include CCC’s Hyperbook and I-DOC, a prototype tool developed by the University of Southern California with DARPA funding. Feedback would also be provided to developers for tool improvement.

- Reexamine quality of Ada code using tools
  The quality of the enhanced Ada/Ada 95 code would be re-measured using tools and compared with translated code from the initial step.

EXAMINE PERFORMANCE OF EXECUTING ADA COMPONENTS

This task would compare the performance of three translations and one redesign/rewrite of a portion of an existing CMS-2 system. The translations are correctly executing Ada 95 programs produced by the APL, CCC, and TRADA translators and the fourth is a manual redesign/rewrite in Ada 95 of the CMS-2 components. Comparisons of executable size, memory usage, and run-time performance would be made. Executable size comparisons can be easily done while memory and timing measurements are considerably more difficult. A manageable size operational CMS-2 project would be selected for the performance comparison. QA tests would not be used. MK-2 is a candidate sample.

EVALUATE OTHER TRANSLATOR CAPABILITIES

- Test the overlay capability of the CCC translator using MTASS QA3 and QA60. Both are self checking tests that use a test controller.
APPENDIX F : RECORD FOR REENGINEER UNTIL ADA CODE EXECUTES CORRECTLY

This appendix is intended to assist software engineers who plan to use the translators. It is a log containing the details of the steps followed to achieve correct execution in Ada. QA9 was taken to valid execution following translation by the TRADA, CCCC, and APL translators. Logs are provided for the following combinations of translators and compilers:

QA9 TRADA       VAX Ada
QA9 TRADA       Sun Ada
QA9 TRADA       GNAT
QA9 CCCC        GNAT
QA9 CCCC        Sun Ada
QA9 APL         GNAT
QA9 APL         Sun Ada

The exact compilation and execution errors and fixes are included.

TRADA - REENGINEERING RECORD FOR VAX ADA

1. Made minor corrections to test harness adding additional I/O capabilities.

TRADA - REENGINEERING RECORD SUNADA COMPILER

1. A monolithic file was created from separate TRADA files/packages for handling convenience. This big file was broken down into small files. A TRADA summary file provided the compilation order.

This split the monolithic file into the following files with one file per compilation unit.

CMS_2_types.a  
Qa9e.a         
Qa9d.a         
Qa9c.a         
Qa9b.a         
Qa9a.a         
Start.a        
Dryver.a
Aqtcon.a
Major_header.a
CMS_2_types_b.a
Undefined_extrefs.a
Qsysdd1a.a
Qa9qlook_b.a
Aqtcon_b.a
Dryver_b.a
Undefined_extrefs_b.a
Qa9a_b.a
Qa9b_b.a
Qa9c_b.a
Qa9d_b.a
Qa9e_b.a
Start_b.a

Generate compilation script:
    arg db -p -If files
    asg compile files -luada \-v \-IE u

2. Compilation

    source compile

    /home1/users/ollerton/cms2ada/tradada/vads_qa9/CMS_2_types.a, line 160, char 40: error: RM 3.5.7(12): cannot select predefined type: range too big
    /home1/users/ollerton/cms2ada/tradada/vads_qa9/CMS_2_types.a, line 162, char 15: error: RM 3.5.7(12): cannot select predefined type: digits too big

    Requested range of floating point type exceeded platform limitations. Make the following change to remedy the problem.
-- + Bob Ollerton, June 21, 1996
-- + Sun Ada 1.1(j)
-- + RM 3.5.7(12): cannot select predefined type: range too big.
-- + NOTE: 8#0.77777777# is the closest octal rep of n <= 1.0.
-- + There are two floating point representations for SunAda. One
-- + has 6 digits, and a maximum binary exponent (SAFE_EMAX) of 125,
-- + and the other has 15 digits with SAFE_EMAX = 1021. So, both of
-- + these declarations should have exponents of SAFE_EMAX.
-- +TYPE Float_s
-- + IS DIGITS 7
-- + RANGE -8#0.77777777# * 2.0 ** 1023 .. 8#0.77777777# * 2.0 ** 1023;
-- +TYPE Float_d
-- + IS DIGITS 16
-- + RANGE -8#0.77777777777777776#
-- + * 2.0 ** 1023 .. 8#0.77777777777777776#
-- + * 2.0 ** 1023;
TYPE Float_ss
IS DIGITS 7;
TYPE Float_s is DIGITS 7 RANGE
-8#0.77777777# * 2.0 ** Float_ss'Safe_Emax ..
8#0.77777777# * 2.0 ** Float_ss'Safe_Emax;
TYPE Float_d
IS DIGITS System.Max_Digits;
-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

3. Recompilation, link
source compile
No compilation or link errors

4. Execute Qa9look.

SUMMARY OF ERRORS

EXECUTED - 345
NO TESTS ACCOUNTED- 0
EXECUTION ERRORS - 82

5. Execution errors all appear to be due to explicit conversion of a fixed or floating point exponent to an integer. Only integer exponents are available within the Ada 83 standard math operations. Access to other types of exponentiation operators will require access to a math library offering those capabilities. The following code fragment is typical of part of an exponentiation test.

-- ++++++++++++++++++
-- Exponent converted to Ada integer
-- QA9 0151 SET VAWS9 TO VAWS6**VFD1 $
Qsysddl.Vaws9 :=
T 32_s_9 (Float_43 (Qsysddl.Vaws6) ** Integer
(Qsysddl.Vfd1));

F-3
Explicit type conversion is used extensively in the 82 exponentiation tests. In this particular case, function \text{T\_32\_s\_9} returns a value of type \text{Cms\_2\_Types.A\_32\_s\_9}, which is a fixed point type. \text{Qsysd1a.Vaws6} and \text{Qsysd1a.Vfd1} are also of type \text{Cms\_2\_Types.A\_32\_s\_9}. However, \text{Qsysd1a.Vaws6} is explicitly converted to type \text{Cms\_2\_Types.Float\_43} and \text{Qsysd1a.Vfd1} is being converted to type \text{Integer}. The conversion of the exponent to integer has the dramatic effect on precision that could account for the 82 errors.

There is a straightforward and tedious approach to remedying this problem. First, we assume that all of the problems are due to insufficient precision resulting from conversion to an integer exponent and that the problem will be remedied by changing all such instances to conversion to a floating point exponent. This will necessitate other conversions as well. However, examination of package \text{CMS\_2\_Types} reveals that all six floating point types now have the same precision and underlying representation as the predefined type \text{Float}. That being the case, we can use the SunAda Math,\text{"\textsc{**}"} function and explicitly convert the operands to and from the standard type \text{Float}. The code fragment shown above could then become:

\begin{verbatim}
---------
-- Exponent converted to Ada integer
-- Changed by Bob Ollerton: 6/21/96
Qsysd1a.Vaws9 :=
  T_32_s_9 (Float_43(Float (Qsysd1a.Vaws6) ** Float (Qsysd1a.Vfd1)));
\end{verbatim}

This technique must be applied in all cases except for the case in which the test is designed to test \text{x**n}, where \text{n} is of type integer.

6. Recompilation, link

source compile
No compilation or link errors

7. Execute Qa9look.

**SUMMARY OF ERRORS**

**EXECUTED** - 345

**NO TESTS ACCOUNTED** - 0

**EXECUTION ERRORS** - 0
TRADA - REENGINEERING RECORD FOR GNAT COMPILER

1. Take SunAda source as a starting point.

   
   with Ada.Numerics.Generic_Elementary_Functions;
   package Math is new
   Ada.Numerics.Generic_Elementary_Functions(Float);

2. Split into files and generate compilation order

   gnatchop -s SRC

3. Compilation, link and bind

   sh SRC.sh -gnato
   gnatmake qa9look

   No errors

4. Execute qa9look.

   SUMMARY OF ERRORS

   EXECUTED      - 345
   NO TESTS ACCOUNTED- 0
   EXECUTION ERRORS - 0
CCCE - REENGINEERING LOG FOR GNAT COMPILER

1. Concatenate
   
   `cat PREDEFIN.ADA QA9QL.ADA >> SRC`

2. Split into files and generate compilation order
   
   `gnatchop -s SRC`

3. Compilation
   
   `sh SRC.sh -gnato`

The "-gnato" qualifier enables range and elaboration checks.

```
cms2_to_ada_predefined.adb:6:06: file "math_lib.ads" not found
compilation abandoned
math_lib_cms2.ads:2:06: file "math_lib.ads" not found
compilation abandoned
qa9qlook.adb:6:08: file "math_lib.ads" not found
qa9qlook.adb:6:08: "QA9QLOOK (body)" depends on "MATH_LIB_CMS2 (spec)"
qa9qlook.adb:6:08: "MATH_LIB_CMS2 (spec)" depends on "MATH_LIB (spec)"
compilation abandoned
```

This identified a dependency on math_lib.ads which was not part of the distribution.

This is a generic math library with a generic formal parameter named "real."

```ada
with math_lib;
package math_lib_cms2 is new math_lib(real=>float);
```

for math_lib.

```ada
--with math_lib;
--package math_lib_cms2 is new math_lib(real=>float);
with Ada.Numerics.Generic_Elementary_Functions;
package math_lib_cms2 is new
   Ada.Numerics.Generic_Elementary_Functions(Float);
```

F-6
4. Recompilation

No remaining compilation errors, the following warnings were issued:

qa9qlook.adb:694:09: warning: "LX2" is never assigned a value
qa9qlook.adb:695:09: warning: "LX3" is never assigned a value
qa9qlook.adb:833:09: warning: "LX1" is never assigned a value


procedure QA9 is
begin
  QA9qlook.Dryver.Driver;
end QA9;

6. Compile, link, bind. No Errors.

7. Run qa9. Execution output

  raised PROGRAM_ERROR

8. Due to previous experience, assume that the exception was due to

  "access before elaboration." ¹

There are two functions in package QA9QL.QSYSDD1A that are called before
their bodies are elaborated:

FUNCTION TV10H_item_address_access_init RETURN TV10H_item_pointer;
TV10H_data : TV10H_item_pointer:=TV10H_item_address_access_init ;
FUNCTION TV16D_item_address_access_init RETURN TV16D_item_pointer;
TV16D_data : TV16D_item_pointer:=TV16D_item_address_access_init ;

¹ The QA9 test suite for the AN/UYK-7 was input the CCCC translator by mistake. It was during that reengineering
effort that the source of the program_error exception was identified. It was pinpointed by compiling the sample with
the Alys compiler and running it in the Alys debugger. This became quite time-consuming since the required math
library, which is normally part of the Alys distribution, was either missing or was not properly installed. Since the
Alys compiler was no longer under maintenance, we were unable to get technical support to assist us in accessing
the library. The problem was overcome by using the Ada math library provided on the Walnut Creek CD-ROM. It
enabled us to pinpoint the source of the program_error exception, but other run-time errors resulted. Eventually, we
discovered that some of functions in the math libraries from the Walnut Creek CD-ROM were yielding incorrect
results. Use of these libraries was discontinued. Since we neither looked for nor read any documentation on the
Walnut Creek CD-ROM math libraries, we are not in a position to state that they are faulty. We may not have used
them in the intended manner and can only state that they sometimes yielded incorrect results in the manner in which
we used them.
One approach to fixing this problem is to initialize TV10H_data and TV16D_data in the initialization code of the body.

The following changes were made to the specification of QA9QL.QSYSDD1A:

```
-- ***** ***** Changed by Bob Ollerton 8/4/96  ***** *****
FUNCTION TV10H_item_address_access_init
RETURN TV10H_item_pointer ;
TV10H_data : TV10H_item_pointer; -=->TV10H_item_address_access_init
;
FUNCTION TV16D_item_address_access_init
RETURN TV16D_item_pointer ;
TV16D_data : TV16D_item_pointer; -=->TV16D_item_address_access_init
;
-- ***** ***** ***** *****  ***** ***** ***** *****  ***** ***** ******
```

The following was added to the body of QA9QL.QSYSDD1A:

```
...  
-- ********** Added by Bob Ollerton 8/4/96 **********
begin
   TV10H := TV10H_item_address_access_init;
   TV16D := TV16D_item_address_access_init;
   -- ********** **********
END QSYSDD1A ;
```

9. Recompilation

No remaining compilation errors, the following warnings were issued:
qa9qlook.adb:694:09: warning: "LX2" is never assigned a value
qa9qlook.adb:695:09: warning: "LX3" is never assigned a value
qa9qlook.adb:833:09: warning: "LX1" is never assigned a value


Results => no visible behavior.

Modify the program to output an indication of which parts of the program execute.

a) Write and Compile procedure Write.

```
use Ada.Text_Io;
procedure Write
   (Msg : in   String) is
begin
   Put_Line("=>> " & Msg);
end Write;
```

F-8
b) Insert calls to Write at strategic places in Qa9qlook.Dryver.Driver;

-- ******************************************************************
-- Added by Bob Ollerton
-- ******************************************************************
-- Added by Bob Ollerton

WITH cms2_to_ada_predefined;
USE cms2_to_ada_predefined;
WITH UNCHECKED_CONVERSION;
WITH SYSTEM;

PACKAGE BODY DRYVER IS
PROCEDURE DRIVER IS
BEGIN
  Write("calling Start");
  START;
  Write("calling QA9AA");
  QA9A;
  Write("calling QA9AB");
  QA9B;
  Write("calling QA9AC");
  QA9C;
  Write("calling QA9AD");
  QA9D;
  Write("calling QA9AE");
  QA9E;
  Write("calling QTSYNOPS");
  QTSYNOPS;
  Write("calling CMS2_EXEC");
  CMS2_EXEC (8);
  Write("done!");
END DRIVER;
END DRYVER;
c) Insert calls to Write in function TV10H_item_address_access_init and TV16D_item_address_access_init

BEGIN
  Write("calling TV10H_item_address_access_init");
  ...
  Write("returning from TV10H_item_address_access_init");
END
...
BEGIN
  Write("calling TV16D_item_address_access_init");
  ...
  Write("returning from TV16D_item_address_access_init");
END


13. Bind and Link qa9


Results are as desired. Output indicates that all routines were called.

=>> calling TV10H_item_address_access_init
=>> returning from TV10H_item_address_access_init
=>> calling TV16D_item_address_access_init
=>> returning from TV16D_item_address_access_init
=>> calling Start
=>> calling QA9AA
=>> calling QA9AB
=>> calling QA9AC
=>> calling QA9AD
=>> calling QA9AE
=>> calling QTSYNOPS
=>> calling CMS2_EXEC
=>> done!
CCC - REENGINEERING RECORD FOR THE SUNADA COMPILER

Code reengineered for GNAT was used as a starting point

1. There is no standard math library for Ada 83, so attempted to use package Math from Verdielib. Assume that the only operation required from the Math library is exponentiation with floating
point exponent. Develop and compile the following package.

```ada
with math;
package math_lib_cms2 is
  function "**" (Left, right: Float) return Float renames Math."**";
end math_lib_cms2;
```

2. Concatenate the following packages together into one file called SRC:

```ada
cms2_to_ada_predefined.adb
cms2_to_ada_predefined.ads
math_lib_cms2.ads
qa9.adb
qa9qlook.adb
qa9qlook.ads
write.adb

cat *.ad*> SRC
```

3. Split the files apart using the Ada PRImitive Compilation Tool (Apricot) and generate a compilation script.

```bash
apricot SRC db -s
arg db -p -If files
asg compile files -luada -v \!E u
```

4. Execute the compilation script.

```bash
source compile
```
5. Compilation errors.

Package cms2_to_ada_predefined.ads contains a reference to type "long_float" on line 342. This is not a predefined type in Ada 83. Ada 95 provides compiler implementors the option of including the definition of long_float in package standard as a predefined type (ARM 95 3.5.7.16-17).

    function long_flt_image(r: in long_float) return string;

6. Fix: Precede the declaration of long_flt_image in package cms2_to_ada_predefined with the following subtype declaration:

    subtype long_float is float;

7. Compilation errors.

```
*************** cms2_to_ada_predefined.b.a

459:    field_h_proc_x(float_to_bit(value),bstart,blength,dest_word);
A -------------------------^  
A:warning: RM 13.10.2(2): operand is bigger than target
479:    return bit_to_float(field_h_fcn_x(source_word,bstart,blength));
A -------------------------^  
A:warning: RM 13.10.2(2): operand is smaller than target
525:    meu_table_word_proc_x(float_to_cms2word(value),
A -------------------------^  
A:warning: RM 13.10.2(2): operand is bigger than target
536:    meu_table_word_proc_x(
A ---------^  
A:internal: assertion error at file il_code.c, line 181
/home1/users/ollerton/cms2ada/ccc/large/cms2_to_ada_predefined.b.a,
line 459, char 22:warning: RM 13.10.2(2): operand is bigger than target
/home1/users/ollerton/cms2ada/ccc/large/cms2_to_ada_predefined.b.a,
line 479, char 14:warning: RM 13.10.2(2): operand is smaller than target
/home1/users/ollerton/cms2ada/ccc/large/cms2_to_ada_predefined.b.a,
line 525, char 29:warning: RM 13.10.2(2): operand is bigger than target
/home1/users/ollerton/cms2ada/ccc/large/cms2_to_ada_predefined.b.a,
line 536, char 7:internal: assertion error at file il_code.c, line 181
```

F-12
8. The compilation error on line 536 is not a compilation error as such. It is a message stating that
the compiler has crashed. The relevant code fragment is properly constructed:

```pascal
procedure meu_table_word_proc(value: in string;
    size_dim1: in integer;
    size_dim2: in integer;
    array_addr: in address) is

    function bit32_to_cmos2word is new unchecked_conversion
        (source=>bit_string_32, target=>cms2_word);
    begin
        --536
        meu_table_word_proc_x(
            bit32_to_cmos2word(string4_to_bit32(pad(value,4))),
            size_dim1, size_dim2, array_addr);
    end meu_table_word_proc;
```

Past experience has shown that Verdict compilers are sensitive to complex
expressions. We will attempt to simplify the expression.

```pascal
procedure meu_table_word_proc(value: in string;
    size_dim1: in integer;
    size_dim2: in integer;
    array_addr: in address) is

    function bit32_to_cmos2word is new unchecked_conversion
        (source=>bit_string_32, target=>cms2_word);
    begin
        Target := bs32_to_cmos2word(Bs32);
        meu_table_word_proc_x(Target, size_dim1, size_dim2, array_addr);
    end meu_table_word_proc;
```

9. Compiler errors: None. Compiler warnings:

```
************ cms2_toAda_predefined_b.a
************

459:   field_h_proc_x(float_to_bit(value),bstart,blength,dest_word);
A ------------------------^  
A:warning: RM 13.10.2(2): operand is bigger than target
479:   return bit_to_float(field_h_fcn_x(source_word,bstart,blength));
A ------------------------^  
A:warning: RM 13.10.2(2): operand is smaller than target
525:   meu_table_word_proc_x(float_to_cmos2word(value),
A ------------------------^  
A:warning: RM 13.10.2(2): operand is bigger than target
```

F-13
10. Link and bind. No errors.


    ==> calling TV10H_item_address_access_init
    ==> returning from TV10H_item_address_access_init
    ==> calling TV16D_item_address_access_init
    ==> returning from TV16D_item_address_access_init
    ==> calling Start
    ==> calling QA9AA
    ==> calling QA9AB
    ==> calling QA9AC
    ==> calling QA9AD
    ==> calling QA9AE
    ==> calling QTSYNOPS
    ==> calling CMS2_EXEC
    ==> done!
APL - REENGINEERING RECORD FOR GNAT COMPILER

1. Compilation
   gnatchop -s COMP
   sh COMP.sh -gnato

   A list of compilation errors is shown in Appendix A

2. Reengineering
   A list of compilation error fixes is shown in Appendix A.

3. Execute qa9qlook
   SUMMARY OF ERRORS

   EXECUTED - 345

   NO TESTS ACCOUNTED - 0

   EXECUTION ERRORS - 82

4. Execution errors all appear to be due to explicit conversion of a fixed or floating point exponent to an integer. Only integer exponents are available within the Ada 83 standard math operations. Access to other types of exponentiation operators will require access to a math library offering those capabilities. Instantiating the package Ada.Numerics.Generic_Elementary_Functions in Ada 95 which has the capabilities to handle floating point exponents solved the problem.

   with Ada.Numerics.Generic_Elementary_Functions;
   package ft is new Ada.Numerics.Generic_Elementary_Functions(Float);

5. Compilation, link and bind
   sh COMP.sh -gnato
   gnatmake qa9qlook
6. Execute qa9qlook

**SUMMARY OF ERRORS**

**EXECUTED**  -  345

**NO TESTS ACCOUNTED** -  0

**EXECUTION ERRORS** -  0

**APL - REENGINEER RECORD FOR SUN ADA COMPILER**

The GNAT compiled APL source code was taken as the starting point.

1. There is no standard math library for Ada 83, so attempt to use package Math from Verdixlib. Assume that the only operation required from the Math library is exponentiation with floating point exponent. Add the following line to the body.
   ```
   with math;
   use math;
   ```

2. Comment out the following lines from the GNAT code.
   ```
   --with ada.numerics.generic_elementary_functions;
   --package ft is new
   ada.numerics.generic_elementary_functions(float);
   --use ft
   ```

3. Compile the spec and body of basic_defns and qa9qlook.

4. Compile and link the driver.

5. Execute qa9qlook

**SUMMARY OF ERRORS**

**EXECUTED**  -  345

**NO TESTS ACCOUNTED** -  0

**EXECUTION ERRORS** -  0
APPENDIX G: PERSON-HOURS

This appendix contains person hours spent doing
- Preliminary tasks
- Quick Look tasks
- Stress Testing tasks
- Reengineering tasks
- Other tasks

Table G-1. Hours Performing Preliminary Tasks - 1

<table>
<thead>
<tr>
<th>TASK</th>
<th>HOURS</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Prepare / maintain plan</td>
<td>388</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Identify NRaD computers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. SPARC 10/ OS 4.1.3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>b. VAX 11/785 VMS 5.5-1</td>
<td>2</td>
<td>Reload accounts and set up access</td>
</tr>
<tr>
<td>c. PC MSDOS 6.22</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3. Identify, collect, install, and</td>
<td></td>
<td></td>
</tr>
<tr>
<td>learn CMS-2 source code analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tools (VAX &amp; PC)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. METRICS generator</td>
<td>1</td>
<td>Revision 6.2</td>
</tr>
<tr>
<td>b. DESIGN analyzer</td>
<td>1</td>
<td>Revision 6.1</td>
</tr>
<tr>
<td>TASK</td>
<td>HOURS</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>4. Identify collect, and install CMS-2 source files to be translated</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. MTASS QA files</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>b. ELF project</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>c. MK-2 project</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>d. S3-TMP project</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>e. SPY project</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>f. H60B project</td>
<td>33(^1)</td>
<td></td>
</tr>
<tr>
<td>5. Identify, collect, install, and learn Ada metrics tools</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. SLOC counter</td>
<td>6</td>
<td>Includes writing Ada line counter.</td>
</tr>
<tr>
<td>b. Logiscope</td>
<td>0</td>
<td>Already installed and learned</td>
</tr>
<tr>
<td>c. Ada-ASSURED</td>
<td>0</td>
<td>Already installed and learned</td>
</tr>
<tr>
<td>6. Install, obtain, and learn Ada compilers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. GNAT version 3.05</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>b. VAX version 2.2-38</td>
<td>1</td>
<td>reestablish compiler is up and available</td>
</tr>
<tr>
<td>c. Sun version 1.1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\) There were problems reading H60B tapes and with ftp transfers of H60B files.
<table>
<thead>
<tr>
<th>TASK</th>
<th>HOURS</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>7. APL translator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. Obtain and install</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>b. Learn/ receive training</td>
<td>14</td>
<td>Developer says all constructs translate</td>
</tr>
<tr>
<td>8. CCCC transformer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. Obtain and install</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>b. Learn/ receive training</td>
<td>39</td>
<td>Listed in user guide section 7</td>
</tr>
<tr>
<td>9. TRADA translator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. Obtain and install</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>b. Learn/ receive training</td>
<td>2</td>
<td>Listed in user manual section 3.8</td>
</tr>
<tr>
<td>TASK</td>
<td>HOURS</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>---------------------------------------------------------------------</td>
<td>-------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>10. Assembler Design Extractor (low to high level)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. Obtain and install</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>b. Learn/ receive training</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>11. Determine metrics to be collected during evaluation process</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>607</td>
<td>Hours for preliminary tasks</td>
</tr>
<tr>
<td>TASK</td>
<td>HOURS</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>a. Adapt QA9 to INCLUDE SYS_DD and TC directly</td>
<td>14</td>
<td>SYS-DD previously used as a compool, an the test controller, QTCON, added at link time.</td>
</tr>
<tr>
<td>b. MTASS compile, link, and execute</td>
<td>57</td>
<td>Reestablish QA testing COMmand files and logicals.</td>
</tr>
<tr>
<td>c. Analyze execution results</td>
<td>4</td>
<td>Executes in SIM43 - 346 tests, 20 expected errors in exponentiation section QA9A.</td>
</tr>
<tr>
<td>2. Gather CMS-2 source code metrics.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. Get SLOC, keywords &amp; complexity metrics</td>
<td>2</td>
<td>Used CMS-2 source code METRICS generator.</td>
</tr>
<tr>
<td>3. Translate to Ada</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. APL translator</td>
<td>&lt;1</td>
<td></td>
</tr>
<tr>
<td>b. CCCC transformer</td>
<td>4</td>
<td>SPYLOOP was used for CCCC and TRADA as a small sample before translating the much bigger QA9</td>
</tr>
<tr>
<td>c. TRADA translator</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4. Run Ada metrics generator for SLOC.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. APL translator</td>
<td>1</td>
<td>SLOCs may be seen in Figure A-1</td>
</tr>
<tr>
<td>b. CCCC transformer</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>c. TRADA translator</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>TASK</td>
<td>HOURS</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>5. Compile Ada samples produced by translators.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. APL compile by GNAT</td>
<td>&lt;1</td>
<td>These hours include times to prepare command files and compilation time</td>
</tr>
<tr>
<td>b. APL by Sun Ada</td>
<td>&lt;1</td>
<td></td>
</tr>
<tr>
<td>c. CCCC by GNAT Ada</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>d. CCCC by VAX Ada</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>e. TRADA by GNAT</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>f. TRADA by VAX</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6. Modify/ reengineer Ada as needed to achieve successful compile.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. APL compile by GNAT</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>b. APL by Sun Ada</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>c. CCCC by GNAT Ada</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>d. CCCC by SUNAda</td>
<td>1</td>
<td>CCCC transformer corrected to achieve clean Ada</td>
</tr>
<tr>
<td>e. CCCC by VAX</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>e. TRADA by GNAT</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>f. TRADA by Sun Ada</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>g. TRADA by VAX</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>TASK</td>
<td>HOURS</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>----------</td>
</tr>
</tbody>
</table>
| 7. Examine successfully compiled Ada code using Logiscope and Ada line counter. a. APL compile by GNAT | 13 | The Logiscope statistics (Halstead and McCabe) are only reported when using GNAT. These statistics are virtually identical for all three compilers. 
| b. APL by Sun Ada | <1 | 
| c. CCCC by GNAT Ada | 13 | 
| d. CCCC by Sun Ada | <1 | 
| e. CCCC by VAX Ada | <1 | 
| f. TRADA by GNAT | 13 | 
| g. TRADA by Sun Ada | <1 | 
| h. TRADA by VAX | <1 | 
| TOTAL | 150 | Hours for Quick Look tasks |
Table G-3. Hours Performing Stress Testing Tasks - 1

<table>
<thead>
<tr>
<th>TASK</th>
<th>HOURS</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Prepare CMS-2 test cases</td>
<td>8</td>
<td>All 84 QA files modified to use INCLUDE directive to include Test Controller (QTCOM &amp; SYSDD)</td>
</tr>
</tbody>
</table>

2. APL Translator
   a. Build COMmand file | 6 |
   b. Translate files | 5 |
   c. Gather metrics for translator failures | 8 |
   d. Compile gener. Ada VAX |
      Sun | 5 |
      GNAT | 4 |

Subtotal | 40 |

3. CCCC Transformer
   a. Build COMmand file | 30 | CCCC_STRESS.COM series |
   b. Translate files | 134 |
   c. Gather metrics for translator failures | 24 | supporting data for CCCC corrections |
   d. Compile gener. Ada VAX |
      Sun | 7 |
      GNAT | 6 |

Subtotal | 217 |
### Table G-3. Hours Performing Stress Testing Tasks - 2

<table>
<thead>
<tr>
<th>TASK</th>
<th>HOURS</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. TRADA Translator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. Build COMmand file</td>
<td>35</td>
<td>TRADA_STRESS.COM series, and shell scripts</td>
</tr>
<tr>
<td>b. Translate files</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>c. Gather metrics for</td>
<td>16</td>
<td>supporting data for TRADA corrections</td>
</tr>
<tr>
<td>translator failures</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d. Compile gener. Ada VAX</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>Sun</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>GNAT</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td><strong>Subtotal</strong></td>
<td>153</td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td>410</td>
<td>Hours for translator stress testing</td>
</tr>
<tr>
<td>TASK</td>
<td>HOURS</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>2. CMS-2 reengineering to get valid execution.</td>
<td>0</td>
<td>See Quick Look task 1</td>
</tr>
<tr>
<td>3. Translate CMS-2 sample.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. APL</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>b. CCCC</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>c. TRADA</td>
<td>2</td>
<td>Consolidate all single package files into 1 big file for easy compiling and transfers among host computers.</td>
</tr>
<tr>
<td>4. Reengineer Ada to get clean compile.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. APL by Sun Ada</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>b. APL by GNAT</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>c. CCCC by GNAT</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>d. CCCC by Sun Ada</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>e. TRADA by GNAT</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>f. TRADA by Sun Ada</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>g. TRADA by VAX Ada</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>TASK</td>
<td>HOURS</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>5. Redesign/rewrite QA9 in Ada 95</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>a. for APL</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>b. for CCCC</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>c. for TRADA</td>
<td>4</td>
<td>Ada Text_IO, Integer_IO, etc used in harness.</td>
</tr>
<tr>
<td>6. Reengineer Ada to get valid execution.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. APL by Sun Ada</td>
<td>1</td>
<td>Number in parenthesis is the time required to</td>
</tr>
<tr>
<td>b. APL by GNAT</td>
<td>18</td>
<td>fully implement exponentiation with a</td>
</tr>
<tr>
<td>c. CCCC by GNAT</td>
<td>2</td>
<td>floating point exponent</td>
</tr>
<tr>
<td>d. CCCC by Sun Ada</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>e. TRADA by GNAT</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>f. TRADA by Sun Ada</td>
<td>0 (6)</td>
<td></td>
</tr>
<tr>
<td>g. TRADA by VAX Ada</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7. Run Ada-ASSURED, Logiscope and SLOC counter</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>84</td>
<td>Hours performing Reengineering tasks</td>
</tr>
<tr>
<td>TASK</td>
<td>HOURS</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1. Consolidate metrics into graphs and tables.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. for Quick Look</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>b. for Stress Test</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>c. for Reengineering</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2. Write final report narrative.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. for Quick Look</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>b. for Stress Test</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>c. for Reengineering</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>d. for all other</td>
<td>284</td>
<td></td>
</tr>
<tr>
<td>3. Prepare and give status reports and presentations.</td>
<td>92</td>
<td>(status meeting w/ Colket and Chiara, Riegle and Mumm and FY 96 project review)</td>
</tr>
<tr>
<td>TOTAL</td>
<td>450</td>
<td>Hours for General Tasks and Final Report</td>
</tr>
</tbody>
</table>
PERSON-HOURS TO TRANSLATE QA9 SAMPLE

Tables G-6 and G-7 were used to calculate the total person-hours required to translate the CMS-2 QA9 sample to Ada. Table G-6 shows the person-hours spent in different phases of the translation process and includes total hours by translator. The hours are given when we used the Sun compiler. Less time was required with the GNAT compiler.

Table G-7 shows the person-hours required to translate 100 source lines of CMS-2 code for the QA9 sample. Person-hours per 100 SLOC are reported when counting SLOC as delimiting "S" and as lines counted by a text editor.

The reader should note the following:

1. The columns "Hours to achieve successful compilation" and "Hours to achieve successful execution" were obtained from Table C-3. For these columns, the Table C-3 Sun and GNAT hours were added together because the APL translated code was run through the GNAT compiler first and taken as the starting point when we used the Sun compiler. The same was done for the CCCC translated code.

2. Less learning and training time was required for the TRADA translator than the others. An NRaD software engineer who participated in the evaluation was already very familiar with the TRADA translator.

3. Person-hours are biased because of differences in the capabilities and experience of the people who worked on the evaluation. Different people worked with different translators and Ada compilers.

4. Less time would be required to translate QA9 today because of bug fixes by the translator developers.

5. The times shown in Table G-6 are only for transliteration. If plans are for translator produced Ada to be deployed and maintained then an additional phase is needed for Ada quality improvement. Examples of needed improvements include removal of GOTOs, removal of deal code, improved packaging, better information hiding, conformance to Ada quality and style guidelines, and other enhancements.

6. QA9 did not include IO to special devices, direct code, or overlays. The translation of CMS-2 software for actual systems will be considerably more time consuming.
<table>
<thead>
<tr>
<th></th>
<th>Obtaining and installing translator</th>
<th>Learning and training</th>
<th>Developing harness</th>
<th>Translating to Ada</th>
<th>Hours to achieve successful compilation</th>
<th>Hours to achieve successful execution</th>
<th>Total Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>APL</td>
<td>4 (tape)</td>
<td>14</td>
<td>2</td>
<td>1</td>
<td>9</td>
<td>19</td>
<td>49</td>
</tr>
<tr>
<td>CCCC</td>
<td>16 (tape)</td>
<td>39</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>10</td>
<td>68</td>
</tr>
<tr>
<td>TRADA</td>
<td>7 (electronic transfer)</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>Delimiting $ SLOC</td>
<td>Text editor lines SLOC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>APL</td>
<td>100(49/3568)= 1.37</td>
<td>100(49/4926)= .99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCCC</td>
<td>100(68/3568)= 1.91</td>
<td>100(68/4926)= 1.38</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRADA</td>
<td>100(22/3568)= .62</td>
<td>100(22/4926)= .45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX H : ADA 95 QA9: REENGINEERING A MIXED MODE MATH TEST IN ADA 95

The Ada 95 QA9 was developed to provide additional context in which to assess CMS-2 to Ada translation. The QA9 test suite was chosen for application redevelopment. Application redevelopment affords many opportunities for improvement due to requirement-level reengineering, exploiting modern language features, and design for reuse. By requirement-level reengineering we mean reconsidering functionality offered in a CMS-2 application and generating a design that provides the same functionality as well as meeting new requirements. In this case the new requirements were to minimize McCabe cyclomatic complexity and to maximize reuse.

The CMS-2 QA9 program tests accuracy of mathematical operations placing an emphasis on mixed-mode arithmetic. The QA9 application tests various combinations of integer, real, and fixed point operands and receptacles. The Ada 95 QA9 was designed to provide the same functionality in a more extensible way with very little control (McCabe) complexity. The functionality was provided by designing a class hierarchy of test cases which contains a total of 126 subclasses.

The number of test cases required is the product of

- 3 different kinds of receptacles (integer, real, fixed),
- 9 different operand pairs (integer, real, fixed ⇒ 3 left x 3 right for infix operations), and
- 5 different infix operations (+, -, /, *, **).

Since there is no exponentiation (**) operation for fixed point numbers, 9 (1*3*3) must be subtracted from 135 (9*3*5) to yield 126 subclasses.

Control complexity was minimized since the selection of which mathematical operation to execute and which combination of numeric representation and type conversion to use is performed by the Ada 95 run-time dispatcher for polymorphic operations. That is what allowed the implementation to achieve a weighted McCabe complexity metric of 1.1.

Figure H-1 is a graphical depiction of the Target (receptacle) object information and class structure. Each Target instance has a test case number (Num.), a result, lower and upper bounds on the answer, and a target of the operation. The test case number and result are inherited from the Target superclass. Each subclass has a different type for the bounds and operation target.

Figure H-2 is a graphical depiction of the (infix) Operation object information and class structure. It shows all 9 combinations of kinds of operand pairs.

Figure H-3 is a graphical depiction of the integer-based part of Test_Case object information and class structure. It shows that each test case has a Target, and Operation (operand combination), and a mathematical operation.

Figure H-4 is a graphical depiction of the real-based part of Test_Case object information and class structure. It shows that each test case has a Target, and Operation (operand combination), and a mathematical operation.
Figure H-1. Class Structure for Target Object

Figure H-5 is a graphical depiction of the fixed-based part of Test_Case object information and class structure. It shows that each test case has a Target, and Operation (operand combination), and a mathematical operation.

Given any leaf in the class structure tree, the meaning of the test case can be discerned from the name. For example, test case R_Test_Xi_M is has a real target, its left operand is fixed (X), its right operand is int (I) and it performs multiplication (M). Since the left operand is fixed, the right operand will be converted to fixed for the computation, and the result will be converted to the target type, real.
Figure H-2. Class Structure for the Operation Object
Figure H-3. Information Structure for the Integer-based Test_Case_Subclasses
Figure H-4. Information Structure for the Real-based Test_Case Subclasses
Figure H-5. Information Structure for the Fixed-based Test_Case Subclasses Fixed-based Test_Case Subclasses
APPENDIX I: ADA QUALITY AND STYLE CRITERIA

This appendix provides some additional information on the Ada quality and style produced by the translators. The questions were answered by members of the evaluation team who examined the Ada QA9s produced by the translators. Analysis tools were not used to answer these questions. An entry of "NC" (meaning not covered) in the table indicates that the criteria could not be measured by the QA9 sample.

<table>
<thead>
<tr>
<th>General Criteria</th>
<th>APL</th>
<th>CCCC</th>
<th>TRADA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Did the Ada code compile correctly?</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>COMMENTS: Answers to Table 1-1 were given by Ron Iwamiya</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. a. Were portions that are not translatable commented out?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>b. Did comments clearly indicate what is not translated?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. a. Did translator determine and produce typing that is more explicit than the CMS-2 types (e.g., integer, floating, character, etc.)?</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. a. Did translator produce records (for heterogeneous but related data), arrays, loops, blocks, constants, etc., when appropriate?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>b. Did it associate names with loops and blocks?</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>c. Were FOR loops rather than plain loops produced? (FOR loops are considered to be more maintainable.)</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table I-1. Ada Quality and Style Criteria - 1
<table>
<thead>
<tr>
<th>General Criteria</th>
<th>APL Y/N</th>
<th>CCC Y/N</th>
<th>TRADA Y/N</th>
</tr>
</thead>
<tbody>
<tr>
<td>5. Did translator produce GENERICS when appropriate?</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.a. Did code produced use UNCHECKED CONVERSIONS?</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>b. Is the use of UNCHECKED CONVERSIONS justified?</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. Did all mathematical functions translate?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. Could translator produce operators ABS, MOD, or REM?</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.a. Did translator produce exception handlers?</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>b. Did it produce shells for exception handlers that will handle predefined exceptions?</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMENTS: APL Translator provided one INDEX_OUT_OF RANGE exception</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maintainability</td>
<td>APL Y/N</td>
<td>CCC Y/N</td>
<td>TRADA Y/N</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------</td>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>1. Did translator decide what should go into package specifications versus bodies (e.g., variable/constant definitions, type definitions, subprogram definitions)?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2a. Did translator produce multiple packages in a way that logically carries forward structure from CMS-2 source code? (Desirable)</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>b. If not, did it produce one big package?</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMENTS: CCC produced one big file containing the package specification and body</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Did translator produce Ada GOTO statements?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>COMMENTS: Transfered from the CMS-2 code.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Are the variable names produced readable (e.g., do variable names produced resemble names in CMS-2 code? or Are they randomly produced)?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Did translator produce anonymous arrays?</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Was the Ada source code indented?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maintainability</td>
<td>APL Y/N</td>
<td>CCC Y/N</td>
<td>TRADA Y/N</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------</td>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>7.a. Were USE clauses always produced?</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>b. If not, were fully qualified names produced?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMENTS: TRADA is user selectable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. Did subprograms contain only one return statements?</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>COMMENTS: Some contained more than one.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.a. Did translator produce CASE statements?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>b. If so, did the CASE statement have an others clause?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. Are EQUALS and MEANS (CMS-2 constructs) translated into Ada in such a way that the Ada code is equally as easy to maintain as the CMS-2 code? (Question contributed by Dave Martin, Loral Federal Systems)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>COMMENT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. Did translator produce code that uses named association (e.g., in calls to subprograms, in generics, etc.)?</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12. Were CMS-2 comments preserved next to the appropriate Ada statements?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13. Did the translator produce multiple statements per line?</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maintainability</td>
<td>APL Y/N</td>
<td>CCC Y/N</td>
<td>TRADA Y/N</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------</td>
<td>---------</td>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>14. Were reserved words and other elements distinct from each other (i.e., reserved words may be lower case)?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.a. Did the translator produce one big file?</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>b. Multiple files?</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>c. A big file that can easily be broken up into individual files (such as pager format)?</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>d. Were specifications and bodies in different files?</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16. Was the use of the WITH clause minimized in the package specification?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17. For arrays, were attributes 'FIRST', 'LAST', 'LENGTH, or 'RANGE used instead of numeric literals?</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18. Were parentheses used in Ada to specify order of expression evaluation?</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19. Were BOOLEAN types produced?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Portability</td>
<td>APL Y/N</td>
<td>CCC Y/N</td>
<td>TRADA Y/N</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>1.a. Were types with range constraints or subtypes produced?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>1.b. Were types produced that have range constraints that are appropriate for the target computer?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Were MAX_INT, MAX_DIGITS, MIN_INT, MAX_MANTISSA used? (They should be avoided.)</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Were types INTEGER, LONG_INTEGER, SHORT_INTEGER, FLOAT, LONG_FLOAT, SHORT_FLOAT used?</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>APL</td>
<td>CCCC</td>
<td>TRADA</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------</td>
<td>-----</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>1. Were variables initialized when declared?</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td><strong>COMMENTS:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Were invariant objects declared as constants rather than variables?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>COMMENTS:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.a. Did translator figure out mode for subprogram parameters (e.g., in, out, in/out)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>b. Did it make everything in/out?</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td><strong>COMMENTS:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX J : ADA LINE COUNTER

ADA SOURCE FOR SLOC COUNTER (ASLOC)

The program below was written for this project to count delimiting semicolons, straight lines of
text, and comments for Ada source code.

-- Ada SLOC Counter
with Ada.Text_Io;
use Ada.Text_Io;
with Ada.Command_Line;
procedure Asloc is

package Acl renames Ada.Command_Line;

Unterminated_String : exception;
Invalid_Argument : exception;

Lines : Natural := 0;
Loc   : Natural := 0;
Cmt   : Natural := 0;

Echo  : Boolean := False;
Help  : Boolean := False;
Row   : Boolean := True;
Parms : Boolean := True;

File  : Natural := 0;
F     : File_Type;

subtype Length is Natural range 0 .. 512;
subtype Index is Length range 1 .. Length'last;
subtype Buffers is string(Index);

Len   : Length;
Idx   : Index;
Buffer : Buffers;
procedure Print is
begin
  Set_Col(1);
  if Echo then
    if File > 0 then
      Put(Acl.Argument(File));
    else
      Put("<standard_input>");
    end if;
  end if;
  if Row then
    Put_Line(Natural'image(Loc) & Natural'image(Cmt) & Natural'image(Lines));
  else
    Set_Col(1);
    Put("Ada LOC('';'");
    Set_Col(16);
    Put("Ada Comments");
    Set_Col(31);
    Put("Text Lines");
    Set_Col(1);
    Put(Natural'image(Loc));
    Set_Col(16);
    Put(Natural'image(Cmt));
    Set_Col(31);
    Put_Line(Natural'image(Lines));
  end if;
end Print;

procedure Get_Buff is
begin
  Lines := Lines + 1;
  Get_Line(Buffer, Len);
  Idx := 1;
end Get_Buff;

procedure Incr is
begin
  Idx := Idx + 1;
end Incr;
pragma Inline(Incr);

function In_String
return Boolean is
begin
  return Buffer(Idx) = '"';
end In_String;

procedure Check_Char_Literal is
begin
  if Len - Idx >= 2 and then Buffer(Idx+2) = '"' then
    Idx := Idx + 2;
  end if;
end Check_Char_Literal;

function Apostrophe
return Boolean is
begin
  return Buffer(Idx) = '"';
end Apostrophe;

J-2
procedure Find_End_String is
begin
  while Idx < Len loop
    Incr;
    if Buffer(Idx) = '"' then
      return;
    end if;
  end loop;
  raise Unterminated_String;
end Find_End_String;

function Eol
  return Boolean is
begin
  return Idx > Len;
end Eol;

function Comment
  return Boolean is
begin
  if Buffer(Idx) = '-' then
    if (Idx < Len) and then Buffer(Idx+1) = '-' then
      Cmt := Cmt + 1;
      return True;
    else
      return False;
    end if;
  else
    return False;
  end if;
end Comment;

function Left_Paren
  return Boolean is
begin
  return Buffer(Idx) = '(';
end Left_Paren;

procedure Skip_Right_Paren is
begin
  if notParms then
    Incr;
    loop
      while not Eol loop
        if Buffer(Idx) = ')' then
          return;
        end if;
      skip_Right_Paren;
    Incr;
    end loop;
    Get_Buff;
  end loop;
  end if;
end Skip_Right_Paren;
procedure Check_Semicolon is
begin
  if Buffer(Idx) = ';' then
    Loc := Loc + 1;
  end if;
end Check_Semicolon;

procedure Print_Help is
begin
  Set_Col(1);
  Put_Line(Acl.Command_Name & " input: [-h] [-r] [-e] [file_name]");
  Put_Line(" -v (off): verbose output format setting switch");
  Put_Line(" -e (off): echo filename switch");
  Put_Line(" -p (on): count ';' in parameter lists switch");
  Put_Line(" -h : print help switch");
  Put_Line(" filename is the input file. default is <standard_input>");
end Print_Help;

procedure Process_Arg
(N : in Positive) is
begin
  if Acl.Argument(N)(1) = '-' then
    if Acl.Argument(N)(2) = 'e' then
      Echo := not Echo;
    elsif Acl.Argument(N)(2) = 'p' then
     Parms := notParms;
    elsif Acl.Argument(N)(2) = 'v' then
      Row := Not Row;
    elsif (Acl.Argument(N)(2) = 'h') then
      Help := not Help;
    else
      raise Invalid_Argument;
    end if;
  else
    File := N;
  end if;
end Process_Arg;

procedure Set_Mode is
begin
  for This in 1 .. Acl.Argument_Count loop
    Process_Arg(This);
  end loop;
  if Help then
    File := 0;
    Echo := False;
  end if;
  if File > 0 then
    Open( File => F,
        Name => Acl.Argument(File),
        Mode => In_File);
    Set Input(F);
  end if;
end Set_Mode;
begin
    Set_Mode;
    if Help then
        Print_Help;
    else
        while not End_Of_File loop
            Get_Buff;
            Check_Line:
                while not Eol loop
                    if Comment then
                        exit Check_Line;
                    elsif In_String then
                        Find_End_String;
                    elsif Left_Paren then
                        Skip_Right_Paren;
                    elsif Apostrophe then
                        Check_Char_Literal;
                    else
                        Check_Semicolon;
                    end if;
                    Incr;
                end loop Check_Line;
        end loop;
        Print;
    end if;
exception
    when Invalid_Argument =>
        Print_Help;
    when others =>
        Put("Line:");
        Put_Line(Natural'image(Lines));
        raise;
end Asloc;
APPENDIX K: SAMPLE SOURCE CODE: QA9 PROCEDURE
QTSYNOPS CMS-2 AND TRANSLATOR PRODUCED ADA

This appendix contains source code for QTSYNOPS, one of the QA9 procedures translated
during Quick Look. The source code included is for CMS-2 QTSYNOPS and the Ada QTSYNOPS
produced by the three translators. The source code is included so that the reader can see how the
CMS-2 code translate. All of QA9 at various stages of the translation process is being made
available on the Web.

CMS-2 QTSYNOPS

CQT 0546  (EXTDEF)  PROCEDURE QTSYNOPS $
CQT 0547  COMMENT  PUT QA NUMBER IN HEADER $
CQT 0548  SET CHAR(28,4)(VHSYNHE) TO VMTESTNO $
CQT 0549  QTHEAD INPUT VHSYNHE $
CQT 0550  IF VINOTSTS LT 1 THEN RETURN$ 
CQT 0551  SET VHTEM TO H( ) ''TOP OF FORM CONTROL VRBL'' $
CQT 0552  LOOP.  VARY VSX2 THRU 4 $ 
CQT 0553  QTSYV1.  VARY VX1 THRU (VINOTSTS-1) $ 
CQT 0554  QTSYW4.  SET VX2 TO TAQR(VX1,ERRORNO) $ 
CQT 0555  QTSYN1.  IF VX2 EQ 0 THEN RESUME QTSYV1 $ 
CQT 0556  COMMENT  IF THE CODE IS 0 THEN NO TST IS EXPECTED.BYPASS MESSAGE $ 
CQT 0557  SET VIH1L TO VX1 $ 
CQT 0558  COMMENT SAVE LOOP INDEX$ 
CQT 0559  SET VX1 TO VSX1*5+VX1  ''COMPUTE TEST NO. FROM LOOP INDEX''$ 
CQT 0560  IF VSX2 NOT 0 THEN GOTO FAIL $ 
CQT 0561  IF VITESTYP EQ 0 THEN GOTO FAIL $ 
CQT 0562  COMMENT  ''MUST BE QA SO NO LIST OF TESTS PASSED NEEDED'' $ 
CQT 0563  IF VPASS THEN GOTO PASS$ 
CQT 0564  comment OUTPUT PRINT (VHASTER ,VHFOLLOW,VHPASS,VHASTER) FHEDSYN $ 
CQT 0565  SET VPASS TO 1 $ 
CQT 0566  PASS.  IF VX2 NOT 6 THEN GOTO FAIL $ 
CQT 0567  SET VX3 TO (VX1+1) + 1000*1(VITESTNO-10) $ 
CQT 0568  comment OUTPUT PRINT VX3 FPASS $ 
CQT 0569  COMMENT  PRINTS A LIST OF TESTS THAT PASSED $ 
CQT 0570  GOTO LOOPRES1$ 
CQT 0571  FAIL.  IF VSX2 NOT 1 THEN GOTO NOTEEXEC $ 
CQT 0572  IF VX2 EQ 7 THEN GOTO EXECED ''PRINT OUT HEADER IF FIRST $ 
CQT 0573  FAILURE IS A GENERATION ERROR '' $ 
CQT 0574  IF VX2 GT 5 AND VX2 LT 9D THEN QTCONSW USING VX2 THEN 
CQT 0575  ''RECORDS TESTS EXECUTED'' 
CQT 0576  GOTO LOOPRES1 $ 
CQT 0577  EXECED.  IF VEXEC THEN GOTO EXEC1 ''SKIP HEADER'' $ 
CQT 0578  IF VITESTYP EQ 1 ''QR TEST'' THEN SET VHTEM TO H(1) $ 
CQT 0579  comment OUTPUT PRINT VHTEM ''TOP OF FORM IF THIS IS A QR TEST'' $ 
CQT 0580  comment OUTPUT PRINT (VHASTER ,VHFOLLOW,VHFAIL,VHASTER) 
CQT 0581  comment OUTPUT PRINT (VHASTER ,VHFOLLOW,VHFAIL,VHASTER) 
CQT 0582  FHEDSYN $ 

K-1
CQT 0583  comment OUTPUT PRINT H(0) $
CQT 0584   SET VEXEC TO 1 $
CQT 0585   EXEC1.  QTCONSW USING VX2 ''PRINT OUT
CQT 0586            EXECUTION ERROR'' $
CQT 0587   GOTO LOOPRES1 $
CQT 0588   NOTEEXEC. IF VSX2 NOT 2 THEN GOTO NOTSKIP $
CQT 0589   IF VITESTYP EQ 0 THEN GOTO QTSYN2
CQT 0590   ''MUST BE QA TEST SO NO LIST OF SKIPPED TESTS NEEDED'' $
CQT 0591   IF VX2 NOT 30D THEN GOTO NOTSKIP $
CQT 0592   IF VSkip THEN GOTO SKIP $
CQT 0593   comment OUTPUT PRINT (VHASTER1,VHfollow,VHSKIP,VHASTER) FHEDSYN$
CQT 0594   comment OUTPUT PRINT H(0) $
CQT 0595   SET VSkip TO 1 $
CQT 0596   SKIP.  SET VX1 TO VITESTNO-10 $
CQT 0597   SET VX3 TO (VX1+1)+1000D*VAX1 $
CQT 0598   comment OUTPUT PRINT VX3 FPASS $
CQT 0599   COMMENT PRINTS A LIST OF TESTS THAT WERE SKIPPED (CODE 30) $
CQT 0600   GOTO LOOPRES1$
CQT 0601   NOTSKIP. IF VSX2 NOT 3 THEN GOTO NOTVIS $
CQT 0602   IF VX2 GT 13D THEN GOTO NOTVIS $
CQT 0603   IF VX2 LT 9D THEN GOTO LOOPRES $
CQT 0604   IF VITESTYP EQ 0 THEN GOTO QTSYN2
CQT 0605   ''THIS MUST BE A QA TEST SO NO VISUALS'' $
CQT 0606   IF VVIS THEN GOTO VISUAL $
CQT 0607   comment OUTPUT PRINT (VHASTER1,VHfollow,VHVISUAL,VHASTER)
CQT 0608   FHEDSYN $
CQT 0609   comment OUTPUT PRINT H(0) $
CQT 0610   SET VVIS TO 1 $
CQT 0611   VISUAL. QTERRD ''VISUAL TESTS PRINT OUT '' $
CQT 0612   GOTO LOOPRES1 $
CQT 0613   NOTVIS. IF VSX2 NOT 4 OR VX2 LT 6 OR(VX2
CQT 0614   GT 8D AND VX2 LT 14D) OR VX2 EQ 30D THEN GOTO LOOPRES $
CQT 0615   IF VITESTYP EQ 0 THEN GOTO QTSYN2
CQT 0616   ''THIS MUST BE A QA TEST SO NO SPECIALS'' $
CQT 0617   IF VSPEC THEN GOTO SPEC1 $
CQT 0618   comment OUTPUT PRINT (VHASTER1,VHfollow,VHSPEC,VHASTER)
CQT 0619   FHEDSYN $
CQT 0620   comment OUTPUT PRINT H(0) $
CQT 0621   SET VSPEC TO 1 $
CQT 0622   SPEC1. QTERRE ''ERROR CODES 14-29'' $
CQT 0623   COMMENT ((LINE* $
CQT 0624   LOOPRES1. SET TAO RTYP(VX2, TERRORCT) TO TAO RTYP(VX2, TERRORCT) +1$
CQT 0625   LOOPRES. SET VX1 TO VIHIL $
CQT 0626   END QTSYV1 $
CQT 0627   END LOOP $
CQT 0628   COMMENT PRINT OUT HEADER AND ALL TOTALS $
CQT 0629   QTSYN2. QTMES SW USING 4$  
CQT 0630   COMMENT PRINT OUT NUMBER OF STUBBED TESTS $
CQT 0631   IF STUBCNT NOT 0 THEN BEGIN $
CQT 0632   comment OUTPUT PRINT STUBCNT FORMSTUB $
CQT 0633   END $
CQT 0634   SET VEXEC, VVIS, VSPEC, VPASS, VSKIP TO 0 ''RESET FLAGS'' $
CQT 0635   comment OUTPUT PRINT H(A) ''CLEAR MAJOR HEADER AND TOP OF FORM''$
CQT 0636   RETURN $

K-2
procedure QTSYNOPS is
   begin
      vsynhed(29..32) := vmtestno ;
   end if;
   vhtemp := " " & c2a_blanks(1..19);
   top of FORM CONTROL VRBL
   <<LOOP_D>>
   for vsx2_x in 0 .. 4 loop
      <<QTSYV1>>
      vx1 := 0 ;
      while vx1 <= (vinotsts-1) loop
         <<QTSYW4>>
         vx2 := taqr(vx1).errorno ;
         <<QTSYN1>>
         if vx2 = 0 then
            goto QTSYV1_E ;
         end if;
      end while
      if the code is 0 then no test is expected. bypass message
      vhi1 := vx1 ;
   end if;
   save loop index
   vx1 := vsx1 * 5 + vx1 ;
   if vsx2_x /= 0 then
      goto FAIL ;
   end if;
   if vitestyp = 0 then
      goto FAIL ;
   end if;
   must be qa so no list of tests passed needed
   if vpass then
      goto PASS ;
   end if;
   output print (vhaster,vfollow,vhpass,vhaster) fhedsyn
   vpass := TRUE ;
   if vx2 /= 6 then
      goto FAIL ;
   end if;
   vix3 := (vx1 + 1) + 1000 * (vitestno - 10) ;
   output print vix3 fpass
   prints a list of tests that passed
   goto loopres1 ;
goto NOTEXEC ;  --
end if ;  --
if vx2 = 7 then  -- 1392 PRINT OUT HEADER IF FIRST
    goto EXECHED ;  --
end if ;  --
if vx2 > 5 and then vx2 < 9 then  -- 1394
    QTCNSW ( vx2 ) ;  --
-- 1395 RECORDS TESTS EXECUTED
    goto LOOPRES1 ;  -- 1396
end if ;  --
if vx2 > 5 then  -- 1397
    goto NOTEXEC ;  --
end if ;  --
<<EXECHED>>  -- 1398 SKIP HEADER
if vexec then  --
    goto EXECL ;  --
end if ;  --
if vitestyp = 1 then  -- 1399 QR TEST
    vhtemp := "1" & c2a_blanks(1..19) ;  --
end if ;  --
-- 1400 OUTPUT PRINT VHTEMP TOP OF FORM IF THIS IS A QR TEST
-- 1401 OUTPUT PRINT (VHASTER ,VHFOLLOW,VHFAIL,VHASTER) FHDYSN
-- 1402 FHDYSN
-- 1403 OUTPUT PRINT H(0)
    vexec := TRUE ;  -- 1404 PRINT OUT
<<EXECl>>  -- 1405 EXECUTION ERROR
    QTCNSW ( vx2 ) ;  --
    goto LOOPRES1 ;  -- 1407
<<NOTEXEC>>  -- 1408
if vsx2.x /= 2 then  --
    goto NOTSKIP ;  --
end if ;  --
if vitestyp = 0 then  -- 1409
    goto QTSYN2 ;  --
end if ;  --
-- 1410 MUST BE QA TEST SO NO LIST OF SKIPPED TESTS NEEDED
if vx2 /= 30 then  -- 1411
    goto NOTSKIP ;  --
end if ;  --
if vskip then  -- 1412
    goto SKIP ;  --
end if ;  --
-- 1413 OUTPUT PRINT (VHASTER1,VHFOLLOW,VHSKIP,VHASTER) FHDYSN
-- 1414 OUTPUT PRINT H(0)
    vskip := TRUE ;  -- 1415
<<SKIP>>  -- 1416
    vax1 := vitestno - 10 ;  --
    vix3 := ( vx1 + 1 ) + 1000 * vax1 ;  -- 1417
-- 1418 OUTPUT PRINT VIX3 FPASS
-- 1419 PRINTS A LIST OF TESTS THAT WERE SKIPPED (CODE 30)
    goto LOOPRES1 ;  -- 1420
<<NOTSKIP>>  -- 1421
    if vsx2.x /= 3 then  --
        goto NOTVIS ;  --
    
K-4
end if;
if vx2 > 13 then goto NOTVIS;
end if;
if vx2 < 9 then goto LOOPRES;
end if;
if vitestyp = 0 then goto QTSYN2;
end if;

-- 1425 THIS MUST BE A QA TEST SO NO VISUALS
if vvis then goto VISUAL;
end if;

-- 1427 OUTPUT PRINT (VHASTER1,VHOLLOW,VHVISUAL,VHASTER)
-- 1428 FHEDSYN
-- 1429 OUTPUT PRINT H(0)
  vvis := TRUE;
  <<VISUAL>>
  QTERRE;
  goto LOOPRES1;
  <<NOTVIS>>
  if vx2_x /= 4 or else vx2 < 6 or else (vx2 --
  > 8 and then vx2 < 14) or else vx2 = 30 then goto LOOPRES;
end if;
if vitestyp = 0 then goto QTSYN2;
end if;

-- 1436 THIS MUST BE A QA TEST SO NO SPECIALS
if vspec then goto SPEC1;
end if;

-- 1438 OUTPUT PRINT (VHASTER1,VHOLLOW,VHSPEC,VHASTER)
-- 1439 FHEDSYN
-- 1440 OUTPUT PRINT H(0)
vspec := TRUE;
<<SPEC1>>
QTERRE;
<<LOOPRES1>>
taqrtyt(vx2).terrorct := taqrtyt(vx2).terrorct + 1;
<<LOOPRES>>
vx1 := vih11;
<<QTSYV1_E>>
  vx1 := vx1 + 1;
end loop;
vsx2 := vsx2_x + 1;
end loop;

-- 1448 PRINT OUT HEADER AND ALL TOTALS
<<QTSYN2>>
QTMESWW (4);

-- 1450 PRINT OUT NUMBER OF STUBBED TESTS
if stucnt /= 0 then

-- 1452 OUTPUT PRINT STUCNT FORMSTUB

K-5
null; -- 1453
end if;  -- 1454
vexec := FALSE;  -- 1455
vvis := FALSE;  -- 1455
vspec := FALSE;  -- 1455
vpass := FALSE;  -- 1455
vskip := FALSE;  -- 1455

-- 1455 OUTPUT PRINT H(A) CLEAR MAJOR HEADER AND TOP OF FORM
return;  -- 1456
end QTSYNOPS;  -- 1457

-- ---- ----

CCCC GENERATED ADA QTSYNOPS

PROCEDURE QTSYNOPS IS
  -- PUT QA NUMBER IN HEADER
BEGIN
  ASSIGN_CHAR_SUBSTRING ( VHSYNHED.ALL.OVER, 28, 4, VMTESTNO.ALL.OVER );
  QTHEAD ( VHSYNHED.ALL.OVER );
  IF VINOTSTS.ALL.OVER<1 THEN
    RETURN;
  END IF;
  VHTEMP.ALL.OVER := PAD(" ", 20);
  << LOOP_0 >>
  VSX2.ALL.OVER := 1;
  WHILE (VSX2.ALL.OVER<=4) LOOP
    << QTSYV1 >>
    VX1.ALL.OVER := 1;
    WHILE (VX1.ALL.OVER<=(VINOTSTS.ALL.OVER-1)) LOOP
      << QTSYW4 >>
      VX2.ALL.OVER := FIELD_HFcn_INTEGER(TAQR_words.ALL(0, VX1.ALL.OVER ), 0, 8);
      << QTSYN1 >>
      IF VX2.ALL.OVER=0 THEN
        GOTO next_stmt_QTSYV1;
        -- IF THE CODE IS 0 THEN NO TST IS EXPECTED. BYPASS
      END IF;
      VIH1ALL.OVER := VX1.ALL.OVER;
      -- SAVE LOOP INDEX
      VX1.ALL.OVER := INTEGER(VSX1.ALL.OVER)*5+VX1.ALL.OVER;
      IF VSX2ALL.OVER/=0 THEN
        GOTO FAIL;
      END IF;
      IF VITESTYP.ALL.OVER=0 THEN
        GOTO FAIL;
        -- 'MUST BE QA SO NO LIST OF TESTS PASSED NEEDED'
      END IF;
END IF;

K-6
END IF;
IF int_to_bool(VPASS.ALL.OVER) THEN
  GOTO PASS;
  -- OUTPUT PRINT (VHASTER,VHOLLOW,VHFAIL,VHASTER)

END IF;
VPASS.ALL.OVER := 1;
<< PASS >>
IF VX2.ALL.OVER/=6 THEN
  GOTO FAIL;
END IF;

VIX3.ALL.OVER := (VX1.ALL.OVER+1)+1000*(VITESTNO.ALL.OVER-10);
-- OUTPUT PRINT VIX3 FPASS
-- PRINTS A LIST OF TESTS THAT PASSED
GOTO LOOPRES1;
<< FAIL >>
IF VSX2.ALL.OVER/=1 THEN
  GOTO NOTEXEC;
END IF;
IF VX2.ALL.OVER/=7 THEN
  GOTO EXECHED;
END IF;
IF VX2.ALL.OVER>5 AND VX2.ALL.OVER<9 THEN
  DECLARE
    QTCONSW_invalid : BOOLEAN;
  BEGIN
    QTCONSW ( VX2.ALL.OVER , QTCONSW_invalid )
  END;
  GOTO LOOPRES1;
END IF;
IF VX2.ALL.OVER>5 THEN
  GOTO NOTEXEC;
END IF;
<< EXECHED >>
IF int_to_bool(VEXEC.ALL.OVER) THEN
  GOTO EXEC1;
END IF;
IF VITESTyp.ALL.OVER=1 THEN
  --QR TEST
  VHTEMP.ALL.OVER := PAD("1",20);
  -- OUTPUT PRINT VHTEMP "TOP OF FORM IF THIS IS A QR TEST"
  -- OUTPUT PRINT (VHASTER,VHOLLOW,VHFAIL,VHASTER)
  -- FHEDS NY
  -- OUTPUT PRINT H(O)
END IF;

VEXEC.ALL.OVER := 1;
<< EXEC1 >>
DECLARE
  QTCONSW_invalid : BOOLEAN;
BEGIN
  QTCONSW ( VX2.ALL.OVER , QTCONSW_invalid )
END;
GOTO LOOPRES1;

K-7
< NOTEXEC >
IF VX2.ALL.OVER/=2 THEN
  GOTO NOTSKIP;
END IF;
IF VITESTYP.ALL.OVER=0 THEN
  GOTO QTSYN2;
END IF;
IF VX2.ALL.OVER/=30 THEN
  GOTO NOTSKIP;
END IF;
IF int_to_bool(VSKIP.ALL.OVER) THEN
  GOTO SKIP;
  --  OUTPUT PRINT (VHASTER1,VHFOLLOW,VHSKIP,VHASTER)
  --  OUTPUT PRINT H(0)
END IF;
VSKIP.ALL.OVER := 1;
< SKIP >
VAX1.ALL.OVER := fixed32s0(VITESTNO.ALL.OVER-10);
VIX3.ALL.OVER := integer((VX1.ALL.OVER+1)+float(1000*VAX1.ALL.
  OVER)) ;
  --  OUTPUT PRINT VIX3 PPASS
  --  PRINTS A LIST OF TESTS THAT WERE SKIPPED (CODE 30)
GOTO LOOPRES1;
< NOTSKIP >
IF VX2.ALL.OVER/=3 THEN
  GOTO NOTVIS;
END IF;
IF VX2.ALL.OVER>13 THEN
  GOTO NOTVIS;
END IF;
IF VX2.ALL.OVER<9 THEN
  GOTO LOOPRES;
END IF;
IF VITESTYP.ALL.OVER=0 THEN
  GOTO QTSYN2;
END IF;
IF int_to_bool(VVIS.ALL.OVER) THEN
  GOTO VISUAL;
  --  OUTPUT PRINT (VHASTER1,VHFOLLOW,VHVISUAL,VHASTER)
  --  FHEDSYN
  --  OUTPUT PRINT H(0)
END IF;
VVIS.ALL.OVER := 1;
< VISUAL >
QTERRD;
GOTO LOOPRES1;
< NOTVIS >
IF VX2.ALL.OVER/=4 OR VX2.ALL.OVER<6 OR (VX2.ALL.OVER>8 AND
  .ALL.OVER<14) OR VX2.ALL.OVER=30 THEN
  GOTO LOOPRES;
END IF;
IF VITESTYP.ALL.OVER=0 THEN

K-8
GOTO QTSYN2;
END IF;
IF int_to_bool(VSPEC.ALL.OVER) THEN
  GOTO SPEC1;
  --      OUTPUT PRINT (VHASTER1,VHFOLLOW,VHSPEC,VHASTER)
  --      FHEDSYN
  --      OUTPUT PRINT H(0)
END IF;
VSPEC.ALL.OVER := 1;
<< SPEC1 >>
QTERRE;
--      ((LINE*)
<< LOOPRES1 >>
FIELD_H_PROC_INTEGER ( FIELD_H.FCN_INTEGER(TAQRTPY_words.ALL(0,
   VX2.ALL.OVER),16,16)+1,16,16,TAQRTPY_words.ALL(0,VX2.ALL.OVER)
  );
<< LOOPRES >>
VX1.ALL.OVER := VIH1L.ALL.OVER;
<< next_stmt_QTSYV1 >>
VX1.ALL.OVER := VX1.ALL.OVER+1;
END LOOP;
<< next_stmt_LOOP_0 >>
VSX2.ALL.OVER := INTEGER(VSX2.ALL.OVER)+1;
END LOOP;
--      PRINT OUT HEADER AND ALL TOTALS
<< QTSYN2 >>
DECLARE
  QTMESSW_invalid : BOOLEAN;
BEGIN
  QTMESSW ( 4 , QTMESSW_invalid ) ;
END;
--      PRINT OUT NUMBER OF STUBBED TESTS
IF STUBCNT.ALL.OVER/=0 THEN
  NULL;  --      OUTPUT PRINT STUBCNT FORMSTUB
END IF;
VEXEC.ALL.OVER := 0;
--RESET FLAGS
VVIS.ALL.OVER := 0;
VSPEC.ALL.OVER := 0;
VPASS.ALL.OVER := 0;
VSKIP.ALL.OVER := 0;
--      OUTPUT PRINT H(A) 'CLEAR MAJOR HEADER AND TOP OF FORM'
RETURN;
END QTSYNOPS;

TRADA GENERATED ADA QTSYNOPS

PROCEDURE Qtsynops IS
Invalid_parameter : Boolean;

BEGIN -- QTSYNOPS

-- PUT QA NUMBER IN HEADER
Vhsynhed (29 .. 32) := Vmtstno;
Qthead (Vhead_input => Vhsynhed & "
IF Vinotsts < 1
THEN
    RETURN;
END IF;

Vhtemp := "
"; -- TOP OF FORM CONTROL VRBL
<< Loop_X >>
Vsx2 := 0;
LOOP
<< Qtsyv1 >>
Vx1 := 0;
LOOP

-----------
-- ERRorno is overlaid
-- CQT 0554 QTSYW4. SET VX2 TO TAQR(VX1,ERRorno) $
<< Qtsyw4 >>
Vx2 := Taqr (Vx1).Errorno;
<< Qtsyn1 >>
IF Vx2 = 0
THEN
    GOTO Qtsyv1_resume;
END IF;
-- IF THE CODE IS 0 THEN NO TST IS EXPECTED.BYPASS MESSAGE
Vih11 := Vx1;
-- SAVE LOOP INDEX
Vx1 := Vsx1 * 5 + Vx1; -- COMPUTE TEST NO. FROM LOOP INDEX
IF Vsx2 /= 0
THEN
    GOTO Fail;
END IF;
IF Vitestyp = 0
THEN
    GOTO Fail;
END IF;
-- 'MUST BE QA SO NO LIST OF TESTS PASSED NEEDED'
IF Vpass
THEN
    GOTO Pass;
END IF;
-- OUTPUT PRINT (VHASTER ,VHFOllOW,VHPASS,VHASTER) FHEDSYN
Vpass := True;
<< Pass >>
IF Vx2 /= 6
THEN
    GOTO Fail;
END IF;

K-10
Vix3 := Vx1 + 1 + 1000 * (Vitestno - 10);
-- OUTPUT PRINT VIX3 FPASS
-- PRINTS A LIST OF TESTS THAT PASSED
GOTO Loopres1;
<< Fail >>
IF Vx2 /= 1
THEN
  GOTO Notexec;
END IF;
IF Vx2 = 7
THEN
  GOTO Excehed;
  -- ^=== Embedded note(s):
  -- ''PRINT OUT HEADER IF FIRST
  -- FAILURE IS A GENERATION ERROR''
END IF;
IF Vx2 > 5 AND THEN Vx2 < 9
THEN
  Qtconsw (Vx2, Invalid_parameter);
  IF Invalid_parameter
  THEN
    RAISE Constraint_error;
  END IF;
  -- RECORDS TESTS EXECUTED
  GOTO Loopres1;
END IF;
IF Vx2 > 5
THEN
  GOTO Notexec;
END IF;
<< Excehed >>
IF Vexec
THEN
  GOTO Exce1;
  -- ^=== Embedded note(s): ''SKIP HEADER''
END IF;
IF Vitestyp = 1
THEN
  Vhtemp := "1"
;
END IF;
-- OUTPUT PRINT VHTEMP ''TOP OF FORM IF THIS IS A QR TEST''
-- OUTPUT PRINT (VHEADER, VHORDER, VHORDER, VHHEADER)
-- FHDERR
-- OUTPUT PRINT H(0)
Vexec := True;
<< Exce1 >>
Qtconsw (Vx2, Invalid_parameter);
IF Invalid_parameter
THEN
  RAISE Constraint_error;
END IF;
-- ^=== Embedded note(s):
-- ''PRINT OUT
-- EXECUTION ERROR''
GOTO Loopres1;
<< Notexec >>
IF Vx2 /= 2 THEN
   GOTO Notskip;
END IF;
IF Vistypep = 0 THEN
   GOTO Qtsyn2;
   -- "== Embedded note(s): 'MUST BE QA TEST SO NO LIST OF"
   -- "    SKIPPED TESTS NEEDED''
END IF;
IF Vx2 /= 30 THEN
   GOTO Notskip;
END IF;
IF Vskip THEN
   GOTO Skip;
END IF;
-- OUTPUT PRINT (VHASTER1,VHfollow,Vhskip,VHaster) PHEDSYN
-- OUTPUT PRINT H(0)
Vskip := True;
<< Skip >>
Vax1 := A_32_S_0 (Vitestno - 10);
Vix3 := I_32_S_0 (Vx1 + 1) + A_32_S_0 (1000 * Vax1));
-- OUTPUT PRINT VIX3 FFAS
-- PRINTS A LIST OF TESTS THAT WERE SKIPPED (CODE 30)
GOTO Loopres1;
<< Notskip >>
IF Vx2 /= 3 THEN
   GOTO Notvis;
END IF;
IF Vx2 > 13 THEN
   GOTO Notvis;
END IF;
IF Vx2 < 9 THEN
   GOTO Loopres;
END IF;
IF Vistypep = 0 THEN
   GOTO Qtsyn2;
   -- "== Embedded note(s): 'THIS MUST BE A QA TEST SO NO"
   -- "    VISUALS''
END IF;
IF Vvis THEN
   GOTO Visual;
END IF;
-- OUTPUT PRINT (VHASTER1,VHfollow,Vhvisual,VHaster)
-- PHEDSYN
-- OUTPUT PRINT H(0)
Vvis := True;
<< Visual >>
Gterrd;
-- ^== Embedded note(s): 'VISUAL TESTS PRINT OUT '
GOTO Loopres1;
<< Notvis >>
IF Vsx2 /= 4
  OR ELSE Vx2 < 6
  OR ELSE (Vx2 > 8 AND THEN Vx2 < 14)
  OR ELSE Vx2 = 30
THEN
  GOTO Loopres;
END IF;
IF Vitemp = 0
THEN
  GOTO Qtsyn2;
-- ^== Embedded note(s): 'THIS MUST BE A QA TEST SO NO -- SPECIALS'
END IF;
IF Vspec
THEN
  GOTO Spec1;
END IF;
-- OUTPUT PRINT (VHASTER1,VHFOLLOW,VHSPEC,VHASTER)
-- FHDGSYN
-- OUTPUT PRINT H(0)
Vspec := True;
<< Spec1 >>
Qterre;
-- ^== Embedded note(s): 'ERROR CODES 14-29'

--
******************************************************************************

<< Loopres1 >>
Taqrtype (Vx2).Terrorct := Taqrtype (Vx2).Terrorct + 1;
<< Loopres >>
Vx1 := Vih11;
<< Qtsyvl_resume >>
Vx1 := Vx1 + 1;
EXIT WHEN Vx1 > Vinots - 1;
END LOOP;
-- ^== Embedded note(s): 'QR TEST'
Vsx2 := Vsx2 + 1;
EXIT WHEN Vsx2 > 4;
END LOOP;
-- PRINT OUT HEADER AND ALL TOTALS
<< Qtsyn2 >>
Qtmessw (4, Invalid_parameter);
IF Invalid_parameter
THEN
  RAISE Constraint_error;
END IF;
-- PRINT OUT NUMBER OF STUBBED TESTS
IF Stubcnt /= 0 THEN
    NULL;
    -- ^=== Embedded note(s): 'OUTPUT PRINT STUBCNT FORMSTUB '
END IF;

Vskip := False; -- RESET FLAGS
Vpass := Vskip; -- RESET FLAGS
Vspec := Vpass; -- RESET FLAGS
Vvis := Vspec; -- RESET FLAGS
Vexec := Vvis; -- RESET FLAGS
-- OUTPUT PRINT H(A) 'CLEAR MAJOR HEADER AND TOP OF FORM'
RETURN;

END Qtsynops;
APPENDIX L : TRANSLATION ANALYSIS TOOLS

Table L-1 is a table that contains a description and points-of-contact for analysis tools used during the experiment in addition to the CMS-2 to Ada translators.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
<th>Point-of-Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ada-ASSURED</td>
<td>Checks for conformance to guidelines and can automatically make some changes to the code so that it conforms.</td>
<td>Jeffrey Burns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GrammaTech</td>
</tr>
<tr>
<td></td>
<td></td>
<td>One Hopkins Place</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ithaca, NY 14850</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(607) 273-7340</td>
</tr>
<tr>
<td>Ada SLOC Counter¹</td>
<td>Counts Ada source lines of code (:), Ada comments, and total lines.</td>
<td>Hans Mumm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NRaD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>53140 Systems St.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>San Diego, CA 92152</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(619)553-4004</td>
</tr>
<tr>
<td>Assembler Design Extractor (ADE)</td>
<td>Converts assembler to CMS-2</td>
<td>Jim O'Sullivan</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYNETICS Corporation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4485 Danube Drive, Suite 24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bayberry Office Park</td>
</tr>
<tr>
<td></td>
<td></td>
<td>King George, VA 22485</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(540)663-2137</td>
</tr>
<tr>
<td>CMS-2 Source Code Design Analyzer</td>
<td>Assists in the reengineering of CMS-2 code prior to translation to Ada.</td>
<td>Hans Mumm</td>
</tr>
<tr>
<td>(DESAN)</td>
<td>Identifies overlays, data units that are defined but not referenced, and</td>
<td>NRaD</td>
</tr>
<tr>
<td></td>
<td>data units that are referenced but not set to a value.</td>
<td>53140 Systems St.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>San Diego, CA 92152</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(619)553-4004</td>
</tr>
<tr>
<td>CMS-2 Source Code Metrics Generator</td>
<td>Produces source code statistics (e.g., SLOC for CMS-2 and direct code,</td>
<td>Hans Mumm</td>
</tr>
<tr>
<td>(METRC)</td>
<td>source statements in DDs and SYSPROCS), a keyword report, and Halstead and</td>
<td>NRaD</td>
</tr>
<tr>
<td></td>
<td>McCabe complexity metrics.</td>
<td>53140 Systems St.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>San Diego, CA 92152</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(619)553-4004</td>
</tr>
<tr>
<td>Logiscope</td>
<td>Produces many quality metrics from source code, including Halstead and</td>
<td>Dennis Andrews</td>
</tr>
<tr>
<td></td>
<td>McCabe measures, comments per lines of executable statements, mean SLOC for</td>
<td>Verilog</td>
</tr>
<tr>
<td></td>
<td>a subprogram, number of GOTO statements, number of returns in a subprogram</td>
<td>3010 LBJ Freeway</td>
</tr>
<tr>
<td></td>
<td>and others. A CMS-2 Logiscope capability is available from Verilog.</td>
<td>Suite 900</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dallas, TX 75234</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(800)424-3095, x24</td>
</tr>
</tbody>
</table>

¹ SLOC count is provided in Appendix J.
Table L-2 is a table that contains a description and points-of-contact for analysis tools that are potentially useful to a project that translates source code from CMS-2 to Ada.

**Table L-2. Description and POCs for Potentially Useful Analysis Tools - 1**

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
<th>Point-of-Contact</th>
</tr>
</thead>
</table>
| AdaMat          | Provides detailed information on the maintainability, portability, and reliability of Ada source. | Chris McGuire  
Dynamics Research Corporation  
60 Frontage Road  
Andover, MA 01810  
(508)475-9090, x1730 |
| CLUE            | Prototype CMS-2 reverse engineering tool that produces data flow diagrams, control flow diagrams and reports to assist the programmer in understanding CMS-2 source code. | Suzy Roberts  
Mitre Corporation  
Clue@mitre.org  
202 Burlington Road  
Mail Stop K329  
Bedford MA 01730  
(617)271-8963 |
| HyperBook       | Facilitates the analysis of program documentation, specifically source code. The tool facilitates software understanding and maintenance. Software is analyzed to produce a documentation database. The database is browsed from UNIX or PC workstations on a network by using programs written in Java. | Noah Prywes  
Computer Command and Control Company  
2300 Chestnut Street  
Suite 230  
Philadelphia, PA 199103  
(215)854-0555 |
| Logiscope CMS-2 | Produces many quality metrics from CMS-2 source code, including Halstead and McCabe measures, comments per lines of executable statements, mean SLOC for a subprogram, number of GOTO statements, number of returns in a subprogram and others. A CMS-2 Logiscope capability is available from Verilog. | Dennis Andrews  
Verilog  
3010 LBJ Freeway  
Suite 900  
Dallas, TX 75234  
(800)424-3095, x24 |
<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
<th>Point-of-Contact</th>
</tr>
</thead>
</table>
| Object Abstractor    | Assists in making translated Ada higher quality. It includes a capability to convert non object oriented Ada to object oriented Ada in a semi-automated manner. | Romel Rivera
Xinotech Research Incorporated
1313 Fifth Street Southeast
Suite 213
Minneapolis, MN 55414
(612)379-3844 |
| Pretty printers      | Makes the Ada source code more readable and maintainable.                    | For pretty printers in the Public Ada Library (PAL)
http://wuarchive.wustl.edu/languages/ada/ |
| Reengineering Toolkit| Aids software engineers in restructuring existing Ada source code. The restructuring facilitates readability and maintainability. This toolset is especially useful when source code is reused or translated from another language into Ada. | Kevin McQuown
Rational
3963 Via Holgura
San Diego, CA 92130
(619)794-6801 |
APPENDIX M : MK-2 CMS-2L AND ADA SOURCE CODE

This appendix contains CMS-2L and Ada source code for the NAVSEA project, Combat Control System MK-2 Fire Control System. This software computes target location information. The CMS-2L code contains no direct code.

The CMS-2L code was translated by the APL, CCCC, and TRADA translators. The APL translator produced some Ada statements, was incomplete, and did not compile. The CCCC translator produced code that compiled and executed. The TRADA translator produced no Ada source code. For purposes of comparison, the CMS-2L code was also translated to Ada by hand. The hand version included some re-engineering. These artifacts are provided as sections of this appendix.

- Original CMS-2L MK-2 Fire Control System
- Ada Translation Using APL Translator
- APL Translator Predefined Packages
- Ada Translation Using CCCC Translator
- CCCC Translator Predefined Packages
- Ada Reengineering of MK-2 Code by Hand

The Ada Code Reengineering of MK-2 code produced by hand represents the final desired product from the reengineering of CMS-2 Code. In this regard, it is useful as a benchmark for comparison.

Of the two successful translations both were problematic.

- The CCCC translation was successful in that it compiled correctly. Unfortunately, the code produced did not use the features of Ada that facilitate code maintenance or reengineering, but rather used features undesirable in a mission-critical, safety-critical application. If any reengineering or code evolution is required, it would be far better to perform a manual translation from the CMS-2 than to use any of the CCCC generated translated output. On the other hand, the CCCC translator could be extremely useful in translating code where that code would be integrated into a modern Ada environment, unchanged. This could be a legitimate requirement for many applications. However, this approach is not recommended should there ever be a desire to evolve or reengineer the code.

- The APL translation did not generate compilable code. In fact the 100+ additional comments represent areas the APL translator could not translate. However, most of these comments represented code where manual intervention is really desirable in order to produce higher quality translated code. In a sense, the APL translator could be used as an effective tool in supporting an engineer in the reengineering of the CMS-2 code into Ada.

Basically, the output of the CCCC translation could be used as is with minimal modifications but could not be easily reengineered; the output of the APL translator would require significant work resulting in a reasonably engineered translation. Any translated product would require additional reengineering in order to evolve the code with new requirements. Comparisons between the hand generated code and the translated code are made in the following areas:
- Source Code Lines of Code (SLOC)
- Naming Conventions
- Elimination of Intermediate Variables
- Use of Standard Packages
- Memory Management
- Performance
- Position to Reengineer

**SOURCE CODE LINES OF CODE (SLOC)**

Table M-1 provides the SLOC counts for the MK-2 source code.

<table>
<thead>
<tr>
<th></th>
<th>Lines of text</th>
<th>(Delimiting $ or ;)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMS-2L MK-2 Code</td>
<td>298</td>
<td>205</td>
<td>178/204¹</td>
</tr>
<tr>
<td>APL Ada</td>
<td>374</td>
<td>97</td>
<td>274</td>
</tr>
<tr>
<td>APL Basic_Defns</td>
<td>642</td>
<td>317</td>
<td>165</td>
</tr>
<tr>
<td>APL Total</td>
<td>1016</td>
<td>414</td>
<td>439</td>
</tr>
<tr>
<td>CCCC Ada</td>
<td>936</td>
<td>454</td>
<td>175</td>
</tr>
<tr>
<td>CCCC pre_defined</td>
<td>1305</td>
<td>1305</td>
<td>0</td>
</tr>
<tr>
<td>CCCC Total</td>
<td>2241</td>
<td>1759</td>
<td>175</td>
</tr>
<tr>
<td>TRADA Ada</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>TRADA</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>TRADA Total</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Hand translation</td>
<td>288</td>
<td>99</td>
<td>132</td>
</tr>
</tbody>
</table>

It should be noted that the hand translation contains about 50% SLOC compared to the original CMS-2L code.

¹ The first number represents the number of informational comments while the second is the number of lines of text.
NAMING CONVENTIONS

The original CMS-2L MK-2 code used cryptic 8 letter naming conventions. Ada translations require meaningful names to facilitate understanding of the code. Automatic name conversion is not possible. The last page M-54 of the Hand reengineered Ada code contains mappings from CMS-2 identifiers to Ada 95 identifiers. Tools to support automatic name conversion throughout all system packages are highly desirable.

ELIMINATION OF INTERMEDIATE VARIABLES

Intermediate variables are used extensively in CMS-2. In Ada, their use is avoided. For example, to compute latitude, Ada might use the statement:

\[
\text{Latitude := Arcsin (Sin(Lat)*Cos(Theta) + Cos(Lat)*Sin(Theta)*Cos(Brg))};
\]

In CMS-2, one would typically break the statement into a number of intermediate statements with locally declared variables. The data definitions would appear as:

```ada
LOCBL sub-dd $
  varl TEMPARG f $ "interim value for arcsin" $
  varl COSTHET f $ "Cosine R/Re" $
  varl SINTHET f $ "Sin R/Re" $
  varl COSLAT1 f $ "Cosine LAT1" $
  varl SINGLAT1 f $ "Sin LAT1" $
  varl COSBRG f $ "Cosine BRG" $
  varl SINGBRG f $ "Sin BRG" $
  end-sub-dd LOCBL $
```

And the intermediate statements might appear as:

```ada
set SINGLAT1 to SIN(LAT) $
set COSTHET to COS(THETA) $
set COSLAT1 to COS(LAT) $
set SINTHET to SIN(THETA) $
set COSBRG to COS(BRG) $
set TEMPARG to SINGLAT1*COSTHET+CSOSLAT1*SINTHET*COSBRG$ 
set LATITUD to ASIN(TEMPARG) $
```

Such intermediate statements are used extensively in CMS-2 as a means to provide code optimization to improve performance. In the MK-2 example, SINGLAT1, COSTHET, COSLAT1, SINTHET, and COSBRG are also used for the computation of longitude. Hence the intermediate variable would eliminate the additional costly computation. In Ada, such a breakdown is counterproductive as a good optimizing compiler would recognize the opportunity to optimize the code and perform the optimization automatically.

The elimination of intermediate variables is one of the reasons why the code translated by hand is approximately 50% of the original CMS-2L. These extra intermediate forms contributed to

M-3
complicating the translated CCCC. Unfortunately, a translator is not capable of eliminating the intermediate variables. Translators simply converts existing CMS-2 code to Ada. A manual conversion is desirable after the code translation. Normal text editing tools are quite satisfactory for this transformation. The last page page M-54 of the Hand reengineered Ada code identifies the intermediate variables that were not required.

The APL translator handled intermediate variables in an interesting way. In CMS-2, intermediate variables are typically coded as SUB-DDs or LOCRBLLs instead of SYS-DDs. Instead of making the translation, the APL translator generated an error message, thus pointing out a situation where the intermediate variable should be eliminated. For example, the "vrbl COSLAT1 ff" statement above was flagged as an error in the Ada "$ \sim \text{vrblcoslat1f} -- 366" comment. This facilitated the reengineering of the code, but resulted in an output which would require a manual reengineering.

USE OF STANDARD PACKAGES

One might expect a translator to take advantage of the standard Ada packages such as Ada.Numerics and Ada.Calendar. This was not done by any of the translators. Yet this is something desirable for the reengineering of any application. Both of these packages were used in the manual translation.

Both CCCC and APL used a package to facilitate the mapping of CMS-2 constructs to Ada. The APL package was called Basic_Defns and the CCCC package was called pre_defined. Each package provided its own math package. At the time the translators were developed, a standard Ada math package did not exist. Ada95 now has Ada.Numerics.

CCCC uses a pre_defined specification (536 SLOC) and body (769 SLOC) to facilitate the mapping of CMS-2 constructs to Ada. Both the pre_defined.ads and pre_defined.adb are required by the CCCC translated Ada code. Only a small portion of this code was actually needed by the CCCC Ada MK-2 code. However, the total SLOC required was 1,759, higher by an order of magnitude than any other alternative.

These translator packages might be useful in facilitating a translation that can compile and execute, but in the long run should be removed. Any serious code reengineering activity would want to eliminate dependencies on these translator supplied packages. The packages hinder code understanding and may not be portable for all environments.

MEMORY MANAGEMENT

Modern memory management is typically performed either using stack or heap mechanisms. Stack mechanisms are default for objects and their operations. Stacks can grow or shrink as memory is required. Heap mechanisms are evoked using Ada access types with operations on these types. Garbage collection is typically required to reclaim unused heap memory.

CMS-2 uses a fixed memory management with overlays. Depending on the overlay, an different objects can be mapped to the same location. This primitive memory mechanism creates serious translation problems. For example, the CMS-2L statement for own ship longitude:

\[ \text{VRBL SUDVOSLN F P \(-120.0\ast (FKPI2/360.0)\) } \]$
Could possibly be translated to:

```plaintext
subtype Sudvosln_type is Float;
Sudvosln : Sudvosln_Type := -120.0*fkpi2/360.0 ;
```

Which might be reengineered to:

```plaintext
subtype Longitude is Float range -180.0 .. + 180.0;
Own_Ship_Longitude: Longitude := -120.0*2*PI/360.0;
```

Had good CMS-2 programming practices been used this translation would be effective.

However, memory was a serious constraint on many CMS-2 systems. As a solution, overlays were used, thus providing a single memory location with multiple declarations. Unfortunately, CMS-2 programmers also frequently used undesirable side-effects with the overlays. For example, all assignments to the value of SUDVOSLN should be of the form: “set SUDVOSLN to something$” - However, if the overlay mapped LONG to the same address, the value of SUDVOSLN could be easily changed through: “set LONG to somethingelse$.” This side-effect saved the additional instruction of: “set SUDVOSLN to LONG$.” Hence, top rated CMS-2 programmers prided themselves in the ability to optimize CMS-2 code through the use of side-effects. In the mid 1980s, this practice was viewed as extremely dangerous. Hence this problem is pervasive legacy CMS-2 code. In the MK-2 code used for this comparison which was developed in the late 1980s-early to 1990s, overlays were not used.

APL and TRADA took the approach that side-effects would not be considered in the translation. Hence users would have to test the translated code for possible side effects, an additional burden on the developer as many side-effects are subtle and hard to find.

As the use of “side-effects” was a common practice, CCCC took the approach of using heap memory with access types. Hence when an overlay was used, the access types could point to the same memory address and the side-effect would be captured. To the credit of CCCC, their translation mechanism was the only one to correctly translate and execute the MK-2 example.

Unfortunately the price for this correction is high. The translated code is extremely difficult to understand and modify, requires many extra statements, and requires heap memory management. CCCC translated the above CMS-2L statement to:
TYPE SUDVOSLN_item_type IS
   RECORD
      OVER : FLOAT := (-120.0)*(FKPI2/360.0);
   END RECORD;

TYPE SUDVOSLN_item_pointer IS ACCESS SUDVOSLN_item_type;
TYPE SUDVOSLN_one_type IS ARRAY (0..0) OF cms2_word;
TYPE SUDVOSLN_one_pointer IS ACCESS SUDVOSLN_one_type;

FUNCTION SUDVOSLN_item_address_access IS NEW UNCHECKED_CONVERSION
   (SOURCE=>ADDRESS,TARGET=>SUDVOSLN_item_pointer);
SUDVOSLN : SUDVOSLN_item_pointer := SUDVOSLN_item_address_access
   (SUDVOSLN_memory'ADDRESS);
FUNCTION SUDVOSLN_one_address_access IS NEW UNCHECKED_CONVERSION
   (SOURCE => ADDRESS, TARGET => SUDVOSLN_one_pointer);
SUDVOSLN_one : SUDVOSLN_one_pointer :=
SUDVOSLN_one_address_access
   (SUDVOSLN_memory'ADDRESS);

The use of access types seems to complicate code unnecessarily. Also the use of the generic
Unchecked_Conversion is not desirable and potentially extremely dangerous. It also explains why
the CCC translation is an order of magnitude larger than alternative translation methods. The use of
access types and Unchecked_Conversion are clearly undesirable from a code readability and
understandability perspective. The CCC code is not useful to evolve the system should later
changes be desired.

The access type forcing heap memory management is NOT recommended for mission-
critical/safety-critical systems. Heaps are dangerous and impact performance when garbage
collection must be performed to re-acquire unused blocks of memory. Stacks are more easily
controlled as stack elements are created and destroyed as practical. Further, stacks are safer than
heaps because when a heap is exceeded, the system crashes; when a stack is exceeded, only the task
owning the stack is effected. Code could terminate the task and reinitialize the task. In practice, safe
stack sizes can be engineered for any system where recursion is not used. Safe heaps are almost
impossible to manage/control.

PERFORMANCE

Performance was not measured for any of the translations. However, some comments can be
made based on the different approaches used by CCC and APL. Neither the stack nor the heap
memory management scheme has a significant performance advantage. Memory management on the
stack is controlled as the stack is used; memory management on the heap must be performed when
the heap runs out of space or periodically using a process called garbage collection. As noted, the
CCC code is an order of magnitude larger using the Unchecked_Conversion function pervasively.
This extra code does not add a burden for execution. Both the CCC and APL when compiled
without optimization should execute at about the same speed. As most compilers have fine-tuned
optimizations for stack processing compared to heap processing, the APL translated code would be expected to execute significantly faster than the CCCC translated code, when both are optimized.

POSITION TO REENGINEER

One motivation to translate code might be to reengineer the code for an evolved system. The APL Ada Code appears to support this objective. The CCCC translated code appears to violate the reasons for using Ada. It would be significantly easier to reengineer the original CMS-2 code than the translated CCCC Ada. The use of CCCC translated code could be counterproductive to evolving a CMS-2 application to an Ada application.

Subsequent sections contain the source code for the MK-2 CMS-2L, the MK-2 Ada produced by the translators, and the MK-2 Ada that was manually translated.
ORIGINAL CMS-2L MK-2 FIRE CONTROL SYSTEM

MK2 SYSTEM $
COMMENT THIS CMS2 SYSTEM CONTAINS ONE SYS-DD (SYSD) AND
ONE SYS-PROC (SYSP) $
END-HEADS
SYSD SYS-DD $

FKPI EQUALS 3.1416  " constant PI " $
FKPI2 EQUALS 2*FKPI  " constant 2*PI " $

VRBL SUVTIMEl F P O  " current system time in sec" $
VRBL ICMX I 32 S P 1  " table index " $
VRBL SUDVOSXP F P O  " own ship x-position in yards " $
VRBL SUDVOSYP F P O  " own ship y-position in yards " $
VRBL SUDVRAD1 F P O  " x-position diff, in yards " $
VRBL SUDVRAD2 F P O  " y-position diff, in yards " $

TABLE FTCNDAT V 1 99 $
FIELD FVEQRADG A 32 S 4 P 6975563.33 " earth radius in yards"$
END-TABLE FTCNDAT $

TABLE FTCSS V 5 99  " system solution table " $
FIELD FVTIME F P 0  " solution update time " $
FIELD FVTX F P 0  " X position in yards " $
FIELD FVTY F P 0  " Y position in yards " $
FIELD FVTXV F P 0  " X velocity in yards/sec " $
FIELD FVTYY F P 0  " Y velocity in yards/sec " $
END-TABLE FTCSS $

TABLE FTPKSS V 6 99  " PK system solution table " $
FIELD FVTXP F P 0  " PKed target X position in yards " $
FIELD FVTYP F P 0  " PKed target Y position in yards " $
FIELD FVTRG F P 0  " PKed target range in yards " $
FIELD FVTRG F P 0  " PKed target bearing in radians " $
FIELD FVTGTL0 F P 0  " PKed target latitude " $
FIELD FVTLONG F P 0  " PKed target longitude " $
END-TABLE FTPKSS $

VRBL SUDVOSL T F P 32.0*(FKPI2/360.0) " own ship latitude"$
VRBL SUDVOSL T F P -120.0*(FKPI2/360.0) " own ship longitude"$
VRBL SUDVRG F  " (parameter) range " $
VRBL SUDVRG F  " (parameter) bearing " $
VRBL SUDVLT1 F  " (parameter) input latitude " $
VRBL SUDVLT2 F  " (parameter) output latitude " $
VRBL SUDVL1 F  " (parameter) input longitude " $
VRBL SUDVL2 F  " (parameter) output longitude " $
VRBL (VRAD1,VRAD2) F  " (parameter) two ATAN arguments " $

END-SYS-DD SYSD $

SYSP SYS-PROC $

FUNCTION SUDPATAN (VRAD1,VRAD2) F $ $
SUB-DO $ $
VRBL VATAN F $ $
END-SUB-DO $ $
if VRAD1 LT 0.00001 AND VRAD2 LT 0.00001 THEN $ $
SET VATAN TO 0.0 $ $
ELSE $ $
SET VATAN TO ATAN2(VRAD1,VRAD2) $ $
RETURN (VATAN) $ $
END-FUNCTION SUDPATAN $

M-8
PROCEDURE SUDPFC $

COMMENT

COMMENT Segment: FCS $
COMMENT CSCI Name: TMAB $ 
COMMENT TCLCS: SUD $ 
COMMENT LLGSC: SUDLTD $ 
COMMENT UNIT: SUDPFC $ 
COMMENT Part Number PRG528777 $ 
COMMENT Classification: UNCLASSIFIED $ 
COMMENT Company_ID Raytheon, CAGE Code 49956 $ 

COMMENT $ 

COMMENT Library Name MK2ECP6:[SRC.FC.TMAB.SUD.SRC] $ 
COMMENT Element Name SUDPFC$SRC $ 
COMMENT Revision Number 1 $ 
COMMENT Revision Date, Time 25-NOV-1992 10:57 $ 
COMMENT Current Date, Time 3-MAR-1995 16:44 $ 

COMMENT $ 

COMMENT Author: Mark Damiani $ 

COMMENT $ 

COMMENT Overview: This purpose of this procedure is to perform $ 
COMMENT the following for all FCS tactical/training $ 
COMMENT targets not including OTH targets: $ 
COMMENT 1) Compute PKed Target X Position. $ 
COMMENT 2) Compute PKed Target Y Position. $ 
COMMENT 3) Compute PKed Target Range $ 
COMMENT 4) Compute PKed Target Bearing $ 
COMMENT 5) Compute PKed Target Latitude and Longitude $ 
COMMENT by calling the SUDPRLL system common $ 
COMMENT routine. $ 

COMMENT $ 

COMMENT Effects: $ 

COMMENT $ 

COMMENT Requirements Trace: $ 

COMMENT $ 

COMMENT Algorithm: $ 

COMMENT $ 

COMMENT Notes: This procedure will be called during a SUD Time $ 
COMMENT Dependent entrance. $ 

COMMENT $ 

COMMENT Exceptions Raised: $ 

COMMENT $ 

COMMENT $ 

sudloc1 sub-dd "Unit Local Data" $ 

vrl1 SUDVTIME f "Target Solution PK Delta Time"$ 
VRL1 GTGLAT f "PKed Target Latitude"$ 
vrl1 GTGLONG f "PKed Target Longitude"$ 

end-sub-dd sudloc1 "End Unit Local Data"$ 

M-9
--- Comment: Compute FCS Position Kept Target X and Y Positions ---
- Compute FCS Position Kept Target X and Y Positions
---

--- Comment: Set Target Solution Delta Time to current System Time minus System Solution table Solution Update Time for current ICN. ---
set SUDVDTME to SUDVTME - FTCSS(ICNX, FVTIME) $ 

--- Comment: Compute FCS PK Target X Position. ---
set FTPKSS(ICNX, FVTXP) to FTCSS(ICNX, FVTXP) + (FTCSS(ICNX, FVTXX) * SUDVDTME) $

--- Comment: Compute FCS PK Target Y Position. ---
set FTPKSS(ICNX, FVTYP) to FTCSS(ICNX, FVTYP) + (FTCSS(ICNX, FVTYY) * SUDVDTME) $

--- Comment: Compute FCS Position Kept Target Range. ---
- Compute FCS Position Kept Target Range.
---

set FTPKSS(ICNX, FVRNG) to SQRT((FTPKSS(ICNX, FVTXP) - SUDVOSXP) * (FTPKSS(ICNX, FVTXP) - SUDVOSXP) + (FTPKSS(ICNX, FVTYP) - SUDVOSYP) * (FTPKSS(ICNX, FVTYP) - SUDVOSYP))$

if FTPKSS(ICNX, FVRNG) gt 999999 then
  set FTPKSS(ICNX, FVRNG) to 999999$ "Clip target range to MAX"

--- Comment: Compute FCS Position Kept Target Bearing. ---
- Compute FCS Position Kept Target Bearing.
---

set SUDVRAD1 to FTPKSS(ICNX, FVTXP) - SUDVOSXP$
set SUDVRAD2 to FTPKSS(ICNX, FVTYP) - SUDVOSYP$
set FTPKSS(ICNX, FVRNG) to SUDPATAN(SUDVRAD1, SUDVRAD2)$

--- Comment: PKed Target Latitude and PKed Target Longitude shall be computed using the Range, Azimuth to Latitude,Longitude (SUDPRLDL) common conversion function.
Input parameters shall include current Own Ship Latitude and Own Ship Longitude, PKed Target Range, and PKed Target Bearing.
Output parameters shall be PKed Target Latitude and PKed Target Longitude. ---
- PKed Target Latitude and PKed Target Longitude shall be computed using the Range, Azimuth to Latitude,Longitude (SUDPRLDL) common conversion function.
Input parameters shall include current Own Ship Latitude and Own Ship Longitude, PKed Target Range, and PKed Target Bearing.
Output parameters shall be PKed Target Latitude and PKed Target Longitude.
---

set SUDVRNG to FTPKSS(ICNX, FVRNG)$ "convt RNG to a 43 Float"
set SUDVRBG to FTPKSS(ICNX, FVRBG)$ "convt BRG to a 43 Float"

SUDPRLDL input SUDVRNG, SUDVRBG, SUDVOSLT, SUDVOSLN
OUTPUT TGLTLAT, TGLTONG$

--- Comment: Save PKed Target Latitude in PK System Solution table. ---
set FTPKSS(ICNX, FVTGLAT) to TGLTLAT $

--- Comment: Save PKed Target Longitude in PK System Solution table. ---
set FTPKSS(ICNX, FVTGLON) to TGLTONG $
end-proc SUDPKFCS $

(EXTDEF) PROCEDURE SUDPRBLL input SUDVRNG, SUDVBRG, SUDVLAT1, SUDVLCN1
output SUDVLAT2, SUDVLCN2 $

COMMENT $---------------------------------------------$
COMMENT $---------------------------------------------$
COMMENT Segment: FCS $
COMMENT CSCI Name: TMAB $
COMMENT TLCS: SUD $
COMMENT LLCS: SUDLTD $
COMMENT UNIT: SUDPRBLL $
COMMENT Part Number: PRG528777 $
COMMENT Classification: UNCLASSIFIED $
COMMENT Company_ID: Raytheon, CAGE Code 49956 $
COMMENT $---------------------------------------------$
COMMENT $---------------------------------------------$
COMMENT Library Name: MK2ECF6:[SRC.FC.TMAB.SUD.SRC] $
COMMENT Element Name: SUDPRBLL.SRC $
COMMENT Revision Number: 2 $
COMMENT Revision Date, Time: 27-APR-1993 16:28 $
COMMENT Current Date, Time: 3-MAR-1995 16:44 $
COMMENT $---------------------------------------------$
COMMENT $---------------------------------------------$
COMMENT Author: Jim Pryor (JRP), Bill Croasdale (WKC) $
COMMENT Overview: $
COMMENT The Range/Bearing to Lat/Lon unit will calculate the latitude and longitude coordinates of a position represented by a range, bearing from the input latitude/longitude position. $
COMMENT $---------------------------------------------$
COMMENT $---------------------------------------------$
COMMENT Effects: $
COMMENT $---------------------------------------------$
COMMENT Requirements Trace: PROCESS_NAV $
COMMENT $---------------------------------------------$
COMMENT Algorithm: $
COMMENT theta = R/RE $
COMMENT Target Latitude = $ 
COMMENT Arctan[sin(PO) * cos(theta) + $ 
COMMENT cos(PO) * sint(theta) * cos(By)] $
COMMENT $ 
COMMENT Target Longitude = $ 
COMMENT arctan2[sin(theta) * sin(By), $ 
COMMENT cos(PO) * cos(theta) - $ 
COMMENT sin(PO) * sin(theta) * cos(By)] + U0 $
COMMENT R = Range to target from input Lat/Lon (yds) $
COMMENT By = Bearing to target from input Lat/Lon $ 
COMMENT PO = input Latitude $ 
COMMENT U0 = input Longitude $ 
COMMENT RE = Radius of the earth (from FTCNDAT) $
COMMENT Notes: All angles (input/output) in floating point Radians, $
COMMENT and all ranges in floating point yards. $
COMMENT $---------------------------------------------$
COMMENT Exception Raised: $
COMMENT $---------------------------------------------$
LOCBL sub-dd $
vrbl RBLTHET f "'interim value (R/REO' "$
vrbl TEMPFAR9 f "'interim value for arcsin' "$

M-11
vrbl COSTHET f $ 'Cosine R_Re'
vrbl SINTHET f $ 'Sin R_Re'
vrbl COSLAT1 f $ 'Cosine LAT1'
vrbl SINALAT1 f $ 'Sin LAT1'
vrbl COSBRG f $ 'Cosine BRG'
vrbl SINBRG f $ 'Sin BRG'

end-sub-dd LOCRBLL $

''-------------------------------------------------------------''
'' Compute Theta = Target Range / Radius of Earth ''
''-------------------------------------------------------------''
set RBLTHET to SUDVRNG / FTCONDAT(0,FVEQRAGD) $

''-------------------------------------------------------------''
'' Save some CFU - Precompute SIN/COS terms ''
''-------------------------------------------------------------''
set COSTHET to COS(RBLTHET)$ 'Cosine R_Re'
set SINTHET to SIN(RBLTHET)$ 'Sin R_Re'
set COSLAT1 to COS(SUDVLAT1)$ 'Cosine LAT1'
set SINALAT1 to SIN(SUDVLAT1)$ 'Sin LAT1'
set COSBRG to COS(SUDVBRG)$ 'Cosine BRG'
set SINBRG to SIN(SUDVBRG)$ 'Sin BRG'

''-------------------------------------------------------------''
'' Compute Latitude of Target ''
''-------------------------------------------------------------''
set TEMPARG to SINALAT1 * COSTHET + COSLAT1 * SINTHET * COSBRG $
set SUDVLAT2 to ASIN(TEMPARG) $

''-------------------------------------------------------------''
'' Compute Longitude of Target''
''-------------------------------------------------------------''
set SUDVLON2 to SUDPATAN(SINTHET * SINBRG, 
COSLAT1 * COSTHET - 
SINALAT1 * SINTHET * COSBRG) + SUDVLON1 $

if SUDVLON2 gt FKPI then set SUDVLON2 to SUDVLON2 - FKPI2$
'Bound LON to (-PI,PI)'

END-PROC SUDPRBLLS

END-SYS-PROC SYSP $
END-SYSTEM MK2 $
ADA TRANSLATION USING APL TRANSLATOR

with Basic_Defns;
use Basic_Defns;

package Mk2 is

FKPI : constant FLOAT := 3.1416;
FKPI2 : constant FLOAT := 2 * fkpi;
sudtime : FLOAT := 0.0;
icnx : INTEGER32 := 1;
sudvosxp : FLOAT := 0.0;
sudvosyp : FLOAT := 0.0;
sudvrad1 : FLOAT := 0.0;
sudvrad2 : FLOAT := 0.0;

type FTCONDAT_REC is record
    fveqradg : FLOAT;
end record;

type FTCONDAT_TYPE is array (INTEGER range <>) of FTCONDAT_REC;
ftcondat : FTCONDAT_TYPE (0 .. 98) :=
    (0 => (fveqradg=>69755563.33),
     1 .. 98 => (fveqradg=>0.0));

type FTCSS_REC is record
    fvtimc : FLOAT;
    fvtimx : FLOAT;
    fvtimy : FLOAT;
    fvtimv : FLOAT;
end record;

type FTCSS_TYPE is array (INTEGER range <>) of FTCSS_REC;
ftcss : FTCSS_TYPE (0 .. 98) :=
    (0 .. 98 => (fvtimc=>0.0, fvtimx=>0.0, fvtimy=>0.0,
                  fvtimv=>0.0, fvtimv=>0.0));

type FTPKSS_REC is record
    fvtimx : FLOAT;
    fvtimy : FLOAT;
    fvtimv : FLOAT;
    fvtimu : FLOAT;
end record;

type FTPKSS_TYPE is array (INTEGER range <>) of FTPKSS_REC;
ftpkss : FTPKSS_TYPE (0 .. 98) :=
    (0 .. 98 => (fvtimx=>0.0, fvtimy=>0.0, fvtimu=>0.0,
                  fvtimu=>0.0, fvtimu=>0.0, fvtimu=>0.0));
sudvoslt : FLOAT := 32.0;
sudvosln : FLOAT := -120.0;
sudvrg : FLOAT;
sudvbrg : FLOAT;
sudvlat1 : FLOAT;
sudvlat2 : FLOAT;
sudvlon1 : FLOAT;
sudvlon2 : FLOAT;
vrad1 : FLOAT;
vrad2 : FLOAT;

--------------------------------------------------------------------------------
-- S U D P K F C S
--
-- Description:
--
--------------------------------------------------------------------------------

procedure SUDPKFCS;
procedure SUDPRELL (sudvrg : in   FLOAT;
sudvbrg : in   FLOAT;
sudvlat1 : in   FLOAT;
sudvlon1 : in   FLOAT;
sudvlat2 : out FLOAT;
sudvlon2 : out FLOAT);

end Mk2;with Basic_Defns;
use Basic_Defs;
with Mathpac;

package body Mk2 is

--------------------------------------------------------------------------------------------------------------------------
--- S U D P A T A N
---
--- Description:
---
--------------------------------------------------------------------------------------------------------------------------

function SUDPATAN (vrad1 : in FLOAT;

vrad2 : in FLOAT) return FLOAT is

-- MK2 SYSTEM ; -- 1
-- END-HEAD ; -- 4
-- SYSD SYS-DD ; -- 5
-- END-SYS-DD SYSD ; -- 49
-- SYSP SYS-PROC ; -- 51
--0@ could not translate:
--0@ dd
--0@1

function SUDPATAN(vrad1 : in FLOAT;

vrad2 : in FLOAT) return FLOAT is --

begin

--- SUB - -dd ; -- 54
---@ could not translate:
---@ vrblvatanf
---013
--- $$ -vrblvatanf ; -- 55
--- $$ END - SUB - DD ; -- 56
---@ could not translate:
---@ vatan
---@717

if vrad1 < 0.00001 and then vrad2 < 0.00001 then -- 57
= 0.0 ; -- 58
---@ could not translate:
---@ vatan
---@ could not translate:
---@ atan2
---@20

else;

= -atan2(vrad1,vrad2) ; -- 59

end if;

---@ could not translate:
---@ vatan
---@23

return ( vatan ) ;

end SUDPATAN ;

---@ could not translate:
---@ sudiclsub
---@ could not translate:
---@ dd
---@30

procedure SUDPKPCS is

begin

--- $$ -sudiclsub - -dd ; -- 60
---@ could not translate:
---@ vrblsudvdtmef
---032
--- $$ -vrblsudvdtmef ; -- 62
---@ could not translate:
---@ vrbgttlaf
---@34
--- $$ -vrbgttlaf ; -- 63
---@ could not translate:
---@ vrblgtlongf

M-15
-- #36
-- $ $ -vrb1gtlongf ; -- 164
-- # could not translate:
-- # end
-- # could not translate:
-- # sub
-- # could not translate:
-- # ddsudloc1
-- #38
-- $ $ -end - -sub - -ddsdudloc1 ; -- 166
-- # could not translate:
-- # sudvdtme
-- #40
- sudvdtme := sudvtime - ftcss(icnx).ftvtime ; -- 178
-- #43: could not typecast r.h.s. of assignment.
-- # Unknown name.
-- # could not translate:
-- # sudvdtme
-- #44
ftpks(icnx).fvtxp := ftcss(icnx).ftvxp + -- 182
( ftcss(icnx).ftvxv * -sudvdtme ) ; -- 183
-- #47: could not typecast r.h.s. of assignment.
-- # Unknown name.
-- # could not translate:
-- # sudvdtme
-- #48
ftpks(icnx).ftvyp := ftcss(icnx).ftvxp + -- 188
( ftcss(icnx).ftvypv * -sudvdtme ) ; -- 189
( ftpks(icnx).ftvxp - sudvosxp ) + -- 190
( ftpks(icnx).ftvyp - sudvosyp ) * -- 191
( ftpks(icnx).ftvyp - sudvosyp ) ) ; -- 200
if ftpks(icnx).fvrng > 999999 then -- 203
ftpks(icnx).fvrng := 999999.0 ; -- 204
end if ;
--
sudvrad1 := ftpks(icnx).ftvxp - sudvosxp ; -- 210
sudvrad2 := ftpks(icnx).ftvxp - sudvosyp ; -- 211
ftpks(icnx).fvrng := SDRPTAN ( sudvrad1, sudvrad2 ) ; -- 213
sudvrng := ftpks(icnx).fvrng ; -- 220
sudvbrg := ftpks(icnx).fvrng ; -- 229
-- #63 could not typecast parameter list.
-- # Unknown name.
-- # could not translate:
-- # tglat
-- # could not translate:
-- # tlong
-- #64
SUDPRBLL ( sudvrng, sudvbrg, sudvoslt, sudvosln, ~tglat, ~ ~
tglong ) ; -- 232
-- #66: could not typecast r.h.s. of assignment.
-- # Unknown name.
-- # could not translate:
-- # tglat
-- #67
ftpks(icnx).ftvgtlat := ~tglat ; -- 236
-- #69: could not typecast r.h.s. of assignment.
-- # Unknown name.
-- # could not translate:
-- # tlong
-- #70
ftpks(icnx).ftvglon := ~tglong ; -- 241
end SUDPRFCS ; -- 244
-- ----- ----- --

procedure SUDPRBLL(sudvrng : in FLOAT;
sudvbrg : in FLOAT;
sudvlat1 : in FLOAT;
sudvlon1 : in FLOAT;
sudvlon2 : out FLOAT;
sudvlon2 : out FLOAT) is
sudvlon2_t : FLOAT ; --
-- # could not translate:
-- # locbrhsub
-- # could not translate:
-- # dd
-- #84

M-16
```
begin
-- $ -locrbllsub -dd ; -- 248
-0 = could not translate:
-0 vrblhltetf
-0886
-- $ -vrblhltetf ; -- 361
-0 = could not translate:
-0 vrblmpargf
-0888
-- $ -vrblmpargf ; -- 362
-0 = could not translate:
-0 vrblcosthetf
-0890
-- $ -vrblcosthetf ; -- 364
-0 = could not translate:
-0 vrblsinthetf
-0892
-- $ -vrblsinthetf ; -- 365
-0 = could not translate:
-0 vrblcoslatf
-0894
-- $ -vrblcoslatf ; -- 366
-0 = could not translate:
-0 vrblsinlatf
-0896
-- $ -vrblsinlatf ; -- 367
-0 = could not translate:
-0 vrblcosbrgf
-0898
-- $ -vrblcosbrgf ; -- 368
-0 = could not translate:
-0 vrblsinbrgf
-0100
-- $ -vrblsinbrgf ; -- 369
-0 = could not translate:
-0 end
-0 = could not translate:
-0 sub
-0 = could not translate:
-0 ddlocrbll
-0102
-- $ -end - sub - -ddlocrbll ; -- 371
-0 = could not translate:
-0 rrblhltet
-0104
-xblhltet := svdverf / ftcondat(0).fveqcdgs ; -- 380
-0106 could not typecast parameter list.
-0 Unknown name.
-0 = could not translate:
-0 = costhet
-0 = could not translate:
-0 rrblhltet
-0107
-coslatl := Mathpac.Cos ( -rvblhltet ) ; -- 386
-0109 could not typecast parameter list.
-0 Unknown name.
-0 = could not translate:
-0 = sinlatl
-0 = could not translate:
-0 rrblhltet
-0110
-sinlatl := Mathpac.Sin ( -rvblhltet ) ; -- 387
-0 = could not translate:
-0 coslatl
-0112
-cosbrgl := Mathpac.Cos ( svdverf ) ; -- 389
-0 = could not translate:
-0 sinlatl
-0114
-sinlatl := Mathpac.Sin ( svdverf ) ; -- 390
-0 = could not translate:
-0 cosbrg
-0116
-cosbrg := Mathpac.Cos ( svdverf ) ; -- 392
-0 = could not translate:
-0 sinbrg
```

M-17
-- @116
-sinbrcg := Mathpac.Sin ( sudvbrg ) ;
-- @0
could not translate:
-- @0	temparg
-- @0
could not translate:
-- @0	sinlat1
-- @0
could not translate:
-- @0
costhet
-- @0
could not translate:
-- @0
coslal1
-- @0
could not translate:
-- @0	sinhet
-- @0
could not translate:
-- @0
cosbrcg
-- @120
-temparg := -sinlat1 * -costhet + -coslat1 * -sinhet * -cosbrcg ;
-- @122: could not typecast r.h.s. of assignment.
-- @0
Unknown name.
-- @123 could not typecast parameter list.
-- @0
Unknown name.
-- @0
could not translate:
-- @0
temparg
-- @124
sudvlat2 := Mathpac.Asin ( -temparg ) ;
-- @128: could not typecast r.h.s. of assignment.
-- @0
Unknown name.
-- @129 could not typecast parameter list.
-- @0
Unknown name.
-- @0
could not translate:
-- @0	sinhet
-- @0
could not translate:
-- @0	inbrcg
-- @0
could not translate:
-- @0
coslal1
-- @0
could not translate:
-- @0
costhet
-- @0
could not translate:
-- @0	sinlat1
-- @0
could not translate:
-- @0	sinhet
-- @0
could not translate:
-- @0
cosbrcg
-- @130
sudvlon2_t := SUDPATAN ( -sinhet * -sinbrcg , -- @0
-coslal1 * -costhet -
-sinlat1 * -sinhet * -cosbrcg ) + sudvlon1 ;
-- @0
if sudvlon2 > fkpi then
-- @0
sudvlon2_t := sudvlon2_t - fkpi2 ;
--
end if ;
--
sudvlon2 := sudvlon2_t ;
-- @0
end SUDPATL1 ;
--
----- -----
--
-- END-SYS-PROC SYSP ;
-- @0
end Mx2 ;
-- @0
APL TRANSLATOR COMMON PACKAGES

with System;
with UNCHECKED_CONVERSION;
package Basic_Defs is

-- Unsigned INTEGER types.

subtype INTEGERU1 is INTEGER range 0 .. 1;
subtype INTEGERU2 is INTEGER range 0 .. 3;
subtype INTEGERU3 is INTEGER range 0 .. 7;
subtype INTEGERU4 is INTEGER range 0 .. 15;
subtype INTEGERU5 is INTEGER range 0 .. 31;
subtype INTEGERU6 is INTEGER range 0 .. 63;
subtype INTEGERU7 is INTEGER range 0 .. 127;
subtype INTEGERU8 is INTEGER range 0 .. 255;
subtype INTEGERU9 is INTEGER range 0 .. 511;
subtype INTEGERU10 is INTEGER range 0 .. 1023;
subtype INTEGERU11 is INTEGER range 0 .. 2047;
subtype INTEGERU12 is INTEGER range 0 .. 4095;
subtype INTEGERU13 is INTEGER range 0 .. 8191;
subtype INTEGERU14 is INTEGER range 0 .. 16_383;
subtype INTEGERU15 is INTEGER range 0 .. 32_767;
subtype INTEGERU16 is INTEGER range 0 .. 65_535;
subtype INTEGERU17 is INTEGER range 0 .. 131_071;
subtype INTEGERU18 is INTEGER range 0 .. 262_143;
subtype INTEGERU19 is INTEGER range 0 .. 524_287;
subtype INTEGERU20 is INTEGER range 0 .. 1_048_575;
subtype INTEGERU21 is INTEGER range 0 .. 2_097_151;
subtype INTEGERU22 is INTEGER range 0 .. 4_194_303;
subtype INTEGERU23 is INTEGER range 0 .. 8_388_608;
subtype INTEGERU24 is INTEGER range 0 .. 16_777_216;
subtype INTEGERU25 is INTEGER range 0 .. 33_554_431;
subtype INTEGERU26 is INTEGER range 0 .. 67_108_863;
subtype INTEGERU27 is INTEGER range 0 .. 134_217_728;
subtype INTEGERU28 is INTEGER range 0 .. 268_435_456;
subtype INTEGERU29 is INTEGER range 0 .. 536_870_912;
subtype INTEGERU30 is INTEGER range 0 .. 1_073_741_824;
subtype INTEGERU31 is INTEGER range 0 .. 2_147_483_647;
-- INTEGER32 should be range 0 .. 4_294_967_296, but
-- since Ada reserves the sign bit for its own use, and
-- integers are a maximum of 4 bytes on the Verdix
-- compiler, INTEGER32 will have the same definition
-- as INTEGER31.
subtype INTEGER32 is INTEGER range 0 .. 2_147_483_647;

--
-- Signed INTEGER types.
--
subtype INTEGERS2 is INTEGER range -1 .. 1;
subtype INTEGERS3 is INTEGER range -3 .. 3;
subtype INTEGERS4 is INTEGER range -7 .. 7;
subtype INTEGERS5 is INTEGER range -15 .. 15;
subtype INTEGERS6 is INTEGER range -31 .. 31;
subtype INTEGERS7 is INTEGER range -63 .. 63;
subtype INTEGERS8 is INTEGER range -127 .. 127;
subtype INTEGERS9 is INTEGER range -255 .. 255;
subtype INTEGERS10 is INTEGER range -511 .. 511;
subtype INTEGERS11 is INTEGER range -1023 .. 1023;
subtype INTEGERS12 is INTEGER range -2047 .. 2047;
subtype INTEGERS13 is INTEGER range -4095 .. 4095;
subtype INTEGERS14 is INTEGER range -8191 .. 8191;
subtype INTEGERS15 is INTEGER range -16_383 .. 16_383;
subtype INTEGERS16 is INTEGER range -32_767 .. 32_767;
subtype INTEGERS17 is INTEGER range -65_535 .. 65_535;
subtype INTEGERS18 is INTEGER range -131_071 .. 131_071;
subtype INTEGERS19 is INTEGER range -262_143 .. 262_143;
subtype INTEGERS20 is INTEGER range -524_287 .. 524_287;
subtype INTEGERS21 is INTEGER range -1_048_575 .. 1_048_575;
subtype INTEGERS22 is INTEGER range -2_097_151 .. 2_097_151;
subtype INTEGERS23 is INTEGER range -4_194_303 .. 4_194_303;
subtype INTEGERS24 is INTEGER range -8_388_608 .. 8_388_608;
subtype INTEGERS25 is INTEGER range -16_777_215 .. 16_777_215;
subtype INTEGERS26 is INTEGER range -33_554_431 .. 33_554_431;
subtype INTEGERS27 is INTEGER range -67_108_863 .. 67_108_863;
subtype INTEGERS28 is INTEGER range -134_217_727 .. 134_217_727;
subtype INTEGERS29 is INTEGER range -268_435_455 .. 268_435_455;
subtype INTEGERS30 is INTEGER range -536_870_911 .. 536_870_911;
subtype INTEGERS31 is INTEGER range -1_073_741_823 .. 1_073_741_823;
subtype INTEGERS32 is INTEGER range -2_147_483_647 .. 2_147_483_647;
-- INTEGERS64 should be range -(2**64)+1 .. (2**64)-1, but
-- integers are a maximum of 4 bytes on the VerdiX
-- compiler, so INTEGERS64 will have the same definition
-- as INTEGERS32.
subtype INTEGERS64 is INTEGER range -2_147_483_647 .. 2_147_483_647;

-- Fixed point definitions.
-- type FIXED is delta (1/2_147_483_647);

--
-- Used for tables with no storage type.
--

type WORD_ARRAY is array (INTEGER range <>) of INTEGERS32;

--
-- Used for simulating INVALID option on F-SWITCH calls.
--
INDEX_OUT_OF_RANGE : exception;

--
-- Some useful conversion functions to take care of
-- CORAD's.
--
function INT_TO_ADDR is new
  UNCHECKED_CONVERSION (INTEGER, System.ADDRESS);

function ADDR_TO_INT is new
  UNCHECKED_CONVERSION (System.ADDRESS, INTEGER);

--
-- Some useful functions to eliminate the need for
-- as many type conversions.
--
function "+" (LEFT: in INTEGER; RIGHT: in FLOAT) return FLOAT;
function "+" (LEFT: in FLOAT; RIGHT: in INTEGER) return FLOAT;
function "+" (LEFT: in INTEGER; RIGHT: in FLOAT) return FLOAT;
function "+" (LEFT: in FLOAT; RIGHT: in INTEGER) return FLOAT;
function "+" (LEFT: in INTEGER; RIGHT: in FLOAT) return FLOAT;
function "+" (LEFT: in FLOAT; RIGHT: in INTEGER) return FLOAT;
function "*" (LEFT: in INTEGER; RIGHT: in FLOAT) return FLOAT;
function "*" (LEFT: in FLOAT; RIGHT: in INTEGER) return FLOAT;
function "/" (LEFT: in INTEGER; RIGHT: in FLOAT) return FLOAT;
function "/" (LEFT: in FLOAT; RIGHT: in INTEGER) return FLOAT;
function "<" (LEFT: in INTEGER; RIGHT: in FLOAT) return BOOLEAN;
function "<" (LEFT: in FLOAT; RIGHT: in INTEGER) return BOOLEAN;
function ">" (LEFT: in INTEGER; RIGHT: in FLOAT) return BOOLEAN;
function ">" (LEFT: in FLOAT; RIGHT: in INTEGER) return BOOLEAN;
function "<=" (LEFT: in INTEGER; RIGHT: in FLOAT) return BOOLEAN;
function "<=" (LEFT: in FLOAT; RIGHT: in INTEGER) return BOOLEAN;
function ">=" (LEFT: in INTEGER; RIGHT: in FLOAT) return BOOLEAN;
function ">=" (LEFT: in FLOAT; RIGHT: in INTEGER) return BOOLEAN;
function "<=" (LEFT: in INTEGER; RIGHT: in FLOAT) return BOOLEAN;
function "<=" (LEFT: in FLOAT; RIGHT: in INTEGER) return BOOLEAN;
pragma inline ("+", "-", "+", "/", ",", ",">", ","<", ","=");

generic
type FIXED is delta <>
package FIXED_CONVERSION is
  function "+" (LEFT: in FIXED; RIGHT: in FLOAT) return FLOAT;

M-21
function "=" (LEFT: in FLOAT; RIGHT: in FIXED) return FLOAT;
function "+" (LEFT: in FIXED; RIGHT: in FLOAT) return FLOAT;
function "+" (LEFT: in FLOAT; RIGHT: in FIXED) return FLOAT;
function "+" (LEFT: in FIXED; RIGHT: in FIXED) return FLOAT;
function "+" (LEFT: in FLOAT; RIGHT: in FLOAT) return FLOAT;
function "/" (LEFT: in FIXED; RIGHT: in FLOAT) return FLOAT;
function "/" (LEFT: in FLOAT; RIGHT: in FIXED) return FLOAT;
function "+" (LEFT: in FLOAT; RIGHT: in FIXED) return BOOLEAN;
function "+" (LEFT: in FIXED; RIGHT: in FLOAT) return BOOLEAN;
function "+" (LEFT: in FLOAT; RIGHT: in FIXED) return BOOLEAN;
function "+" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "+" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "+" (LEFT: in FLOAT; RIGHT: in FIXED) return BOOLEAN;
function "+" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "+" (LEFT: in INTEGER; RIGHT: in FIXED) return FIXED;
function "+" (LEFT: in FIXED; RIGHT: in INTEGER) return FIXED;
function "+" (LEFT: in INTEGER; RIGHT: in FIXED) return FIXED;
function "+" (LEFT: in FIXED; RIGHT: in INTEGER) return FIXED;
function "+" (LEFT: in INTEGER; RIGHT: in FIXED) return FIXED;
function "+" (LEFT: in INTEGER; RIGHT: in FIXED) return FIXED;
function "+" (LEFT: in FIXED; RIGHT: in INTEGER) return FIXED;
function "+" (LEFT: in FIXED; RIGHT: in FIXED) return FIXED;
function "+" (LEFT: in FIXED; RIGHT: in FIXED) return FIXED;
function "+" (LEFT: in FIXED; RIGHT: in FIXED) return FIXED;
function "+" (LEFT: in FIXED; RIGHT: in FIXED) return FIXED;
function "<" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "<" (LEFT: in FIXED; RIGHT: in FLOAT) return BOOLEAN;
function "<" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "<" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "<" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "<" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "<" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "<" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "<" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "<" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function ">" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function ">" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function ">" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function ">" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function ">" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function ">" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function ">" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function ">" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function ">" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function ">" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function ">=" (LEFT: in INTEGER; RIGHT: in FIXED) return BOOLEAN;
function ">=" (LEFT: in FIXED; RIGHT: in INTEGER) return BOOLEAN;
function ">=" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function ">=" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function ">=" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function ">=" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function ">=" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function ">=" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function ">=" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function ">=" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "=" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "=" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "=" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "=" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "=" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "=" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "=" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;
function "=" (LEFT: in FIXED; RIGHT: in FIXED) return BOOLEAN;

pragma inline ("="
,-" 
, 
,"/" 

end FIXED_CONVERSION;

end BasicDefs;

M-22
ADA TRANSLATION USING CCCC TRANSLATOR

-- MK2
WITH cms2_to_ada_predefined;
USE cms2_to_ada_predefined;
WITH UNCHECKED_CONVERSION;
WITH SYSTEM;
USE SYSTEM;
PACKAGE MK2 IS
  --SYSTEM
  PACKAGE memory use IS
    FKPI : CONSTANT := 3.1416;
    FKPI2 : CONSTANT := 2*FKPI;
    SUDVTIME_memory : FLOAT := 0.0;
    ICNX_memory : INTEGER := 1;
    SUDVOSKP_memory : FLOAT := 0.0;
    SUDVOSVP_memory : FLOAT := 0.0;
    SUDVRAD1_memory : FLOAT := 0.0;
    SUDVRAD2_memory : FLOAT := 0.0;
    FTCONDAT_memory : ARRAY (0..98, 0..0) OF cms2_word;
    FTCSS_memory : ARRAY (0..98, 0..4) OF cms2_word;
    FPRSS_memory : ARRAY (0..98, 0..5) OF cms2_word;
    SUDVOSLT_memory : FLOAT := 32.0*(FKPI2/360.0);
    SUDVOSLN_memory : FLOAT := (-120.0)*(FKPI2/360.0);
    SUDVRNG_memory : FLOAT;
    SUDVERQ_memory : FLOAT;
    SUDVLAT1_memory : FLOAT;
    SUDVLAT2_memory : FLOAT;
    SUDVLONG1_memory : FLOAT;
    SUDVLONG2_memory : FLOAT;
    VRA1D_memory : FLOAT;
    VRA2D_memory : FLOAT;
    VATAN_memory : FLOAT;
    SUDVTIME_memory : FLOAT;
    TGFLAT_memory : FLOAT;
    TGFLONG_memory : FLOAT;
    BRRLLHTH_memory : FLOAT;
    TEMPRAG_memory : FLOAT;
    COSTHE_memory : FLOAT;
    SYNTHET_memory : FLOAT;
    COSLAT1_memory : FLOAT;
    SINALAT1_memory : FLOAT;
    COSBGR_memory : FLOAT;
    SINBGR_memory : FLOAT;
    exit_index : INTEGER;
    END memory_use;
-- THIS CMS2 SYSTEM CONTAINS ONE SYS-DD (SYSD) AND
-- ONE SYS-PROC (SYSP)
USE memory_use;
PACKAGE SYSD IS
  --SYS-DD
  TYPE SUDVTIME_item_type IS
    RECORD
      OVER : FLOAT := 0.0;
      -- current system time in sec
    END RECORD;
    TYPE SUDVTIME_item_pointer IS ACCESS SUDVTIME_item_type;
    TYPE SUDVTIME_one_type IS ARRAY (0..0) OF cms2_word;
    TYPE SUDVTIME_one_pointer IS ACCESS SUDVTIME_one_type;
    FUNCTION SUDVTIME_item_address_access IS
      NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>SUDVTIME_item_pointer)
    END;

    SUDVTIME : SUDVTIME_item_pointer:=SUDVTIME_item_address_access(
      SUDVTIME_memory'ADDRESS);
    FUNCTION SUDVTIME_one_address_access IS
      NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>SUDVTIME_one_pointer);
    SUDVTIME_one : SUDVTIME_one_pointer:=SUDVTIME_one_address_access(
SUDVTME_memory ADDRESS) ;
TYPE ICNX_item_type IS
  RECORD
    -- integer
    OVER : INTEGER := 1;
  END RECORD;

TYPE ICNX_item_pointer IS ACCESS ICNX_item_type;
TYPE ICNX_one_type IS ARRAY (0..0) OF cm52_word;
TYPE ICNX_one_pointer IS ACCESS ICNX_one_type;
FUNCTION ICNX_item_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>ICNX_item_pointer);
ICNX_item_pointer := ICNX_item_address_access(ICNX_memory ADDRESS) ;

FUNCTION ICNX_one_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>ICNX_one_pointer);
ICNX_one := ICNX_one_address_access(ICNX_memory ADDRESS) ;

TYPE SUDVOSXP_item_type IS
  RECORD
    -- float
    OVER : FLOAT := 0.0;
    -- own ship x-position in yards
  END RECORD;

TYPE SUDVOSXP_item_pointer IS ACCESS SUDVOSXP_item_type;
TYPE SUDVOSXP_one_type IS ARRAY (0..0) OF cm52_word;
TYPE SUDVOSXP_one_pointer IS ACCESS SUDVOSXP_one_type;
FUNCTION SUDVOSXP_item_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>SUDVOSXP_item_pointer);
SUDVOSXP_item_pointer := SUDVOSXP_item_address_access(
  SUDVOSXP_memory ADDRESS) ;
FUNCTION SUDVOSXP_one_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>SUDVOSXP_one_pointer);
SUDVOSXP_one := SUDVOSXP_one_address_access(
  SUDVOSXP_memory ADDRESS) ;

TYPE SUDVOSYP_item_type IS
  RECORD
    -- float
    OVER : FLOAT := 0.0;
    -- own ship y-position in yards
  END RECORD;

TYPE SUDVOSYP_item_pointer IS ACCESS SUDVOSYP_item_type;
TYPE SUDVOSYP_one_type IS ARRAY (0..0) OF cm52_word;
TYPE SUDVOSYP_one_pointer IS ACCESS SUDVOSYP_one_type;
FUNCTION SUDVOSYP_item_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>SUDVOSYP_item_pointer);
SUDVOSYP_item_pointer := SUDVOSYP_item_address_access(
  SUDVOSYP_memory ADDRESS) ;
FUNCTION SUDVOSYP_one_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>SUDVOSYP_one_pointer);
SUDVOSYP_one := SUDVOSYP_one_address_access(
  SUDVOSYP_memory ADDRESS) ;

TYPE SUDVRA1_item_type IS
  RECORD
    -- float
    OVER : FLOAT := 0.0;
    -- x-position diff, in yards
  END RECORD;

TYPE SUDVRA1_item_pointer IS ACCESS SUDVRA1_item_type;
TYPE SUDVRA1_one_type IS ARRAY (0..0) OF cm52_word;
TYPE SUDVRA1_one_pointer IS ACCESS SUDVRA1_one_type;
FUNCTION SUDVRA1_item_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>SUDVRA1_item_pointer);
SUDVRA1_item_pointer := SUDVRA1_item_address_access(
  SUDVRA1_memory ADDRESS) ;
FUNCTION SUDVRA1_one_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>SUDVRA1_one_pointer);
SUDVRA1_one := SUDVRA1_one_address_access(
  SUDVRA1_memory ADDRESS) ;

TYPE SUDVRA2_item_type IS
RECORD
  OVER : FLOAT := 0.0;
  -- y-position diff, in yards
END RECORD;

TYPE SUDVRAD2_item_pointer IS ACCESS SUDVRAD2_item_type;
TYPE SUDVRAD2_one_type IS ARRAY (0.0) OF cms2_word;
TYPE SUDVRAD2_one_pointer IS ACCESS SUDVRAD2_one_type;
FUNCTION SUDVRAD2_item_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>SUDVRAD2_item_pointer);
SUDVRAD2 : SUDVRAD2_item_pointer:=SUDVRAD2_item_address_access(
  SUDVRAD2_memory'ADDRESS);
FUNCTION SUDVRAD2_one_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>SUDVRAD2_one_pointer);
SUDVRAD2_one : SUDVRAD2_one_pointer:=SUDVRAD2_one_address_access(
  SUDVRAD2_memory'ADDRESS);
TYPE FTCONDAT_item_type IS
RECORD
  FVEQRADG : fixed32s4;
END RECORD;

TYPE FTCONDAT_one_type IS ARRAY (0.98) OF cms2_word;
TYPE FTCONDAT_one_pointer IS ACCESS FTCONDAT_one_type;
TYPE FTCONDAT_words_type IS ARRAY (0.98, 0.0) OF cms2_word;
TYPE FTCONDAT_words_pointer IS ACCESS FTCONDAT_words_type;
TYPE FTCONDAT_type IS ARRAY (0.98) OF FTCONDAT_item_type;
TYPE FTCONDAT_item_pointer IS ACCESS FTCONDAT_type;
FUNCTION FTCONDAT_one_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>FTCONDAT_one_pointer);
FTCONDAT_one : FTCONDAT_one_pointer:=FTCONDAT_one_address_access(
  FTCONDAT_memory'ADDRESS);
FUNCTION FTCONDAT_words_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>FTCONDAT_words_pointer);
FTCONDAT_words : FTCONDAT_words_pointer:=FTCONDAT_words_address_access(
  FTCONDAT_one.ALL'ADDRESS);
FUNCTION FTCONDAT_item_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>FTCONDAT_item_pointer);
FTCONDAT : FTCONDAT_item_pointer;
TYPE FTCSS_item_type IS
RECORD
  FVTIME : FLOAT;
  -- solution update time
  FVTXP : FLOAT;
  -- X position in yards
  FVTYP : FLOAT;
  -- Y position in yards
  FVTXV : FLOAT;
  -- X velocity in yards/sec
  FVTYV : FLOAT;
  -- Y velocity in yards/sec
END RECORD;

TYPE FTCSS_one_type IS ARRAY (0.494) OF cms2_word;
TYPE FTCSS_one_pointer IS ACCESS FTCSS_one_type;
TYPE FTCSS_words_type IS ARRAY (0.98, 0.4) OF cms2_word;
TYPE FTCSS_words_pointer IS ACCESS FTCSS_words_type;
TYPE FTCSS_type IS ARRAY (0.98) OF FTCSS_item_type;
TYPE FTCSS_item_pointer IS ACCESS FTCSS_type;
FUNCTION FTCSS_one_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>FTCSS_one_pointer);
FTCSS_one : FTCSS_one_pointer:=FTCSS_one_address_access(FTCSS_memory'ADDRESS);
FUNCTION FTCSS_words_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>FTCSS_words_pointer);
FTCSS_words : FTCSS_words_pointer:=FTCSS_words_address_access(FTCSS_one.
  ALL'ADDRESS);
FUNCTION FTCSS_item_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>FTCSS_item_pointer);
FTCSS : FTCSS_item_pointer;
TYPE FTPRSS_item_type IS
RECORD
FVTXPD : FLOAT ;
-- PFed target X position in yards
FVTYPD : FLOAT ;
-- PFed target Y position in yards
FVRNGD : FLOAT ;
-- PFed target range in yards
FVB RDG : FLOAT ;
-- PFed target bearing in radians
FVTGLATD : FLOAT ;
-- PFed target latitude
FVTGLOND : FLOAT ;
-- PFed target longitude
END RECORD;

TYPE FTPKSS_one_type IS ARRAY (0..593) OF cms2_wordType FTPKSS_one_pointer IS ACCESS FTPKSS_one_type;
TYPE FTPKSS_words_type IS ARRAY (0..98 , 0..5) OF cms2_wordType FTPKSS_words_pointer IS ACCESS FTPKSS_words_type;
TYPE FTPKSS_type IS ARRAY (0..98) OF FTPKSS_item_type;
TYPE FTPKSS_item_pointer IS ACCESS FTPKSS_type;
FUNCTION FTPKSS_one_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>FTPKSS_one_pointer);
FTPKSS_one : FTPKSS_one_pointer:=FTPKSS_one_address_access(FTPKSS_memory'ADDRESS);
FUNCTION FTPKSS_words_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>FTPKSS_words_pointer);
FTPKSS_words : FTPKSS_words_pointer:=FTPKSS_words_address_access(
FTPKSS_one.ALL'ADDRESS);
FUNCTION FTPKSS_item_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>FTPKSS_item_pointer);
FTPKSS : FTPKSS_item_pointer;
TYPE SUDVOSL_item_type IS
RECORD
  OVER : FLOAT := 32.0*(FKPI2/360.0);
  -- own ship latitude
END RECORD;

TYPE SUDVOSL_item_pointer IS ACCESS SUDVOSL_item_type;
TYPE SUDVOSL_one_type IS ARRAY (0..0) OF cms2_word;
TYPE SUDVOSL_one_pointer IS ACCESS SUDVOSL_one_type;
FUNCTION SUDVOSL_item_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SUDVOSL_item_pointer);
SUDVOSL : SUDVOSL_item_pointer:=SUDVOSL_item_address_access(SUDVOSL_memory'ADDRESS);
FUNCTION SUDVOSL_one_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SUDVOSL_one_pointer);
SUDVOSL_one : SUDVOSL_one_pointer:=SUDVOSL_one_address_access(SUDVOSL_memory'ADDRESS);
TYPE SUDVOSLN_item_type IS
RECORD
  OVER : FLOAT := (-120.0)*(FKPI2/360.0);
  -- own ship longitude
END RECORD;

TYPE SUDVOSLN_item_pointer IS ACCESS SUDVOSLN_item_type;
TYPE SUDVOSLN_one_type IS ARRAY (0..0) OF cms2_word;
TYPE SUDVOSLN_one_pointer IS ACCESS SUDVOSLN_one_type;
FUNCTION SUDVOSLN_item_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SUDVOSLN_item_pointer);
SUDVOSLN : SUDVOSLN_item_pointer:=SUDVOSLN_item_address_access(SUDVOSLN_memory'ADDRESS);
FUNCTION SUDVOSLN_one_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SUDVOSLN_one_pointer);
SUDVOSLN_one : SUDVOSLN_one_pointer:=SUDVOSLN_one_address_access(SUDVOSLN_memory'ADDRESS);

M-26
TYPE SUDVRNG_item_pointer IS ACCESS SUDVRNG_item_type;
TYPE SUDVRNG_one_type IS ARRAY (0..0) OF cms2_word;
TYPE SUDVRNG_one_pointer IS ACCESS SUDVRNG_one_type;
FUNCTION SUDVRNG_item_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SUDVRNG_item_pointer);

SUDVRNG : SUDVRNG_item_pointer:=SUDVRNG_item_address_access(
  SUDVRNG_memory'ADDRESS);
FUNCTION SUDVRNG_one_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SUDVRNG_one_pointer);
SUDVRNG_one : SUDVRNG_one_pointer:=SUDVRNG_one_address_access(
  SUDVRNG_memory'ADDRESS);

TYPE SUDVRBG_item_type IS RECORD
  OVER : FLOAT;
-- (parameter) bearing
END RECORD;

TYPE SUDVRBG_item_pointer IS ACCESS SUDVRBG_item_type;
TYPE SUDVRBG_one_type IS ARRAY (0..0) OF cms2_word;
TYPE SUDVRBG_one_pointer IS ACCESS SUDVRBG_one_type;
FUNCTION SUDVRBG_item_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SUDVRBG_item_pointer);

SUDVRBG : SUDVRBG_item_pointer:=SUDVRBG_item_address_access(
  SUDVRBG_memory'ADDRESS);
FUNCTION SUDVRBG_one_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SUDVRBG_one_pointer);
SUDVRBG_one : SUDVRBG_one_pointer:=SUDVRBG_one_address_access(
  SUDVRBG_memory'ADDRESS);

TYPE SUDVLAT1_item_type IS RECORD
  OVER : FLOAT;
-- (parameter) input latitude
END RECORD;

TYPE SUDVLAT1_item_pointer IS ACCESS SUDVLAT1_item_type;
TYPE SUDVLAT1_one_type IS ARRAY (0..0) OF cms2_word;
TYPE SUDVLAT1_one_pointer IS ACCESS SUDVLAT1_one_type;
FUNCTION SUDVLAT1_item_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SUDVLAT1_item_pointer);

SUDVLAT1 : SUDVLAT1_item_pointer:=SUDVLAT1_item_address_access(
  SUDVLAT1_memory'ADDRESS);
FUNCTION SUDVLAT1_one_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SUDVLAT1_one_pointer);
SUDVLAT1_one : SUDVLAT1_one_pointer:=SUDVLAT1_one_address_access(
  SUDVLAT1_memory'ADDRESS);

TYPE SUDVLAT2_item_type IS RECORD
  OVER : FLOAT;
-- (parameter) output latitude
END RECORD;

TYPE SUDVLAT2_item_pointer IS ACCESS SUDVLAT2_item_type;
TYPE SUDVLAT2_one_type IS ARRAY (0..0) OF cms2_word;
TYPE SUDVLAT2_one_pointer IS ACCESS SUDVLAT2_one_type;
FUNCTION SUDVLAT2_item_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SUDVLAT2_item_pointer);

SUDVLAT2 : SUDVLAT2_item_pointer:=SUDVLAT2_item_address_access(
  SUDVLAT2_memory'ADDRESS);
FUNCTION SUDVLAT2_one_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SUDVLAT2_one_pointer);
SUDVLAT2_one : SUDVLAT2_one_pointer:=SUDVLAT2_one_address_access(
  SUDVLAT2_memory'ADDRESS);

TYPE SUDVLON1_item_type IS RECORD
  OVER : FLOAT;
-- (parameter) input longitude
END RECORD;

TYPE SUDVLON1_item_pointer IS ACCESS SUDVLON1_item_type;
TYPE SUDVLON1_one_type IS ARRAY (0..0) OF cms2_word;
TYPE SUDVLO1_one_pointer IS ACCESS SUDVLO1_one_type;
FUNCTION SUDVLO1_item address access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>SUDVLO1_item_pointer);
SUDVLO1: SUDVLO1_item_pointer:=SUDVLO1_item_address_access(
    SUDVLO1_memory'ADDRESS);
FUNCTION SUDVLO1_one_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>SUDVLO1_one_pointer);
SUDVLO1_one: SUDVLO1_one_pointer:=SUDVLO1_one_address_access(
    SUDVLO1_memory'ADDRESS);
TYPE SUDVLO2_item_type IS
RECORD
    OVER : FLOAT;
-- (parameter) output longitude
END RECORD;

TYPE SUDVLO2_item_pointer IS ACCESS SUDVLO2_item_type;
TYPE SUDVLO2_one_type IS ARRAY (0..0) OF cms2_word;
TYPE SUDVLO2_one_pointer IS ACCESS SUDVLO2_one_type;
FUNCTION SUDVLO2_item_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>SUDVLO2_item_pointer);
SUDVLO2: SUDVLO2_item_pointer:=SUDVLO2_item_address_access(
    SUDVLO2_memory'ADDRESS);
FUNCTION SUDVLO2_one_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>SUDVLO2_one_pointer);
SUDVLO2_one: SUDVLO2_one_pointer:=SUDVLO2_one_address_access(
    SUDVLO2_memory'ADDRESS);
TYPE VRAD1_item_type IS
RECORD
    OVER : FLOAT;
-- (parameter) two ATAN arguments
END RECORD;

TYPE VRAD1_item_pointer IS ACCESS VRAD1_item_type;
TYPE VRAD1_one_type IS ARRAY (0..0) OF cms2_word;
TYPE VRAD1_one_pointer IS ACCESS VRAD1_one_type;
FUNCTION VRAD1_item_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>VRAD1_item_pointer);
VRAD1: VRAD1_item_pointer:=VRAD1_item_address_access(VRAD1_memory'ADDRESS);
FUNCTION VRAD1_one_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>VRAD1_one_pointer);
VRAD1_one: VRAD1_one_pointer:=VRAD1_one_address_access(VRAD1_memory'ADDRESS);
TYPE VRAD2_item_type IS
RECORD
    OVER : FLOAT;
-- (parameter) two ATAN arguments
END RECORD;

TYPE VRAD2_item_pointer IS ACCESS VRAD2_item_type;
TYPE VRAD2_one_type IS ARRAY (0..0) OF cms2_word;
TYPE VRAD2_one_pointer IS ACCESS VRAD2_one_type;
FUNCTION VRAD2_item_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>VRAD2_item_pointer);
VRAD2: VRAD2_item_pointer:=VRAD2_item_address_access(VRAD2_memory'ADDRESS);
FUNCTION VRAD2_one_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>VRAD2_one_pointer);
VRAD2_one: VRAD2_one_pointer:=VRAD2_one_address_access(VRAD2_memory'ADDRESS);
END VRAD1;
USE memory_use;
PACKAGE SYSF IS
--SYS-PROC
FUNCTION SUDPATAN ( SUDPATAN_VRAD1 : IN FLOAT ; SUDPATAN_VRAD2 : IN FLOAT )
    RETURN INTEGER;
PROCEDURE SUDPATAN ( SUDPATAN_VRAD1 : IN FLOAT ; SUDPATAN_VRAD2 : IN FLOAT )
BEGIN
    SUDPATAN ( SUDPATAN_VRAD1, SUDPATAN_VRAD2 );
END SYSF;

M-28
USE memory_use;
USE SYSD;
USE SYSP;
PACKAGE extdef IS
  PROCEDURE SUDPKEFCS RENAMES SYSP.SUDPKEFCS;
  PROCEDURE SUDPRBLL (SUDPRBLL_SUDVRNG : IN FLOAT; SUDPRBLL_SUDVBRG : IN
    FLOAT; SUDPRBLL_SUDVLAT1 : IN FLOAT; SUDPRBLL_SUDVLO1 : IN FLOAT;
    SUDPRBLL_SUDVLAT2 : OUT FLOAT; SUDPRBLL_SUDVLO2 : OUT FLOAT) RENAMES
    SYSP.SUDPRL;
END extdef;
END MK2;
WITH cms2_to_ada_predefined;
USE cms2_to_ada_predefined;
WITH UNCHECKED_CONVERSION;
WITH SYSTEM;
USE SYSTEM;
WITH math_lib_cms2;
USE math_lib_cms2;
WITH MK2;
USE MK2;
PACKAGE BODY MK2 IS
USE memory_use;
USE SYSD;
USE SYSR;
PACKAGE BODY SYSD IS
PROCEDURE FTCONDAT_item_address_access_init IS
   p : FTCONDAT_item_pointer := FTCONDAT_item_address_access(FTCONDAT_one.
   ALL'ADDRESS);
BEGIN
   p.ALL(0).FVEQRADG := 6975563.33;
   FTCONDAT := p;
END FTCONDAT_item_address_access_init;
PROCEDURE FTCSS_item_address_access_init IS
   p : FTCSS_item_pointer := FTCSS_item_address_access(FTCSS_one.ALL'ADDRESS)
   );
BEGIN
   p.ALL(0).FVTIME := 0.0;
   p.ALL(0).FVTXP := 0.0;
   p.ALL(0).FVTYP := 0.0;
   p.ALL(0).FVTXV := 0.0;
   p.ALL(0).FVTYY := 0.0;
   FTCSS := p;
END FTCSS_item_address_access_init;
PROCEDURE FTPKSS_item_address_access_init IS
   p : FTPKSS_item_pointer := FTPKSS_item_address_access(FTPKSS_one.ALL'
   ADDRESS);
BEGIN
   p.ALL(0).FVTXP := 0.0;
   p.ALL(0).FVTYP := 0.0;
   p.ALL(0).FVTXV := 0.0;
   p.ALL(0).FVTYY := 0.0;
   p.ALL(0).FVTGRLAT := 0.0;
   p.ALL(0).FVTCLAT := 0.0;
   FTPKSS := p;
END FTPKSS_item_address_access_init;
END SYSD;
USE memory_use;
USE SYSD;
USE SYSP;

PACKAGE BODY SYSP IS
FUNCTION SUDPATAN ( SUDPATAN_VRAD1 : IN FLOAT; SUDPATAN_VRAD2 : IN FLOAT ) RETURN INTEGER IS

TYPE Vatan_item_type IS
   RECORD
      OVER : FLOAT;
   END RECORD;

   TYPE Vatan_item_pointer IS ACCESS Vatan_item_type;
   TYPE Vatan_one_type IS ARRAY (0..0) OF cms2_word;
   TYPE Vatan_one_pointer IS ACCESS Vatan_one_type;
   FUNCTION Vatan_item_address_access IS
      NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>Vatan_item_pointer);
   END FUNCTION;

   Vatan : Vatan_item_pointer := Vatan_item_address_access(Vatan_memory'[ADDRESS]);
   FUNCTION Vatan_one_address_access IS
      NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS, TARGET=>Vatan_one_pointer);
   END FUNCTION;

   Vatan_one : Vatan_one_pointer := Vatan_one_address_access(Vatan_memory'[ADDRESS]);
BEGIN
   VRAD1.ALL.OVER := SUDPATAN_VRAD1;
   VRAD2.ALL.OVER := SUDPATAN_VRAD2;
   IF VRAD1.ALL.OVER<0.00001 AND VRAD2.ALL.OVER<0.00001 THEN
      Vatan.ALL.OVER := 0.0;
   ELSE
      Vatan.ALL.OVER := ATAN2(VRAD1.ALL.OVER, VRAD2.ALL.OVER);
   END IF;
   RETURN INTEGER(Vatan.ALL.OVER);
END SUDPATAN;

PROCEDURE SUDPXKCS IS
   -- ----------------- -----------------------------------------------
   -- 
   --                   Segment: FCS
   --                   CSCI Name: TMAB
   --                   TLOCS: SUD
   --                   LLOCS: SUDLTD
   --                   UNIT: SUDPXKCS
   --                   Part Number: PEG528777
   --                   Classification: UNCLASSIFIED
   --                   Company_ID: Raytheon, CAGE Code 49956
   -- 
   -- ----------------- -----------------------------------------------
   --
   -- Library Name: MK2ECP6:[SRC.FC.TMAB.SUD.SRC]
   -- Element Name: SUDPXKCS.SRC
   -- Revision Number: 1
   -- Revision Date, Time: 25-NOV-1992 10:57
   -- Current Date, Time: 3-MAR-1995 16:44
   -- 
   -- ----------------- -----------------------------------------------
   -- Author: Mark Damiani
   -- 
   -- Overview: This purpose of this procedure is to perform
   -- the following for all FCS tactical/training targets not including OTH targets:
   -- 1) Compute FKed Target X Position.
   -- 2) Compute FKed Target Y Position.
   -- 3) Compute FKed Target Range
   -- 4) Compute FKed Target Bearing
   -- 5) Compute FKed Target Latitude and Longitude
   -- by calling the SUDPXKSS system common routine.
   -- 
   -- Effects:
   -- 
   -- Requirements Trace:

M-31
Algorithm:

Notes: This procedure will be called during a SUD Time
       Dependent entrance.

Exceptions Raised:

------------------------------------------------------------------------
TYPE SUDVTME_item_type IS
   RECORD
   OVER : FLOAT ;
   --Target Solution FK Delta Time
END RECORD;

TYPE SUDVTME_item_pointer IS ACCESS SUDVTME_item_type ;
TYPE SUDVTME_one_type IS ARRAY (0..0) OF cms2_word ;
TYPE SUDVTME_one_pointer IS ACCESS SUDVTME_one_type ;
FUNCTION SUDVTME_item_address_access IS
   NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SUDVTME_item_pointer);
SUDVTME : SUDVTME_item_pointer := SUDVTME_item_address_access(
   SUDVTME_memory'ADDRESS) ;
FUNCTION SUDVTME_one_address_access IS
   NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SUDVTME_one_pointer) ;
SUDVTME_one : SUDVTME_one_pointer := SUDVTME_one_address_access(
   SUDVTME_memory'ADDRESS) ;

TYPE TGTLAT_item_type IS
   RECORD
   OVER : FLOAT ;
   --FRed Target Latitude
END RECORD;

TYPE TGTLAT_item_pointer IS ACCESS TGTLAT_item_type ;
TYPE TGTLAT_one_type IS ARRAY (0..0) OF cms2_word ;
TYPE TGTLAT_one_pointer IS ACCESS TGTLAT_one_type ;
FUNCTION TGTLAT_item_address_access IS
   NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>TGTLAT_item_pointer) ;
TGTLAT : TGTLAT_item_pointer := TGTLAT_item_address_access(TGTLAT_memory'ADDRESS) ;
FUNCTION TGTLAT_one_address_access IS
   NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>TGTLAT_one_pointer) ;
TGTLAT_one : TGTLAT_one_pointer := TGTLAT_one_address_access(
   TGTLAT_memory'ADDRESS) ;

TYPE TGTLONG_item_type IS
   RECORD
   OVER : FLOAT ;
   --FRed Target Longitude
END RECORD;

TYPE TGTLONG_item_pointer IS ACCESS TGTLONG_item_type ;
TYPE TGTLONG_one_type IS ARRAY (0..0) OF cms2_word ;
TYPE TGTLONG_one_pointer IS ACCESS TGTLONG_one_type ;
FUNCTION TGTLONG_item_address_access IS
   NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>TGTLONG_item_pointer) ;
TGTLONG : TGTLONG_item_pointer := TGTLONG_item_address_access(
   TGTLONG_memory'ADDRESS) ;
FUNCTION TGTLONG_one_address_access IS
   NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>TGTLONG_one_pointer) ;
TGTLONG_one : TGTLONG_one_pointer := TGTLONG_one_address_access(
   TGTLONG_memory'ADDRESS) ;

BEGIN

SUDVTME.ALL.OVER := SUDVTME.ALL.OVER-FTCSS.ALL(INCX.ALL.OVER).

-- Compute FCS Position Kept Target X and Y Positions
-- Set Target Solution Delta Time to current System Time
-- minus System Solution table Solution Update Time for
-- current IGN.

BEGIN

SUDVTME.ALL.OVER := SUDVTME.ALL.OVER-FTCSS.ALL(INCX.ALL.OVER).

-- Compute FCS PK Target X Position.

M-32
FTFKS.ALL(ICNX.ALL.OVER), FXTP := FTFKSS.ALL(ICNX.ALL.OVER), FXTP+(FTFKSS.ALL(ICNX.ALL.OVER), FXTPV*SUVDVTME.ALL.OVER);
-- Compute FCS PK Target Y Position.
FTFKS.ALL(ICNX.ALL.OVER), FYTP := FTFKSS.ALL(ICNX.ALL.OVER), FYTP+(FTFKSS.ALL(ICNX.ALL.OVER), FYTPV*SUVDVTME.ALL.OVER);
--
-- Compute FCS Position Kept Target Range.
--
-- FTFKSS.ALL(ICNX.ALL.OVER), FYVRG := SQRT((FTFKSS.ALL(ICNX.ALL.OVER), FXTP-SUVDVTME.ALL.OVER)**2+(FTFKSS.ALL(ICNX.ALL.OVER), FYTP-SUVDVTME.ALL.OVER)**2);
--
-- Compute FCS Position Kept Target Bearing.
--
Clip target range to MAX
END IF;
SUDVRAD1.ALL.OVER := FTFKSS.ALL(ICNX.ALL.OVER), FXTP-SUVDVTME.ALL.OVER;
SUDVRAD2.ALL.OVER := FTFKSS.ALL(ICNX.ALL.OVER), FYTP-SUVDVTME.ALL.OVER;
FTFKSS.ALL(ICNX.ALL.OVER), FYVRG := FLOAT(SUDPTAN(SUDVRAD1.ALL.OVER, SUDVRAD2.ALL.OVER));
--
-- PKed Target Latitude and PKed Target Longitude shall be computed using the Range, Azimuth to Latitude, Longitude
-- (SUDPBLR) common conversion function.
-- Input parameters shall include current Own Ship Latitude and Own Ship Longitude, PKed Target Range, and PKed Target Bearing.
-- Output parameters shall be PKed Target Latitude and PKed Target Longitude.
--
SUDVRAD1.ALL.OVER := FTFKSS.ALL(ICNX.ALL.OVER), FYVRG;
SUDVRAD2.ALL.OVER := FTFKSS.ALL(ICNX.ALL.OVER), FYVRG;
SUDPBLR(FTFKSS.ALL(ICNX.ALL.OVER), SUVDVTME.ALL.OVER, SUDVRAD1.ALL.OVER, SUDVRAD2.ALL.OVER, SUGOSLXM.ALL.OVER, SUGOSLTR.ALL.OVER, SGTLAT.ALL.OVER, SGTLONG.ALL.OVER);
-- Save PKed Target Latitude in PK System Solution table.
FTFKSS.ALL(ICNX.ALL.OVER), FYTGLAT := SGTLAT.ALL.OVER;
-- Save PKed Target Longitude in PK System Solution table.
FTFKSS.ALL(ICNX.ALL.OVER), FYTGLON := SGTLONG.ALL.OVER;
END SUDPBLR;
PROCEDURE SUDPBLR ( SUDPBLR_SUDVRADG : IN FLOAT; SUDPBLR_SUDVRBG : IN FLOAT; SUDPBLR_SUDVRAD1 : IN FLOAT; SUDPBLR_SUDVRAD11 : IN FLOAT; SUDPBLR_SUDVRAD12 : OUT FLOAT; SUDPBLR_SUDVRAD13 : OUT FLOAT ) IS
--
-- Segment: FCS
-- CSCI Name: TMAB
-- TLMSC: SUD
-- LLMSC: SUD
-- UNIT: SUDPBLR
-- Part Number: FRG528777
-- Classification: UNCLASSIFIED
-- Company_ID: Raytheon, CAGE Code 49956
--
-- Library Name: MK2EC6:[SRC.FC.TMAB.SUD.SRC]
-- Element Name: SUDPBLR.SRC
-- Revision Number: 2
-- Revision Date, Time: 27-APR-1993 16:28
-- Current Date, Time: 3-MAR-1995 16:44
--
-- Author: Jim Pryor (JRP), Bill Croesdale (WKC)
-- Overview:
-- The Range/Bearing to Lat/Long unit will
calculate the latitude and longitude coordinates of a
position represented by a range,bearing from the input

M-33
latitude/longitude position.

Effects:

Requirements Trace: PROCESS_NAV

Algorithm:

\[ \theta = \frac{R}{R_E} \]

\[ \text{Target Latitude} = \text{Arcsin} \left( \frac{\sin(P0) \cdot \cos(\theta) + \cos(P0) \cdot \sin(\theta) \cdot \cos(By)}{} \right) \]

\[ \text{Target Longitude} = \arctan2(\sin(\theta) \cdot \sin(By), \cos(P0) \cdot \cos(\theta) - \sin(P0) \cdot \sin(\theta) \cdot \cos(By)) + U0 \]

\[ R = \text{Range to target from input Lat/Lon(yds)} \]

\[ By = \text{Bearing to target from input Lat/Lon} \]

\[ P0 = \text{input Latitude} \]

\[ U0 = \text{input Longitude} \]

\[ R_E = \text{Radius of the earth(from FTCONDAT)} \]

Notes:

All angles(input/output) in floating point Radians,
and all ranges in floating point yards.

Exceptions Raised:

---

---

=====================================================================

TYPE RBLLTHET_item_type IS
  RECORD
    OVER : FLOAT;
    --interim value (R/R_E)
END RECORD;

TYPE RBLLTHET_item_pointer IS ACCESS RBLLTHET_item_type;
TYPE RBLLTHET_one_type IS ARRAY (0..0) OF cms2_word;
TYPE RBLLTHET_one_pointer IS ACCESS RBLLTHET_one_type;

FUNCTION RBLLTHET_item_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>RBLLTHET_item_pointer);

RBLLTHET : RBLLTHET_item_pointer := RBLLTHET_item_address_access( RBLLTHET_memory'ADDRESS);

FUNCTION RBLLTHET_one_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>RBLLTHET_one_pointer);

RBLLTHET_one : RBLLTHET_one_pointer := RBLLTHET_one_address_access( RBLLTHET_memory'ADDRESS);

TYPE TEMPARG_item_type IS
  RECORD
    OVER : FLOAT;
    --interim value for arcsin
END RECORD;

TYPE TEMPARG_item_pointer IS ACCESS TEMPARG_item_type;
TYPE TEMPARG_one_type IS ARRAY (0..0) OF cms2_word;
TYPE TEMPARG_one_pointer IS ACCESS TEMPARG_one_type;

FUNCTION TEMPARG_item_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>TEMPARG_item_pointer);

TEMPARG : TEMPARG_item_pointer := TEMPARG_item_address_access( TEMPARG_memory'ADDRESS);

FUNCTION TEMPARG_one_address_access IS
  NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>TEMPARG_one_pointer);

TEMPARG_one : TEMPARG_one_pointer := TEMPARG_one_address_access( TEMPARG_memory'ADDRESS);

TYPE COSTHTET_item_type IS
  RECORD
    OVER : FLOAT;
END RECORD;

TYPE COSTHTET_item_pointer IS ACCESS COSTHTET_item_type;
TYPE COSTHTET_one_type IS ARRAY (0..0) OF cms2_word;

M-34
TYPE COSTHET_one_pointer IS ACCESS COSTHET_one_type;
FUNCTION COSTHET_item_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>COSTHET_item_pointer);
;
COSTHET : COSTHET_item_pointer:=COSTHET_item_address_access(
COSTHET_memory'ADDRESS);
FUNCTION COSTHET_one_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>COSTHET_one_pointer);
;
COSTHET_one : COSTHET_one_pointer:=COSTHET_one_address_access(
COSTHET_memory'ADDRESS);
TYPE SINTHET_item_type IS
RECORD
OVER : FLOAT;
END RECORD;

TYPE SINTHET_item_pointer IS ACCESS SINTHET_item_type;
TYPE SINTHET_one_type IS ARRAY (0..0) OF cms2_word;
TYPE SINTHET_one_pointer IS ACCESS SINTHET_one_type;
FUNCTION SINTHET_item_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SINTHET_item_pointer);
;
SINTHET : SINTHET_item_pointer:=SINTHET_item_address_access(
SINTHET_memory'ADDRESS);
FUNCTION SINTHET_one_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SINTHET_one_pointer);
;
SINTHET_one : SINTHET_one_pointer:=SINTHET_one_address_access(
SINTHET_memory'ADDRESS);
TYPE COSLAT1_item_type IS
RECORD
OVER : FLOAT;
END RECORD;

TYPE COSLAT1_item_pointer IS ACCESS COSLAT1_item_type;
TYPE COSLAT1_one_type IS ARRAY (0..0) OF cms2_word;
TYPE COSLAT1_one_pointer IS ACCESS COSLAT1_one_type;
FUNCTION COSLAT1_item_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>COSLAT1_item_pointer);
;
COSLAT1 : COSLAT1_item_pointer:=COSLAT1_item_address_access(
COSLAT1_memory'ADDRESS);
FUNCTION COSLAT1_one_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>COSLAT1_one_pointer);
;
COSLAT1_one : COSLAT1_one_pointer:=COSLAT1_one_address_access(
COSLAT1_memory'ADDRESS);
TYPE SINLAT1_item_type IS
RECORD
OVER : FLOAT;
END RECORD;

TYPE SINLAT1_item_pointer IS ACCESS SINLAT1_item_type;
TYPE SINLAT1_one_type IS ARRAY (0..0) OF cms2_word;
TYPE SINLAT1_one_pointer IS ACCESS SINLAT1_one_type;
FUNCTION SINLAT1_item_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SINLAT1_item_pointer);
;
SINLAT1 : SINLAT1_item_pointer:=SINLAT1_item_address_access(
SINLAT1_memory'ADDRESS);
FUNCTION SINLAT1_one_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SINLAT1_one_pointer);
;
SINLAT1_one : SINLAT1_one_pointer:=SINLAT1_one_address_access(
SINLAT1_memory'ADDRESS);
TYPE COSBRG_item_type IS
RECORD
OVER : FLOAT;
END RECORD;

TYPE COSBRG_item_pointer IS ACCESS COSBRG_item_type;
TYPE COSBRG_one_type IS ARRAY (0..0) OF cms2_word;
TYPE COSBRG_one_pointer IS ACCESS COSBRG_one_type;
FUNCTION COSBRG_item_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>COSBRG_item_pointer);
;
M-35
COSBRG: COSBRG_item_pointer:=COSBRG_item_address_access(COSBRG_memory'ADDRESS);
FUNCTION COSBRG_one_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>COSBRG_one_pointer);

COSBRG_one: COSBRG_one_pointer:=COSBRG_one_address_access(COSBRG_memory'ADDRESS);
TYPE SINBRG_item_type IS
RECORD
  OVER : FLOAT;
END RECORD;

TYPE SINBRG_item_pointer IS ACCESS SINBRG_item_type;
TYPE SINBRG_one_type IS ARRAY (0..0) OF cms2_word;
TYPE SINBRG_one_pointer IS ACCESS SINBRG_one_type;
FUNCTION SINBRG_one_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SINBRG_item_pointer);

SINBRG: SINBRG_item_pointer:=SINBRG_item_address_access(SINBRG_memory'ADDRESS);
FUNCTION SINBRG_one_address_access IS
NEW UNCHECKED_CONVERSION(SOURCE=>ADDRESS,TARGET=>SINBRG_one_pointer);

SINBRG_one: SINBRG_one_pointer:=SINBRG_one_address_access(SINBRG_memory'ADDRESS);
BEGIN
  SUDPBRLL.SUDPVRG := SUDPBRLL_SUDPVRG;
  SUDPVRG.ALL.OVER := SUDPBRLL_SUDPVRG;
  SUDPVLAT1.ALL.OVER := SUDPBRLL_SUDPVLAT1;
  SUDPVLON11.ALL.OVER := SUDPBRLL_SUDPVLON1;
  RELLTHET.ALL.OVER := RELLTHET.ALL.OVER/FLOAT(FTOCONDAI.ALL.OVER,FVEQRADG)
  COSTHET.ALL.OVER := COS(RELLTHET.ALL.OVER);
  SINHET.ALL.OVER := SIN(RELLTHET.ALL.OVER);
  COSLAT1.ALL.OVER := COS(SUDPVLAT1.ALL.OVER);
  SINLAT1.ALL.OVER := SIN(SUDPVLAT1.ALL.OVER);
  COSBRG.ALL.OVER := COS(SUDPVRG.ALL.OVER);
  SINBRG.ALL.OVER := SIN(SUDPVRG.ALL.OVER);
  TEMPARG.ALL.OVER := SINLT1.ALL.OVER+COSTHET.ALL.OVER+COSLAT1.ALL.
  OVER*SINHET.ALL.OVER*COSBRG.ALL.OVER;
  SUDPVLAT2.ALL.OVER := ASIN(TEMPARG.ALL.OVER);
  SUDPVLON2.ALL.OVER := SUDPATAN(SINHET.ALL.OVER*SINBRG.ALL.OVER,
  COSLAT1.ALL.OVER*COSTHET.ALL.OVER-SINLAT1.ALL.OVER*SINHET.ALL.OVER
  +COSBRG.ALL.OVER)*SUDPVLON11.ALL.OVER;
  IF SUDPVLON2.ALL.OVER>FKPI THEN
    SUDPVLON2.ALL.OVER := SUDPVLON2.ALL.OVER-FKPI2;
  END IF;
  SUDPBRLL_SUDPVLAT2 := SUDPVLAT2.ALL.OVER;
  SUDPBRLL_SUDPVLON2 := SUDPVLON2.ALL.OVER;
END CUDPBRLL;
END SYSP; END M02;
with System;
use System;

with Unchecked_Conversion;

package Cms2_To_Ada_Predefined is

Word : constant := 4; -- storage unit is byte, 4 bytes per word

subtype Unsigned1 is Unsigned_Longword range 0 .. 2**1 - 1; -- I 1++ U $
subtype Unsigned2 is Unsigned_Longword range 0 .. 2**2 - 1; -- I 2++ U $
subtype Unsigned3 is Unsigned_Longword range 0 .. 2**3 - 1; -- I 3++ U $
subtype Unsigned4 is Unsigned_Longword range 0 .. 2**4 - 1; -- I 4++ U $
subtype Unsigned5 is Unsigned_Longword range 0 .. 2**5 - 1; -- I 5++ U $
subtype Unsigned6 is Unsigned_Longword range 0 .. 2**6 - 1; -- I 6++ U $
subtype Unsigned7 is Unsigned_Longword range 0 .. 2**7 - 1; -- I 7++ U $
subtype Unsigned8 is Unsigned_Longword range 0 .. 2**8 - 1; -- I 8++ U $
subtype Unsigned9 is Unsigned_Longword range 0 .. 2**9 - 1; -- I 9++ U $
subtype Unsigned10 is Unsigned_Longword range 0 .. 2**10 - 1; -- I++ 10 U $
subtype Unsigned11 is Unsigned_Longword range 0 .. 2**11 - 1; -- I++ 11 U $
subtype Unsigned12 is Unsigned_Longword range 0 .. 2**12 - 1; -- I++ 12 U $
subtype Unsigned13 is Unsigned_Longword range 0 .. 2**13 - 1; -- I++ 13 U $
subtype Unsigned14 is Unsigned_Longword range 0 .. 2**14 - 1; -- I++ 14 U $
subtype Unsigned15 is Unsigned_Longword range 0 .. 2**15 - 1; -- I++ 15 U $
subtype Unsigned16 is Unsigned_Longword range 0 .. 2**16 - 1; -- I++ 16 U $
subtype Unsigned17 is Unsigned_Longword range 0 .. 2**17 - 1; -- I++ 17 U $
subtype Unsigned18 is Unsigned_Longword range 0 .. 2**18 - 1; -- I++ 18 U $
subtype Unsigned19 is Unsigned_Longword range 0 .. 2**19 - 1; -- I++ 19 U $
subtype Unsigned20 is Unsigned_Longword range 0 .. 2**20 - 1; -- I++ 20 U $
subtype Unsigned21 is Unsigned_Longword range 0 .. 2**21 - 1; -- I++ 21 U $
subtype Unsigned22 is Unsigned_Longword range 0 .. 2**22 - 1; -- I++ 22 U $
subtype Unsigned23 is Unsigned_Longword range 0 .. 2**23 - 1; -- I++ 23 U $
subtype Unsigned24 is Unsigned_Longword range 0 .. 2**24 - 1; -- I++ 24 U $
subtype Unsigned25 is Unsigned_Longword range 0 .. 2**25 - 1; -- I++ 25 U $
subtype Unsigned26 is Unsigned_Longword range 0 .. 2**26 - 1; -- I++ 26 U $
subtype Signed1 is Integer range -2**30 .. 2**31 - 1; -- I + S $
subtype Signed2 is Integer range -2**31 .. 2**31 - 1; -- I + S $
subtype Signed3 is Integer range -2**32 .. 2**32 - 1; -- I + S $
subtype Signed4 is Integer range -2**33 .. 2**33 - 1; -- I + S $
subtype Signed5 is Integer range -2**34 .. 2**34 - 1; -- I + S $
subtype Signed6 is Integer range -2**35 .. 2**35 - 1; -- I + S $
subtype Signed7 is Integer range -2**36 .. 2**36 - 1; -- I + S $
subtype Signed8 is Integer range -2**37 .. 2**37 - 1; -- I + S $
subtype Signed9 is Integer range -2**38 .. 2**38 - 1; -- I + S $
subtype Signed10 is Integer range -2**39 .. 2**39 - 1; -- I + S $
subtype Signed11 is Integer range -2**40 .. 2**40 - 1; -- I + S $
subtype Signed12 is Integer range -2**41 .. 2**41 - 1; -- I + S $
subtype Signed13 is Integer range -2**42 .. 2**42 - 1; -- I + S $
subtype Signed14 is Integer range -2**43 .. 2**43 - 1; -- I + S $
subtype Signed15 is Integer range -2**44 .. 2**44 - 1; -- I + S $
subtype Signed16 is Integer range -2**45 .. 2**45 - 1; -- I + S $
subtype Signed17 is Integer range -2**46 .. 2**46 - 1; -- I + S $

M-37
subtype Signed18 is Integer range -2**17 .. 2**17 - 1; -- I 18 S$
subtype Signed19 is Integer range -2**18 .. 2**18 - 1; -- I 19 S$
subtype Signed20 is Integer range -2**19 .. 2**19 - 1; -- I 20 S$
subtype Signed21 is Integer range -2**20 .. 2**20 - 1; -- I 21 S$
subtype Signed22 is Integer range -2**21 .. 2**21 - 1; -- I 22 S$
subtype Signed23 is Integer range -2**22 .. 2**22 - 1; -- I 23 S$
subtype Signed24 is Integer range -2**23 .. 2**23 - 1; -- I 24 S$
subtype Signed25 is Integer range -2**24 .. 2**24 - 1; -- I 25 S$
subtype Signed26 is Integer range -2**25 .. 2**25 - 1; -- I 26 S$
subtype Signed27 is Integer range -2**26 .. 2**26 - 1; -- I 27 S$
subtype Signed28 is Integer range -2**27 .. 2**27 - 1; -- I 28 S$
subtype Signed29 is Integer range -2**28 .. 2**28 - 1; -- I 29 S$
subtype Signed30 is Integer range -2**29 .. 2**29 - 1; -- I 30 S$
subtype Signed31 is Integer range -2**30 .. 2**30 - 1; -- I 31 S$
subtype Signed32 is Integer; -- range -2**31..2**31-1; -- I 32 S$
subtype Signed33 is Integer; -- range -2**32..2**32-1; -- I 33 S$
subtype Signed34 is Integer; -- range -2**36..2**36-1; -- I 37 S$
subtype Signed40 is Integer; -- range -2**39..2**39-1; -- I 40 S$
subtype Signed48 is Integer; -- range -2**47..2**47-1; -- I 48 S$
subtype Signed56 is Integer; -- range -2**55..2**55-1; -- I 56 S$
subtype Signed64 is Integer; -- range -2**63..2**63-1; -- I 64 S$

-- Fixed point types

type Fixed2s2 is delta 2.*(-2) range -2.*(-1) .. 2.*(-2);
type Fixed3s0 is delta 2.*(-0) range -2.*(-2) .. 2.*(-0);
type Fixed3s5 is delta 2.*(-5) range -2.*(-4) .. 2.*(-5);
type Fixed6s3 is delta 2.*(-3) range -2.*(-4) .. 2.*(-3);
type Fixed7s4 is delta 2.*(-4) range -2.*(-5) .. 2.*(-4);
type Fixed8s6 is delta 2.*(-0) range -2.*(-4) .. 2.*(-0);
type Fixed8s8 is delta 2.*(-8) range -2.*(-7) .. 2.*(-8);
type Fixed9s0 is delta 2.*(-0) range -2.*(-8) .. 2.*(-0);
type Fixed9s3 is delta 2.*(-3) range -2.*(-8) .. 2.*(-3);
type Fixed10s5 is delta 2.*(-5) range -2.*(-9) .. 2.*(-5);
type Fixed11s0 is delta 2.*(-0) range -2.*(-10) .. 2.*(-0);
type Fixed11s12 is delta 2.*(-12) range -2.*(-11) .. 2.*(-12);
type Fixed13s12 is delta 2.*(-12) range -2.*(-13) .. 2.*(-12);
type Fixed14s13 is delta 2.*(-13) range -2.*(-13) .. 2.*(-13);
type Fixed15s3 is delta 2.*(-3) range -2.*(-11) .. 2.*(-3);
type Fixed15s5 is delta 2.*(-5) range -2.*(-9) .. 2.*(-5);

type Fixed16s0 is delta 2.*(-0) range -2.*(-15) .. 2.*(-15);
type Fixed16s1 is delta 2.*(-1) range -2.*(-14) .. 2.*(-1);
type Fixed16s2 is delta 2.*(-2) range -2.*(-13) .. 2.*(-2);
type Fixed16s3 is delta 2.*(-3) range -2.*(-12) .. 2.*(-3);
type Fixed16s4 is delta 2.*(-4) range -2.*(-11) .. 2.*(-4);
type Fixed16s5 is delta 2.*(-5) range -2.*(-10) .. 2.*(-5);
type Fixed16s6 is delta 2.*(-6) range -2.*(-9) .. 2.*(-6);
type Fixed16s7 is delta 2.*(-7) range -2.*(-8) .. 2.*(-7);
type Fixed16s8 is delta 2.*(-8) range -2.*(-7) .. 2.*(-8);
type Fixed16s9 is delta 2.*(-9) range -2.*(-6) .. 2.*(-9);
type Fixed16s10 is delta 2.*(-10) range -2.*(-5) .. 2.*(-10);
type Fixed16s11 is delta 2.*(-11) range -2.*(-4) .. 2.*(-11);
type Fixed16s12 is delta 2.*(-12) range -2.*(-3) .. 2.*(-12);
type Fixed16s13 is delta 2.*(-13) range -2.*(-2) .. 2.*(-13);
type Fixed16s14 is delta 2.*(-14) range -2.*(-1) .. 2.*(-14);
type Fixed16s15 is delta 2.*(-15) range -2.*(-0) .. 2.*(-15);

type Fixed17s50 is delta 2.*(-50) range -2.*(-34) .. 2.*(-50);
type Fixed19s6 is delta 2.*(-6) range -2.*(-12) .. 2.*(-6);
type Fixed24s9 is delta 2.*(-9) range -2.*(-15) .. 2.*(-9);
type Fixed30s3 is delta 2.*(-3) range -2.*(-26) .. 2.*(-3);

type Fixed32s0 is delta 2.*(-0) range -2.*(-31) .. 2.*(-0);
type Fixed32s1 is delta 2.*(-1) range -2.*(-30) .. 2.*(-1);
type Fixed32s2 is delta 2.*(-2) range -2.*(-29) .. 2.*(-2);
type Fixed32s3 is delta 2.*(-3) range -2.*(-28) .. 2.*(-3);
type Fixed32s4 is delta 2.*(-4) range -2.*(-27) .. 2.*(-4);
type Fixed32s5 is delta 2.*(-5) range -2.*(-26) .. 2.*(-5);
type Fixed32s6 is delta 2.*(-6) range -2.*(-25) .. 2.*(-6);
type Fixed32s7 is delta 2.*(-7) range -2.*(-24) .. 2.*(-7);
type Fixed32s8 is delta 2.*(-8) range -2.*(-23) .. 2.*(-8);
type Fixed32s9 is delta 2.*(-9) range -2.*(-22) .. 2.*(-9);
type Fixed32s10 is delta 2.*(-10) range -2.*(-21) .. 2.*(-10);
type Fixed32s11 is delta 2.*(-11) range -2.*(-20) .. 2.*(-11);
<table>
<thead>
<tr>
<th>Type</th>
<th>Delta</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>type Fixed32s12</td>
<td>delta</td>
<td>2.0**(-12)</td>
</tr>
<tr>
<td>type Fixed32s13</td>
<td>delta</td>
<td>2.0**(-13)</td>
</tr>
<tr>
<td>type Fixed32s14</td>
<td>delta</td>
<td>2.0**(-14)</td>
</tr>
<tr>
<td>type Fixed32s15</td>
<td>delta</td>
<td>2.0**(-15)</td>
</tr>
<tr>
<td>type Fixed32s16</td>
<td>delta</td>
<td>2.0**(-16)</td>
</tr>
<tr>
<td>type Fixed32s17</td>
<td>delta</td>
<td>2.0**(-17)</td>
</tr>
<tr>
<td>type Fixed32s18</td>
<td>delta</td>
<td>2.0**(-18)</td>
</tr>
<tr>
<td>type Fixed32s19</td>
<td>delta</td>
<td>2.0**(-19)</td>
</tr>
<tr>
<td>type Fixed32s20</td>
<td>delta</td>
<td>2.0**(-20)</td>
</tr>
<tr>
<td>type Fixed32s21</td>
<td>delta</td>
<td>2.0**(-21)</td>
</tr>
<tr>
<td>type Fixed32s22</td>
<td>delta</td>
<td>2.0**(-22)</td>
</tr>
<tr>
<td>type Fixed32s23</td>
<td>delta</td>
<td>2.0**(-23)</td>
</tr>
<tr>
<td>type Fixed32s24</td>
<td>delta</td>
<td>2.0**(-24)</td>
</tr>
<tr>
<td>type Fixed32s25</td>
<td>delta</td>
<td>2.0**(-25)</td>
</tr>
<tr>
<td>type Fixed32s26</td>
<td>delta</td>
<td>2.0**(-26)</td>
</tr>
<tr>
<td>type Fixed32s27</td>
<td>delta</td>
<td>2.0**(-27)</td>
</tr>
<tr>
<td>type Fixed32s28</td>
<td>delta</td>
<td>2.0**(-28)</td>
</tr>
<tr>
<td>type Fixed32s29</td>
<td>delta</td>
<td>2.0**(-29)</td>
</tr>
<tr>
<td>type Fixed32s30</td>
<td>delta</td>
<td>2.0**(-30)</td>
</tr>
<tr>
<td>type Fixed32s31</td>
<td>delta</td>
<td>2.0**(-31)</td>
</tr>
<tr>
<td>type Fixed32s32</td>
<td>delta</td>
<td>2.0**(-32)</td>
</tr>
<tr>
<td>type Fixed32s33</td>
<td>delta</td>
<td>2.0**(-33)</td>
</tr>
<tr>
<td>type Fixed32s34</td>
<td>delta</td>
<td>2.0**(-34)</td>
</tr>
<tr>
<td>type Fixed32s35</td>
<td>delta</td>
<td>2.0**(-35)</td>
</tr>
<tr>
<td>type Fixed32s36</td>
<td>delta</td>
<td>2.0**(-36)</td>
</tr>
<tr>
<td>type Fixed32s37</td>
<td>delta</td>
<td>2.0**(-37)</td>
</tr>
<tr>
<td>type Fixed32s38</td>
<td>delta</td>
<td>2.0**(-38)</td>
</tr>
<tr>
<td>type Fixed32s39</td>
<td>delta</td>
<td>2.0**(-39)</td>
</tr>
<tr>
<td>type Fixed32s40</td>
<td>delta</td>
<td>2.0**(-40)</td>
</tr>
<tr>
<td>type Fixed32s41</td>
<td>delta</td>
<td>2.0**(-41)</td>
</tr>
<tr>
<td>type Fixed32s42</td>
<td>delta</td>
<td>2.0**(-42)</td>
</tr>
<tr>
<td>type Fixed32s43</td>
<td>delta</td>
<td>2.0**(-43)</td>
</tr>
<tr>
<td>type Fixed32s44</td>
<td>delta</td>
<td>2.0**(-44)</td>
</tr>
<tr>
<td>type Fixed32s45</td>
<td>delta</td>
<td>2.0**(-45)</td>
</tr>
<tr>
<td>type Fixed32s46</td>
<td>delta</td>
<td>2.0**(-46)</td>
</tr>
<tr>
<td>type Fixed32s47</td>
<td>delta</td>
<td>2.0**(-47)</td>
</tr>
<tr>
<td>type Fixed32s48</td>
<td>delta</td>
<td>2.0**(-48)</td>
</tr>
<tr>
<td>type Fixed32s49</td>
<td>delta</td>
<td>2.0**(-49)</td>
</tr>
<tr>
<td>type Fixed32s50</td>
<td>delta</td>
<td>2.0**(-50)</td>
</tr>
<tr>
<td>type Fixed32s51</td>
<td>delta</td>
<td>2.0**(-51)</td>
</tr>
<tr>
<td>type Fixed32s52</td>
<td>delta</td>
<td>2.0**(-52)</td>
</tr>
<tr>
<td>type Fixed32s53</td>
<td>delta</td>
<td>2.0**(-53)</td>
</tr>
<tr>
<td>type Fixed32s54</td>
<td>delta</td>
<td>2.0**(-54)</td>
</tr>
<tr>
<td>type Fixed32s55</td>
<td>delta</td>
<td>2.0**(-55)</td>
</tr>
<tr>
<td>type Fixed32s56</td>
<td>delta</td>
<td>2.0**(-56)</td>
</tr>
<tr>
<td>type Fixed32s57</td>
<td>delta</td>
<td>2.0**(-57)</td>
</tr>
<tr>
<td>type Fixed32s58</td>
<td>delta</td>
<td>2.0**(-58)</td>
</tr>
<tr>
<td>type Fixed32s59</td>
<td>delta</td>
<td>2.0**(-59)</td>
</tr>
<tr>
<td>type Fixed32s60</td>
<td>delta</td>
<td>2.0**(-60)</td>
</tr>
<tr>
<td>type Fixed32s61</td>
<td>delta</td>
<td>2.0**(-61)</td>
</tr>
<tr>
<td>type Fixed32s62</td>
<td>delta</td>
<td>2.0**(-62)</td>
</tr>
<tr>
<td>type Fixed32s63</td>
<td>delta</td>
<td>2.0**(-63)</td>
</tr>
<tr>
<td>type Fixed32s64</td>
<td>delta</td>
<td>2.0**(-64)</td>
</tr>
<tr>
<td>type Fixed32s65</td>
<td>delta</td>
<td>2.0**(-65)</td>
</tr>
<tr>
<td>type Fixed32s66</td>
<td>delta</td>
<td>2.0**(-66)</td>
</tr>
<tr>
<td>type Fixed32s67</td>
<td>delta</td>
<td>2.0**(-67)</td>
</tr>
<tr>
<td>type Fixed32s68</td>
<td>delta</td>
<td>2.0**(-68)</td>
</tr>
<tr>
<td>type Fixed32s69</td>
<td>delta</td>
<td>2.0**(-69)</td>
</tr>
<tr>
<td>type Fixed32s70</td>
<td>delta</td>
<td>2.0**(-70)</td>
</tr>
<tr>
<td>type Fixed32s71</td>
<td>delta</td>
<td>2.0**(-71)</td>
</tr>
<tr>
<td>type Fixed32s72</td>
<td>delta</td>
<td>2.0**(-72)</td>
</tr>
<tr>
<td>type Fixed32s73</td>
<td>delta</td>
<td>2.0**(-73)</td>
</tr>
<tr>
<td>type Fixed32s74</td>
<td>delta</td>
<td>2.0**(-74)</td>
</tr>
<tr>
<td>type Fixed32s75</td>
<td>delta</td>
<td>2.0**(-75)</td>
</tr>
<tr>
<td>type Fixed32s76</td>
<td>delta</td>
<td>2.0**(-76)</td>
</tr>
<tr>
<td>type Fixed32s77</td>
<td>delta</td>
<td>2.0**(-77)</td>
</tr>
<tr>
<td>type Fixed32s78</td>
<td>delta</td>
<td>2.0**(-78)</td>
</tr>
<tr>
<td>type Fixed32s79</td>
<td>delta</td>
<td>2.0**(-79)</td>
</tr>
<tr>
<td>type Fixed32s80</td>
<td>delta</td>
<td>2.0**(-80)</td>
</tr>
<tr>
<td>type Fixed32s81</td>
<td>delta</td>
<td>2.0**(-81)</td>
</tr>
<tr>
<td>type Fixed32s82</td>
<td>delta</td>
<td>2.0**(-82)</td>
</tr>
<tr>
<td>type Fixed32s83</td>
<td>delta</td>
<td>2.0**(-83)</td>
</tr>
<tr>
<td>type Fixed32s84</td>
<td>delta</td>
<td>2.0**(-84)</td>
</tr>
<tr>
<td>type Fixed32s85</td>
<td>delta</td>
<td>2.0**(-85)</td>
</tr>
<tr>
<td>type Fixed32s86</td>
<td>delta</td>
<td>2.0**(-86)</td>
</tr>
<tr>
<td>type Fixed32s87</td>
<td>delta</td>
<td>2.0**(-87)</td>
</tr>
<tr>
<td>type Fixed32s88</td>
<td>delta</td>
<td>2.0**(-88)</td>
</tr>
<tr>
<td>type Fixed32s89</td>
<td>delta</td>
<td>2.0**(-89)</td>
</tr>
<tr>
<td>type Fixed32s90</td>
<td>delta</td>
<td>2.0**(-90)</td>
</tr>
<tr>
<td>type Fixed32s91</td>
<td>delta</td>
<td>2.0**(-91)</td>
</tr>
<tr>
<td>type Fixed32s92</td>
<td>delta</td>
<td>2.0**(-92)</td>
</tr>
<tr>
<td>type Fixed32s93</td>
<td>delta</td>
<td>2.0**(-93)</td>
</tr>
<tr>
<td>type Fixed32s94</td>
<td>delta</td>
<td>2.0**(-94)</td>
</tr>
<tr>
<td>type Fixed32s95</td>
<td>delta</td>
<td>2.0**(-95)</td>
</tr>
<tr>
<td>type Fixed32s96</td>
<td>delta</td>
<td>2.0**(-96)</td>
</tr>
<tr>
<td>type Fixed32s97</td>
<td>delta</td>
<td>2.0**(-97)</td>
</tr>
<tr>
<td>type Fixed32s98</td>
<td>delta</td>
<td>2.0**(-98)</td>
</tr>
<tr>
<td>type Fixed32s99</td>
<td>delta</td>
<td>2.0**(-99)</td>
</tr>
<tr>
<td>type Fixed32s100</td>
<td>delta</td>
<td>2.0**(-100)</td>
</tr>
</tbody>
</table>
type Fixed64s8  is delta 2.0**(-8) range -2.0**23 .. 2.0**23 - 2.0**(-8);
type Fixed64s9  is delta 2.0**(-9) range -2.0**22 .. 2.0**22 - 2.0**(-9);
type Fixed64s10 is delta 2.0**(-10) range -2.0**21 .. 2.0**21 - 2.0**(-10);
type Fixed64s11 is delta 2.0**(-11) range -2.0**20 .. 2.0**20 - 2.0**(-11);
type Fixed64s12 is delta 2.0**(-12) range -2.0**19 .. 2.0**19 - 2.0**(-12);
type Fixed64s13 is delta 2.0**(-13) range -2.0**18 .. 2.0**18 - 2.0**(-13);
type Fixed64s14 is delta 2.0**(-14) range -2.0**17 .. 2.0**17 - 2.0**(-14);
type Fixed64s15 is delta 2.0**(-15) range -2.0**16 .. 2.0**16 - 2.0**(-15);
type Fixed64s16 is delta 2.0**(-16) range -2.0**15 .. 2.0**15 - 2.0**(-16);
            type Fixed64s24 is delta 2.0**(-0) range -2.0**31 .. 2.0**31 - 2.0**(-0);
            type Fixed64s30 is delta 2.0**(-0) range -2.0**31 .. 2.0**31 - 2.0**(-0);
            type Fixed64s32 is delta 2.0**(-0) range -2.0**31 .. 2.0**31 - 2.0**(-0);
            type Fixed64s45 is delta 2.0**(-0) range -2.0**31 .. 2.0**31 - 2.0**(-0);
            type Fixed64s127 is delta 2.0**(-0) range -2.0**31 .. 2.0**31 - 2.0**(-0);
            type Fixed64s127 is delta 2.0**(-0) range -2.0**31 .. 2.0**31 - 2.0**(-0);
            type Fixed64s127 is delta 2.0**(-0) range -2.0**31 .. 2.0**31 - 2.0**(-0);
    type Fixed2u1  is delta 2.0**(-1) range 0.0 .. 2.0**0 - 2.0**(-1);
    type Fixed9u0  is delta 2.0**(-0) range 0.0 .. 2.0**8 - 2.0**(-0);
    type Fixed9u3  is delta 2.0**(-3) range 0.0 .. 2.0**5 - 2.0**(-3);
    type Fixed11u4 is delta 2.0**(-4) range 0.0 .. 2.0**6 - 2.0**(-4);
    type Fixed11u10 is delta 2.0**(-10) range 0.0 .. 2.0**0 - 2.0**(-10);
    type Fixed11u10 is delta 2.0**(-10) range 0.0 .. 2.0**0 - 2.0**(-10);
    type Fixed15u12 is delta 2.0**(-12) range 0.0 .. 2.0**2 - 2.0**(-12);
    type Fixed16u0  is delta 2.0**(-0) range 0.0 .. 2.0**15 - 2.0**(-0);
    type Fixed16u1  is delta 2.0**(-1) range 0.0 .. 2.0**14 - 2.0**(-1);
    type Fixed16u2  is delta 2.0**(-2) range 0.0 .. 2.0**13 - 2.0**(-2);
    type Fixed16u3  is delta 2.0**(-3) range 0.0 .. 2.0**12 - 2.0**(-3);
    type Fixed16u4  is delta 2.0**(-4) range 0.0 .. 2.0**11 - 2.0**(-4);
    type Fixed16u5  is delta 2.0**(-5) range 0.0 .. 2.0**10 - 2.0**(-5);
    type Fixed16u6  is delta 2.0**(-6) range 0.0 .. 2.0**9 - 2.0**(-6);
    type Fixed16u7  is delta 2.0**(-7) range 0.0 .. 2.0**8 - 2.0**(-7);
    type Fixed16u8  is delta 2.0**(-8) range 0.0 .. 2.0**7 - 2.0**(-8);
    type Fixed16u9  is delta 2.0**(-9) range 0.0 .. 2.0**6 - 2.0**(-9);
    type Fixed16u10 is delta 2.0**(-10) range 0.0 .. 2.0**5 - 2.0**(-10);
    type Fixed16u11 is delta 2.0**(-11) range 0.0 .. 2.0**4 - 2.0**(-11);
    type Fixed16u12 is delta 2.0**(-12) range 0.0 .. 2.0**3 - 2.0**(-12);
    type Fixed16u13 is delta 2.0**(-13) range 0.0 .. 2.0**2 - 2.0**(-13);
    type Fixed16u14 is delta 2.0**(-14) range 0.0 .. 2.0**1 - 2.0**(-14);
    type Fixed16u15 is delta 2.0**(-15) range 0.0 .. 2.0**0 - 2.0**(-15);
    type Fixed16u16 is delta 2.0**(-16) range 0.0 .. 2.0**(-1) - 2.0**(-16);
    type Fixed17u3  is delta 2.0**(-3) range 0.0 .. 2.0**13 - 2.0**(-3);
    type Fixed21u11 is delta 2.0**(-11) range 0.0 .. 2.0**9 - 2.0**(-11);
    type Fixed23u10 is delta 2.0**(-10) range 0.0 .. 2.0**12 - 2.0**(-10);
    type Fixed25u8  is delta 2.0**(-8) range 0.0 .. 2.0**16 - 2.0**(-8);
    type Fixed30u10 is delta 2.0**(-10) range 0.0 .. 2.0**19 - 2.0**(-10);
    type Fixed32u28 is delta 2.0**(-28) range 0.0 .. 2.0**3 - 2.0**(-28);
    type Fixed32u29 is delta 2.0**(-29) range 0.0 .. 2.0**2 - 2.0**(-29);
    type Fixed32u31 is delta 2.0**(-31) range 0.0 .. 2.0**0 - 2.0**(-31);
    --
    type Fixed33u32 is delta 2.0**(-31) range 0.0 .. 2.0**0 - 2.0**(-31);

    subtype Cms2_Word is Integer;

    -- common variables
    
    First_Iter: Boolean;
    SX1 : Integer := 1;
    SX2 : Integer := 2;
    SX3 : Integer := 3;
    SX4 : Integer := 4;
    SX5 : Integer := 5;
    SX6 : Integer := 6;
    SX7 : Integer := 7;
    SX8 : Integer := 8;

    function "+" (Left : in Float;
               Right : in Integer)
        return Float;
    function "*"
(Left : in  Integer;
 Right : in  Float)
 return Float;

function "+"
(Left : in  Boolean;
 Right : in  Integer)
 return Integer;

function "+"
(Left : in  Integer;
 Right : in  Boolean)
 return Integer;

function "-"
(Left : in  Float;
 Right : in  Integer)
 return Float;

function "-"
(Left : in  Integer;
 Right : in  Float)
 return Integer;

function "-"
(Left : in  Boolean;
 Right : in  Integer)
 return Integer;

function "+"
(Left : in  Integer;
 Right : in  Boolean)
 return Integer;

function "+"
(Left : in  Float;
 Right : in  Integer)
 return Float;

function "+"
(Left : in  Integer;
 Right : in  Float)
 return Float;

function "*"
(Left : in  Boolean;
 Right : in  Integer)
 return Integer;

function "*"
(Left : in  Integer;
 Right : in  Boolean)
 return Integer;

function "/"
(Left : in  Integer;
 Right : in  Boolean)
 return Integer;

function "/"
(Left : in  Float;
 Right : in  Integer)
 return Float;

function "/"
(Left : in  Integer;
 Right : in  Float)
 return Float;

function "<"
(Left : in  Boolean;
 Right : in  Integer)
 return Boolean;

function "<"
(Left : in  Integer;
 Right : in  Float)
 return Boolean;

function "<"
(Left : in  Float;
 Right : in  Integer)
 return Boolean;

function "<"
(Left : in  Integer;
 Right : in  Float)
 return Boolean;

function ">"
(Left : in  Float;
 Right : in  Integer)
 return Boolean;

function ">"
(Left : in  Integer;
 Right : in  Float)
 return Boolean;

function ">="

(Left : in Float; Right : in Integer) return Boolean;
function ">=" (Left : in Integer; Right : in Float) return Boolean;
function "and" (Left : in Integer; Right : in Boolean) return Boolean;
function "and" (Left : in Boolean; Right : in Integer) return Boolean;
function "or" (Left : in Integer; Right : in Boolean) return Boolean;
function "or" (Left : in Boolean; Right : in Integer) return Boolean;
function Pad (Str : in String; Num : in Integer) return String;

-- function asin2(a: float; b: float) return float; /* MLEE: 09-11-94 */
-- function acos2(a: float; b: float) return float; /* MLEE: 09-11-94 */

-- fixed point arithmetic functions
-- function isgt(a: float) return float; /* MLEE: 09-11-94 */
-- function hln (a: float) return float; /* MLEE: 09-11-94 */
-- function lnl (a: float) return float; /* MLEE: 09-11-94 */
-- function iexp (a: float) return float; /* MLEE: 09-11-94 */
-- function isin (a: float) return float; /* MLEE: 09-11-94 */
-- function icos (a: float) return float; /* MLEE: 09-11-94 */
-- function isns (a: float) return float; /* MLEE: 09-11-94 */
-- function rad (a: float) return float; /* MLEE: 09-11-94 */

-- function sin(r: float) return float;
-- function cos(r: float) return float;
-- function tan(r: float) return float;
-- function log(r: float) return float;

-- pragma interface(fortran, sin);
-- pragma interface(fortran, cos);
-- pragma interface(fortran, tan);
-- pragma interface(fortran, log);

-- function Long_Flt_Image (R : in Long_Float) return String;

-- type Bit_String is array (Natural range <>) of Boolean;
pragma Pack (Bit_String);

subtype Bit_String_32 is Bit_String (0 .. 31);
subtype String4 is String (1..4);

function Space (N : in Integer) return String;
-- Conversion functions

function Bit_To_Integer (Bs : in Bit_String) return Integer;
function Integer_To_Bit (N : in Integer; Nb : in Integer) return Bit_String;
--function char_to_bit(c: in string) return bit_string;
function Int_to_Bool
(N : in Integer)
return Boolean;
--function int_to_bool(n: in unsigned_longword) return boolean;
function Int_to_Bool
(N : in Float)
return Boolean;
function Bool_to_Int
(P : in Boolean)
return Integer;
function Str_to_Int
(P : in String)
return Integer;
function Int_to_Str
(P : in Integer)
return String;

procedure Field_HProc_Integer
(Value : in Integer;
Bstart : in Integer;
Blength : in Integer;
Dest_Word : in out Cms2_Word);

procedure Field_HProc_Float
(Value : in Float;
Bstart : in Integer;
Blength : in Integer;
Dest_Word : in out Cms2_Word);

procedure Field_HProc_String
(Value : in String;
Bstart : in Integer;
Blength : in Integer;
Dest_Word : in out Cms2_Word);

function Field_HFcn_Integer
(Source_Word : in Cms2_Word;
Bstart : in Integer;
Blength : in Integer)
return Integer;

function Field_HFcn_Float
(Source_Word : in Cms2_Word;
Bstart : in Integer;
Blength : in Integer)
return Float;

function Field_HFcn_String
(Source_Word : in Cms2_Word;
Bstart : in Integer;
Blength : in Integer)
return String;

procedure Meu_Table_Word_Proc
(Value : in Integer;
Size_Dim1 : in Integer;
Size_Dim2 : in Integer;
Array_Addr : in Address);

procedure Meu_Table_Word_Proc
(Value : in Float;
Size_Dim1 : in Integer;
Size_Dim2 : in Integer;
Array_Addr : in Address);

procedure Meu_Table_Word_Proc
(Value : in String;
Size_Dim1 : in Integer;
Size_Dim2 : in Integer;
Array_Addr : in Address);

procedure Mdu_Item_Word_Proc
(Value : in Integer;
Size_Dim1 : in Integer;
Array_Addr : in Address);

procedure Mdu_Item_Word_Proc
(Value : in Float;
Size_Dim1 : in Integer;
procedure Mdu_Item_Word_Prc
(Value : in String;
Size_Dim1 : in Integer;
Array_Addr : in Address);

procedure Cms2_Input
(File : in String;
Format : in String;
Item_Num : in Integer;
item : out Integer);

procedure Cms2_Input
(File : in String;
Format : in String;
Item_Num : in Integer;
item : out Float);

procedure Cms2_Output
(File : in String;
Format : in String;
Item_Num : in Integer;
item : out String);

procedure Cms2_Output
(File : in String;
Format : in String;
Item_Num : in Integer;
item : in Float);

procedure Assign_Char_Substring
(Dest : in String;
Charfrom : in Integer;
Charto : in Integer;
Srce : in String);

procedure Assign_Bit_Substring
(Dest : in Cms2_Word;
Charfrom : in Integer;
Charto : in Integer;
Srce : in Integer);

procedure Swap_Data_Units
(Source : in Integer;
Receptacle : in Integer);

procedure Shift_Data_Unit_Circular
(Source : in Integer;
Samount : in Integer;
Receptacle : out Integer);

procedure Shift_Data_Unit_Logical
(Source : in Integer;
Samount : in Integer;
Receptacle : out Integer);

procedure Shift_Data_Unit_Algebraic
(Source : in Integer;
Samount : in Integer;
Receptacle : out Integer);

function Cms_2_Odd
(Expr : in Integer)
return Boolean;
function Cms_2_Evenp
  (Expr : in Integer)
  return Boolean;
function Cms_2_Invalid
  (Expr : in Integer)
  return Boolean;
function Cms_2_Valid
  (Expr : in Integer)
  return Boolean;
-- MLEE : 08 November 1994 : w/ Wu-hung for Implementation Demo:
function Load_Time_Func
  (Val : in Integer)
  return Integer;
function Load_Time_Func
  (Val : in Float)
  return Float;
function Load_Time_Func
  (Val : in String)
  return String;
-- MLEE : 09 November 1994 : Built-in function implementation:
-- based on Wu-hung's summary.
-- Absolute value:
-- function abs(signed_integer : in integer) return integer;
-- function abs(signed_float : in float) return float;
-- Bit string selection:
function Bit
  (Data_Unit : in Cms2_Word;
   Starting_Bit_No : in Integer)
  return Integer;
function Bit
  (Data_Unit : in Cms2_Word;
   Starting_Bit_No : in Integer;
   No_Of_Bit : in Integer)
  return Integer;
-- Character string selection:
function Char
  (Data_Unit : in String;
   Starting_Char_No : in Integer)
  return String;
function Char
  (Data_Unit : in String;
   Starting_Char_No : in Integer;
   No_Of_Chars : in Integer)
  return String;
-- Bit count:
function Cnt
  (Bit_Val : in Cms2_Word)
  return Integer;
-- Memory address of a data unit:
function Corad
  (Data_Unit : in Cms2_Word)
  return Address;
-- Scaling:
function Scal
  (Scale_Factor : in Integer)
  return Cms2_Word;
function Scal
  (Scale_Factor : in Integer;
   Scale_Val : in Cms2_Word)
  return Cms2_Word;
-- Data type conversion:
function Confg
  (Type_Spec : in String)
  return Cms2_Word;
function Confg
  (Type_Spec : in String;
   Convert_Val : in Cms2_Word)
  return Cms2_Word;
-- Temporary definition:
-- function tdef(type_spec : in string) return integer;
-- function tdef(type_spec : in string;
--     bit_str : in integer) return integer;

function Tdef (Type_Spec : in String)
    return Integer;
function Tdef (Type_Spec : in String;
    Bit_Str : in Integer)
    return Integer;

-- Remainder::
function Remndr
    (Operand1 : in Float)
    return Float;

-- Subfile number::
function Fil
    (File_Name : in Cms2_Word)
    return Integer;

-- Subfile position (record number of current subfile)::
function Pos
    (File_Name : in Cms2_Word)
    return Integer;

-- Length of the current record in the named file::
function Length
    (File_Name : in Cms2_Word)
    return Integer;

-- Logical AND::
function Andf
    (Operand1 : in Cms2_Word;
    Operand2 : in Cms2_Word)
    return Cms2_Word;
-- function andf(Operand1 : in unsigned_longword;
--     operand2 : in unsigned_longword) return cms2_word;

-- Logical OR::
function Orf
    (Operand1 : in Cms2_Word;
    Operand2 : in Cms2_Word)
    return Cms2_Word;

-- Logical XOR::
function Xorf
    (Operand1 : in Cms2_Word;
    Operand2 : in Cms2_Word)
    return Cms2_Word;

-- One's complementation::
function Compf
    (Operand : in Cms2_Word)
    return Cms2_Word;

-- Fixed point arithmetic function::
-- Square root::
function Isqrt
    (Operand : in Float)
    return Float;

-- Half natural logarithm::
function Hln
    (Operand : in Float)
    return Float;

-- Natural logarithm::
function Ln
    (Operand : in Float)
    return Float;

-- Exponential::
function Iexp
    (Operand : in Float)
return Float;
-- sine:
function isin
  (Operand : in  Float)
  return Float;
-- cosine:
function icos
  (Operand : in  Float)
  return Float;
-- radian to BAMS conversion:
function Bams
  (Operand : in  Float)
  return Float;
-- radian to BAMS conversion:
function Rad
  (Operand : in  Float)
  return Float;
-- Float point arithmetic function:
-- sine:  function sin (operand : in float) return float;
-- cosin:  function cos (operand : in float) return float;
-- tangent:  function tan (operand : in float) return float;
-- inverse sine:  function asin(operand : in float) return float;
-- inverse cosine:  function acos(operand : in float) return float;
-- inverse tangent:  function atan(operand : in float) return float;
-- exponential:  function exp (operand : in float) return float;
-- natural logarithm:  function alog(operand : in float) return float;
-- square root:  function sqrt(operand : in float) return float;
-- inverse sine:
function Asin2
  (Operand1 : in  Float;
Operand2 : in  Float)
  return Float;
-- inverse cosine:
function Acos2
  (Operand1 : in  Float;
Operand2 : in  Float)
  return Float;
-- inverse tangent:
--function atan2(operand1 : in float;
--operand2 : in float) return float;
-- Successor::
function Succ
  (Operand : in  Integer)
  return Integer;
-- Successor::
function Pred
  (Operand : in  Integer)
  return Integer;
-- Initial value::
function First
  (Status_Type_Name : in  String)
  return Integer;
-- Final value::
function Final
  (Status_Type_Name : in  String)
  return Integer;
-- Logical shift left/right::
function Shiftll
  (Shift_Val : in  Cms2_Word)
  return Cms2_Word;
function Shiftlr
  (Shift_Val : in  Cms2_Word)
  return Cms2_Word;
-- Circular shift left/right::
function Shiftcl
  (Shift_Val : in  Cms2_Word)
  return Cms2_Word;
function Shiftc 
(Shift_Val : in  Cms2_Word) 
return Cms2_Word;

function Address_To_Integer is new Unchecked_Conversion 
(Source => Address, 
Target => Integer);

function Address_ToUnsigned is new Unchecked_Conversion 
(Source => Address, 
Target => Unsigned_Longword);

procedure Cms2_Execute 
(S_Num : in  Integer);

procedure Cms2_Execute 
(S_Num : in  Integer; 
Num : in  Float);

function Cms2_Data_Init 
(P1 : in  Integer; 
P2 : in  Integer; 
P3 : in  Integer; 
P4 : in  Integer) 
return Cms2_Word;

function Cms2_Data_Init 
(P1 : in  Integer; 
P2 : in  Integer; 
P3 : in  Integer; 
P4 : in  Integer) 
return Cms2_Word;

function Cms2_Data_Init 
(P1 : in  Float; 
P2 : in  Integer; 
P3 : in  Integer; 
P4 : in  Integer) 
return Cms2_Word;
end Cms2_To_Ada_Predefined;
ADA REENGINEERING OF MK-2 CODE BY HAND

-- The purpose of this module is to update the Predicted Track Table to the
-- current time based on the observed position and speed of the track.
--
-- The original CMS-2 module performs this task for a single indexed entry,
-- with some external unit performing the update for the whole table. The
-- body of this package iterates over the entire table.
-- This module requires another function to be responsible for updating the
-- Observed Track Table as well as the Own Ship position.
-- Additional reengineering for better integration into the system is desirable.
--
with Ada.Calendar; use Ada.Calendar;
with Ada.Numerics; use Ada.Numerics;
package MK2 is

MK2_Table_Size: Constant := 99; -- allows easy increase of size for track tables

type MK2_Float_Type is new Float; -- allow to be implementation defined
subtype Distance_Type is MK2_Float_Type; -- Distance in yards
subtype Velocity_Type is MK2_Float_Type; -- in yards/second
subtype Radians_Type is MK2_Float_Type; -- in radians
subtype Longitude_Type is MK2_Float_Type range -Pi/2.0 .. Pi/2.0; -- in radians
subtype Latitude_Type is MK2_Float_Type range -Pi .. Pi; -- in radians

Own_Ship_X_Position: Distance_Type := 0.0;
Own_Ship_Y_Position: Distance_Type := 0.0;
Own_Ship_Latitude: Latitude_Type := +32.0 * Pi/180.0;
Own_Ship_Longitude: Longitude_Type := -120.0 * Pi/180.0;

type Observed_Track_Table is
  record
    X: Distance_Type; -- Observed X position
    Y: Distance_Type; -- Observed Y position
    X_Velocity: Velocity_Type; -- Observed X component of velocity
    Y_Velocity: Velocity_Type; -- Observed Y component of velocity
  end record;

type Predicted_Track_Table is
  record
    X: Distance_Type; -- Predicted X position
    Y: Distance_Type; -- Predicted Y position
    Rng: Distance_Type; -- Predicted Range from Own Ship
    Brg: Radians_Type; -- Predicted Bearing from Own Ship
    Latitude: Latitude_Type; -- Predicted Latitude
    Longitude: Longitude_Type; -- Predicted Longitude
  end record;

Observed_Track: array (0 .. MK2_Table_Size) of Observed_Track_Table;
Predicted_Track: array (0 .. MK2_Table_Size) of Predicted_Track_Table;

procedure Compute_Track_Lat_Lng
  (Rng, Brg, Lat, Lng : in Distance_Type;
   Computed_Latitude, Computed_Longitude: out Latitude_Type);

procedure Compute_Bearing_Range
  (X1, Y1, X2, Y2, Rng, Brg : in Distance_Type;
   Computed_Latitude, Computed_Longitude: out Latitude_Type);

procedure Predict_Track_Position

M-49
end MK2;

(Old_X : in Distance_Type;
Old_Y : in Distance_Type;
X_Velocity : in Velocity_Type;
Y_Velocity : in Velocity_Type;
New_X : out Distance_Type;
New_Y : out Distance_Type);
with Ada.Numerics.Generic_Elementary_Functions;
package body MK2 is

package MK2_Numbers is new Ada.Numerics.Generic_Elementary_Functions
(Float_Type => MK2_Float_Type);
use MK2_Numbers;

procedure Predict_Track_Position
(Old_X : in Distance_Type;
 Old_Y : in Distance_Type;
 X_Velocity : in Velocity_Type;
 Y_Velocity : in Velocity_Type;
 New_X : out Distance_Type;
 New_Y : out Distance_Type) is

-- The Predict_Track_Position procedure will compute a predicted X and Y position
-- to the current time based on the old position and the time of observation for
-- the old position.

Delta_Time: Duration;

begin

-- Compute Fire Control Predicted Track X and Y Positions
-- Note: Not only handles time across days, but also handles Y2000 problem
-- Type Duration is implementation defined; possible exception if too large
-- Assume Delta_Time nominally less than 24 hours?
New_X := Old_X + X_Velocity * MK2_Float_Type(Delta_Time);
New_Y := Old_Y + Y_Velocity * MK2_Float_Type(Delta_Time);

end Predict_Track_Position;

procedure Compute_Bearing_Range
(X1 : in Distance_Type;
 Y1 : in Distance_Type;
 X2 : in Distance_Type;
 Y2 : in Distance_Type;
 Rng : out Distance_Type;
 Brg : out Radians_Type) is

-- procedure Compute_Bearing_Range computes the bearing and range from an
-- input position (X1, Y1) to the input position (X2, Y2).

begin

-- Compute Fire Control System Position Kept Track Range
Rng := Sqrt ((X2-X1)**2 + (Y2-Y1)**2);
If (Rng > 999999.0) then
  Rng := 999999.0;  -- Clip Track range to Maximum??????????
end if;

-- Compute Fire Control System Position Kept Track Bearing
If (Abs(X2-X1) < 0.00001) and (Abs(Y2-Y1) < 0.00001) then
  -- Possible error in original CMS - should use Abs function
  Brg := 0.0;
else
  Brg := Arctan ((Y2-Y1), (X2-X1));
end if;

end Compute_Bearing_Range;

M-51
procedure Compute_Track_Lat_Lng
( Rng : in Distance_Type;
  Brg : in Radians_Type;
  Lat : in Latitude_Type;
  Lng : in Longitude_Type;

  Computed_Latitude : out Latitude_Type;
  Computed_Longitude : out Longitude_Type) is

  Theta = Range / Earth_Radius;
  Latitude = Arcsin [Sin(Lat)*Cos(Theta) + Cos(Lat)*Sin(Theta)*Cos(Brg)];
  Longitude = Arctan [sin(Theta)*Sin(Brg),
  Cos(Lat)*Sin(Theta) - Sin(Lat)*Sin(Theta)*Cos(Brg)] - Lng;

  Earth_Radius: constant Distance_Type := 6.375_563.33; -- in yards
  Theta: Radians_Type;
  Arg1, Arg2: MK2_Float_Type;

begin

  Theta := Radians_Type(Rng/Earth_Radius);
  Computed_Latitude := Arcsin [Sin(Lat)*Cos(Theta) +
  Cos(Lat)*Sin(Theta)*Cos(Brg)];
  Arg1 := Sin(Theta)*Sin(Brg);
  Arg2 := Cos(Lat)*Sin(Theta)-Sin(Lat)*Sin(Theta)*Cos(Brg);
  if (abs(Arg1) < 0.00001) and (abs(Arg2) < 0.00001) then

    -- Again possible error in original not using abs function
    Computed_Longitude := 0.0 - Lng;

  else

    Computed_Longitude := Arctan (Arg2, Arg1) - Lng;

  end if;

  if (Computed_Longitude > PI) then -- Bound longitude from -PI to PI.
    Computed_Longitude := Computed_Longitude - 2.0*PI;

  end if;

end Compute_Track_Lat_Lng;

end -- package MK2;

-- Assumes table for Observed_Track is full
-- Then compute table for Predicted_Track
-- Actually in CMS-2 code, some external driver causes the looping for each index
-- There is probably a mechanism to ignore null Tracks in the table

for I in Predicted_Track'range loop -- Original CMS-2 performs this for one index

  -- Compute Predicted Track Position

Predict_Track_Position
(Old_X, => Observed_Track(I).X,
Old_Y, => Observed_Track(I).Y,
X_Velocity, => Observed_Track(I).X_Velocity,
Y_Velocity, => Observed_Track(I).Y_Velocity,
Time_of_Last_Update, => Observed_Track(I).Time_of_Last_Update,
New_X, => Predicted_Track(I).X,
New_Y, => Predicted_Track(I).Y);

-- Compute predicted range and bearing from own ship's position

Compute_Bearing_Range
(X1, => Own_Ship_X_Position,
Y1, => Own_Ship_Y_Position,
X2, => Predicted_Track(I).X,
Y2, => Predicted_Track(I).Y,
Rng, => Predicted_Track(I).Rng,
Brg, => Predicted_Track(I).Brg);

-- Compute Predicted Track Latitude and Longitude

Compute_Track_Lat_Lng
(Rng, => Predicted_Track(I).Rng,
Brg, => Predicted_Track(I).Brg,

M-52
-- Mapping of CMS-2 names to Ada 95 names

-- 1. Identifiers

-- COSBRG intermediate not used
-- COSLAT1 intermediate not used
-- COSTHET intermediate not used
-- FKPI becomes Pi [Ada.Numerics.Pi]
-- FKPI2 becomes 2*Pi; compiler will optimize
-- FTCONDAT becomes Earth_Radius
-- Apparently constant maintained in a table of CMS-2 constants
-- CCC translator converts to (array 0..98, 0..0) of CMS2_Word
-- FTCSS becomes Track
-- FTETKS becomes Predicted_Track
-- FVBRG becomes Bearing in Predicted_Track
-- FVEQRADG becomes Earth_Radius
-- FVNRG becomes Rng in Predicted_Track
-- FVVLGALT becomes Latitude in Predicted_Track
-- FVVGTLON becomes Longitude in Predicted_Track
-- FVTIME becomes Time_of_Last_Update in Observed_Track
-- FVTXFP 1 becomes X in Observed_Track
-- FVTXFP 2 becomes X in Predicted_Track
-- FVTXV becomes X_Velocity in Observed_Track
-- FVTYFP 1 becomes Y in Observed_Track
-- FVTYFP 2 becomes Y in Predicted_Track
-- FVTYV becomes Y_Velocity in Observed_Track
-- ICNX becomes T
-- RBLLTHET becomes Theta
-- SINCBRG intermediate not used
-- SINCALT1 intermediate not used
-- SINTHET intermediate not used
-- SUDVBRG becomes Brg
-- SUDVLAT1 becomes Lat
-- SUDVLAT2 becomes Computed_Latitude
-- SUDVLON1 becomes Long
-- SUDVLON2 becomes Computed_longitude
-- SUDVRNG becomes Rng
-- SUDVTIME becomes Delta_Time
-- SUDVOSLT becomes Own_Ship_Latitude
-- SUDVOSLN becomes Own_Ship_Longitude
-- SUDVOSXP becomes Own_Ship_X_Position
-- SUDVOSYP becomes Own_Ship_Y_Position
-- SUDVRAD1 becomes null (an intermediate computation)
-- SUDVRAD2 becomes null (an intermediate computation)
-- SUDVTIME becomes the function Ada.Calendar.Clock
-- TEMPARG intermediate not used
-- TGLTALT intermediate not used
-- TGLTALT intermediate not used
-- VRAD1 becomes null (an intermediate computation)
-- VRAD2 becomes null (an intermediate computation)

-- 2. Procedures

-- SUDPATAN not needed as converted to simple if then else test
-- SUDPRCS becomes Predict_Track_Position and Compute_Bearing_Range
-- SUDPRCLL becomes Compute_Track_Lat_Lng
-- 3. CMS-2 Math functions provided by Ada 95 Package MK2_Numerics generic

-- Ada.Numerics defines Pi, e,
-- Child package defines
-- Sqr, Log, Exp, **,
-- Sin, Cos, Tan, Cot,
-- Arcsin, Arccos, Arctan, Arccot
-- Sigh, Cosh, Tanh, Coth
-- Arcsign, Arccosh, Arctanh, Coth
The objective of this evaluation was to determine the maturity of the CMS–2 to Ada translators and associated tools, to determine the capabilities of these translators, and to provide information to CMS–2 project managers to assist them in the evaluation of costs and risks of translating CMS–2 to Ada.
<table>
<thead>
<tr>
<th>21a. NAME OF RESPONSIBLE INDIVIDUAL</th>
<th>21b. TELEPHONE (include Area Code)</th>
<th>21c. OFFICE SYMBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hans Mumm</td>
<td>(619) 553-4004</td>
<td>Code D4122</td>
</tr>
<tr>
<td></td>
<td>e-mail: <a href="mailto:mumm@nosc.mil">mumm@nosc.mil</a></td>
<td></td>
</tr>
</tbody>
</table>
INITIAL DISTRIBUTION

Code D0012 Patent Counsel (1)
Code D0271 Archive/Stock (6)
Code D0274 Library (2)
Code D027 M. E. Cathcart (1)
Code D0271 D. Richter (1)
Code D4122 H. Mumm (20)

Defense Technical Information Center
Fort Belvoir, VA 22060-6218 (4)

NCCOSC Washington Liaison Office
Arlington, VA 22245-5200

Center for Naval Analyses
Alexandria, VA 22302-0268

Navy Acquisition, Research and Development
Information Center (NARDIC)
Arlington, VA 22244-5114

GIDEP Operations Center
Corona, CA 91718-8000