WRDC-TR-90-8007
Volume VIII
Part 23

INTEGRATED INFORMATION SUPPORT SYSTEM (IISS)
Volume VIII - User Interface Subsystem
Part 23 - Rapid Application Generator and Report Writer
Development Specification

S. Barker

Control Data Corporation
Integration Technology Services
2970 Presidential Drive
Fairborn, OH 45324-6209

September 1990

Final Report for Period 1 April 1987 - 31 December 1990

Approved for Public Release; Distribution is Unlimited

MANUFACTURING TECHNOLOGY DIRECTORATE
WRIGHT RESEARCH AND DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6533

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, regardless whether or not the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data. It should not, therefore, be construed or implied by any person, persons, or organization that the Government is licensing or conveying any rights or permission to manufacture, use, or market any patented invention that may in any way be related thereto.

This technical report has been reviewed and is approved for publication.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations

David Judson

DAVID L. JUDSON, Project Manager
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 91
DATE

FOR THE COMMANDER:

Bruce Rasmussen

BRUCE A. RASMUSSEN, Chief
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 91
DATE

If your address has changed, if you wish to be removed form our mailing list, or if the addressee is no longer employed by your organization please notify WRDC/MTI, Wright-Patterson Air Force Base, OH 45433-6533 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

Unclassified

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for Public Release;
Distribution is Unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
DS 620344501

5. MONITORING ORGANIZATION REPORT NUMBER(S)
WRDC/MTI-TR-90-8007 Vol. VIII, Part 23

6a. NAME OF PERFORMING ORGANIZATION
Control Data Corporation;
Integration Technology Services

6b. OFFICE SYMBOL *(if applicable)*

7a. NAME OF MONITORING ORGANIZATION
WRDC/MTI

6c. ADDRESS *(City,State, and ZIP Code)*
2970 Presidential Drive
Fairborn, OH 45324-6209

7b. ADDRESS *(City, State, and ZIP Code)*
WPAFB, OH 45433-6533

8a. NAME OF FUNDING/SPONSORING ORGANIZATION
Wright Research and Development Center,
Air Force Systems Command, USAF

8b. OFFICE SYMBOL *(if applicable)*
WRDC/MTI

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUM.
F33600-87-C-0464

8c. ADDRESS *(City, State, and ZIP Code)*
Wright-Patterson AFB, Ohio 45433-6533

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
78011F	595600	F95600	20950607

11. TITLE See block 19
Rapid Ap pment Spec.

12. PERSONAL AUTHOR(S)
Structural Dynamics Research Corporation: Barker, S. , et al.

13a. TYPE OF REPORT
Final Report

13b. TIME COVERED
4/1/87 3/31/90

14. DATE OF REPORT *(Yr.,Mo.,Day)*
1990 September 30

15. PAGE COUNT
41

16. SUPPLEMENTARY NOTATION
WRDC/MTI Project Priority 6203

17. COSATI CODES

FIELD	GROUP	SUB GR.
1308	0905	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify block no.)*

19. ABSTRACT *(Continue on reverse if necessary and identify block number)*

This specification establishes the development, test, qualification, and performance requirements of a computer program identified as the Rapid Application Generator (RAP).

BLOCK 11:

INTEGRATED INFORMATION SUPPORT SYSTEM
Vol VIII - User Interface Subsystem

Part 23 - Rapid Application Generator and Report Writer Development
Specification

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED x SAME AS RPT. DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL
David L. Judson

22b. TELEPHONE NO. *(Include Area Code)*
(513) 255-7371

22c. OFFICE SYMBOL
WRDC/MTI

EDITION OF 1 JAN 73 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

FOREWORD

This technical report covers work performed under Air Force Contract F33600-87-C-0464, DAPro Project. This contract is sponsored by the Manufacturing Technology Directorate, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. It was administered under the technical direction of Mr. Bruce A. Rasmussen, Branch Chief, Integration Technology Division, Manufacturing Technology Directorate, through Mr. David L. Judson, Project Manager. The Prime Contractor was Integration Technology Services, Software Programs Division, of the Control Data Corporation, Dayton, Ohio, under the direction of Mr. W. A. Osborne. The DAPro Project Manager for Control Data Corporation was Mr. Jimmy P. Maxwell.

The DAPro project was created to continue the development, test, and demonstration of the Integrated Information Support System (IISS). The IISS technology work comprises enhancements to IISS software and the establishment and operation of IISS test bed hardware and communications for developers and users.

The following list names the Control Data Corporation subcontractors and their contributing activities:

SUBCONTRACTOR	ROLE
Control Data Corporation	Responsible for the overall Common Data Model design development and implementation, IISS integration and test, and technology transfer of IISS.
D. Appleton Company	Responsible for providing software information services for the Common Data Model and IDEF1X integration methodology.
ONTEK	Responsible for defining and testing a representative integrated system base in Artificial Intelligence techniques to establish fitness for use.
Simpact Corporation	Responsible for Communication development.
Structural Dynamics Research Corporation	Responsible for User Interfaces, Virtual Terminal Interface,and Network Transaction Manager design, development, implementation, and support.
Arizona State University	Responsible for test bed operations and support.

Table of Contents

Table of Contents

List of Illustrations

SECTION 1

SCOPE

1.1 Identification

This specification establishes the development, test, qualification, and performance requirements of a computer program identified as the Rapid Application Generator (RAP). The RAP is one configuration item of the Integrated Information Support System (IISS) User Interface (UI).

1.2 Functional Summary

This Computer Program Configuration Item (CPCI) is used to facilitate development of screen driven interactive programs accessing databases through the Common Data Model (CDM).

The major functions of the Rapid Application Generator are:

1) Provide a screen driven interface to database application programs;

2) Provide a screen based means of displaying the contents of a database;

3) Provide a context within which a single application program can switch between modes of database access: update, query, deletion, and insertion;

4) Allow the application developer to apply human engineering to the means by which the user dialogues with the database program.

SECTION 2

DOCUMENTS

2.1 Reference Documents

[1] Structural Dynamics Research Corporation, IISS Terminal Operator Guide, OM 620344000, 31 May 1988.

[2] Structural Dynamics Research Corporation, IISS Form Processor User Manual, UM 620344200, 31 May 1988.

[3] Control Data Corporation,System Design Specification, 31 May 1988.

[4] Structural Dynamics Research Corporation, Forms Language Compiler Development Specification, DS 620344401, 31 May 1988.

[5] Structural Dynamics Research Corporation, Report Writer Development Specification, DS 620344501, 31 May 1988.

[6] Structural Dynamics Research Corporation, User Interface Services Development Specification, DS 620344100, 31 May 1988.

[7] Structural Dynamics Research Corporation, Form Processor Development Specification, DS 620344200, 31 May 1988.

[8] Systran, ICAM Documentation Standards, IDS150120000C, 15 September 1983.

[9] Control Data Corporation, NDML Programmer's Reference Manual, PRM620341200, 31 May 1988.

2.2 Terms and Abbreviations

Abbreviation id: a user defined abbreviation of a CDM user view that may be used as a qualifier for a data item instead of the user view name itself.

Application Definition Language: a subset of the Forms Definition Language that includes retrieval of database information and conditional actions. It is used to define interactive application programs.

Application Interface: (AI), subset of the IISS User Interface that consists of the callable routines that are linked with applications that use the Form Processor or Virtual Terminal. The AI enables applications to be hosted on computers other than the host of the User Interface.

Application Process: (AP), a cohesive unit of software that can be initiated as a unit to perform some function or functions.

Application id: a user defined name which is used as the root of the file name of the generated application.

Attribute: field characteristic such as blinking, highlighted, black, etc. and various other combinations. Background attributes are defined for forms or windows only. Foreground attributes are defined for items. Attributes may be permanent, i.e., they remain the same unless changed by the application program, or they may be temporary, i.e., they remain in effect until the window is redisplayed.

Common Data Model: (CDM), IISS subsystem that describes common data application process formats, form definitions, etc. of the IISS and includes conceptual schema, external schemas, internal schemas, and schema transformation operators.

Computer Program Configuration Item: (CPCI), an aggregation of computer programs or any of their discrete portions, which satisfies an end-use function.

Conceptual Schema: (CS), the standard definition used for all data in the CDM. It is based on IDEF1 information modelling.

Cursor Position: the position of the cursor after any command is issued.

C code: a user defined sequence of C statements which are to be included in the generated application.

Data Item: a field of a record in the External Schema.

Device Drivers: (DD), software modules written to handle I/O for a specific kind of terminal. The modules map terminal specific commands and data to a neutral format. Device Drivers are part of the UI Virtual Terminal.

Display List: a list of all the open forms that are currently being processed by the Form Processor or the user.

External Schema: (ES), an application's view of the CDM's conceptual schema.

Field: two dimensional space on a terminal screen.

Field Pointer: indicates the ITEM which contains the current cursor position.

Field name: a qualified name enclosed in single quotes which denotes a field in the form processor display list.

Flag id: a user defined name which contains a boolean valued state indicator which may be used in condition and set expressions.

Form: structured view which may be imposed on windows or other forms. A form is composed of fields. These fields may be defined as forms, items, and windows.

Form Definition: (FD), forms definition language after compilation. It is read at run time by the Form Processor.

Form Definition Language: (FDL), the language in which electronic forms are defined.

Form Hierarchy: a graphic representation of the way in which forms, items and windows are related to their parent form.

Form Processor: (FP), subset of the IISS User Interface that consists of a set of callable execution time routines available to an application program for form processing.

Forms Language Compiler: (FLAN), subset of the FE that consists of a batch process that accepts a series of forms definition language statements and produces form definition files as output.

Form id: a name which denotes a form specified in a FDL file.

IISS Function Screen: the first screen that is displayed after logon. It allows the user to specify the function he wants to access and the device type and device name on which he is working.

Integrated Information Support System: (IISS), a computing environment used to investigate, demonstrate, test the concepts and produce application for information management and information integration in the context of Aerospace Manufacturing. The IISS addresses the problems of integration of data resident on heterogeneous data bases supported by heterogeneous computers interconnected via a Local Area Network.

Item: non-decomposable area of a form in which hard-coded descriptive text may be placed and the only defined areas where user data may be input/output.

Item name: a qualified name which denotes an item field.

Key id: a user defined name for a function key which is how the key is referenced within an ADL program.

Logical Device: a conceptual device that identifies a top level window of an application. It is used to distinguish between multiple applications running simultaneously on a physical device.

Message: descriptive text which may be returned in the standard message line on the terminal screen. They are used to warn of errors or provide other user information.

Message Line: a line on the terminal screen that is used to display messages.

Network Transaction Manager: (NTM), IISS subsystem that performs the coordination, communication and housekeeping functions required to integrate the Application Processes and System Services resident on the various hosts into a cohesive system.

Neutral Data Manipulation Language: (NDML), the command language by which the CDM is accessed for the purpose of extracting, deleting, adding, or modifying data.

Open List: a list of all the forms that are currently open for an application process.

Operating System: (OS), software supplied with a computer which allows it to supervise its own operations and manage access to hardware facilities such as memory and peripherals.

Page: instance of forms in windows that are created whenever a form is added to a window.

Paging and Scrolling: a method which allows a form to contain more data than can be displayed with provisions for viewing any portion of the data buffer.

Parameter form id: the name of a form which is to be displayed when the application is started from the IISS function screen and allows the user to specify run time values for the application.

Presentation Schema: (PS), may be equivalent to a form. It is the view presented to the user of the application.

Procedure id: the name of a user supplied or system service procedure which is to be called by the generated application.

Qualified Name: the name of a form, item or window preceded by the hierarchy path so that it is uniquely identified.

Rapid Application Generator: (RAP), part of the Application Generator that generates source code for interactive programs based on a language input.

Subform: a form that is used within another form.

Table id: the name of a CDM user view.

User Data: data which is either input by the user or output by the application programs to items.

User Interface: (UI), IISS subsystem that controls the user's terminal and interfaces with the rest of the system. The UI consists of two major subsystems: the User Interface Development System (UIDS) and the User Interface Management System (UIMS).

User Interface Management System: (UIMS), the runtime UI. It consists of the Form Processor, Virtual Terminal, Application Interface, the User Interface Services and the Text Editor.

User Interface Monitor: (UIM), part of the Form Processor that handles messaging between the NTM and the UI. It also provides authorization checks and initiates applications.

User Interface Services: (UIS), subset of the IISS User Interface that consists of a package of routines that aid users in controlling their environment. It includes message management, change password, and application definition services.

User Interface/Virtual Terminal Interface: (UI/VTI), another name for the User Interface.

User View: a data record in the External Schema.

Window: dynamic area of a terminal screen on which predefined forms may be placed at run time.

Window Manager: a facility which allows the following to be manipulated: size and location of windows, the device on which an application is running, the position of a form within a window. It is part of the Form Processor.

Window name: a qualified name which denotes a window.

SECTION 3

REQUIREMENTS

3.1 Computer Program Definition

The Rapid Application Generator (RAP) is a compiler which translates Form Definition Language source files into binary Form Definition format files and creates source for an application program which accesses databases via the CDM.

While RAP is normally invoked from the IISS function screen, another version is available which can be invoked from the host system. This second version is required so the user may specify a CDM at compile time.

Applications written prior to version 2.3 will require translation to the current syntax. A translation utility, GAPTRAN is used for this purpose. GAPTRAN translates those applications which are syntactically and semantically correct for versions prior to 2.3. GAPTRAN is invoked from the host system and then the user specifies the input and output files.

3.1.1 System Capacities

The RAP is written in C and NDML. It runs on a DEC VAX minicomputer under the VMS operating system.

3.1.2 Interface Requirements

The RAP may be invoked from the IISS function screen or from the host system. In either case the user specifies a Forms Definitions Language (FDL) source file to be compiled. When invoked from IISS the standard CDM is used. When invoked from the host system the user must specify the CDM.

The input FDL source file must correspond to the syntax in Appendix A of [4].

The output consists of binary Form Definition files whose format is constrained to agree with that expected by the Form Processor configuration item. The output C file is syntactically and semantically acceptable to the DIGITAL VAX 11 C compiler. The output NDML file is syntactically and semantically acceptable input for the NDML Precompiler configuration item and the DIGITAL COBOL compiler. Error messages are directed to the user's terminal.

GAPTRAN is invoked from the host system. The user is prompted for the name of the file to be translated and the name of the file to write the translation to. The input to GAPTRAN must be syntactically and semantically correct for version 2.2.5 RAP. The output will be acceptable to the version 2.3 RAP configuration item.

3.1.2.1 Interface Block Diagram

The interface block diagram for the RAP is shown in Figure 3-1.

```
   Key: +-------+  +----------+
        | Data  |  | Process  |
        +-------+  +----------+

              +-------------+
              | MYAPPL.FDL  |
              |             |
              | Source File |
              +------+------+
                     V
              +-------------+        +-------------+
              | Application |        |     CDM     |
              |             |        |             |
              |   Program   |<-------+    Data     |
              |             |        |             |
              |  Generator  |        | Dictionary  |
              +------+------+        +-------------+
        +------------+-------------------+
        V            V                   V
  +-----+-----+  +---+---+         +-----+-----+
  | Generated |  |       |         | Generated |
  |           |  |       |         |           |
  |     C     |  | F1.FD |         |   COBOL   |
  |           |  |       |         |           |
  |  Program  |  |       |         |  Program  |
  +-----+-----+  +-------+         +-----+-----+
        V                                V
```

```
+-------------+               +-----------------+
| C compiler  |               | CDM precompiler |
+-----+-------+               +--------+--------+
      |                +---------------+---------------+
      |                V               V               V
      |         +------+------+  +-----+-----+  +------+-----+
      |         | Precompiled |  |           |  |    RP      |
      |         |             |  |           |  |            |
      |         |  Generated  |  |   ES/CS   |  +------+-----+
      |         |             |  |           |  |      |     |
      |         |    COBOL    |  | Procedure |  | Main | Sub |
      |         |             |  |           |  |            |
      |         |  Procedure  |  |           |  | Procedure  |
      |         +------+------+  +-----+-----+  +------+-----+
      |                +---------------+---------------+
      |                                V
      |                        +----------------+
      |                        | COBOL compiler |
      |                        +-------+--------+
      |                +---------------+---------------+
      V                V               V               V
+-----+-------+ +------+------+  +-----+-----+  +------+-----+
|             | | Precompiled |  |   ES/CS   |  |    RP      |
|             | |             |  |           |  |            |
| MYAPPL.OBJ  | |  Generated  |  | Procedure |  +------+-----+
|             | |             |  |           |  |      |     |
|             | |    COBOL    |  |   .OBJ    |  | Main | Sub |
|             | |             |  |           |  |            |
|             | |    .OBJ     |  |           |  |    .OBJ    |
+-----+-------+ +------+------+  +-----+-----+  +------+-----+
      +----------------+------+--------+---------------+
         +---------------+    |   +----------------+
         | User Supplied |    |   | Form Processor |
         |               |    |   |                |
         |   Procedures  |    |   |   Procedures   |
         +------+--------+    |   +-------+--------+
                +-------------+-----------+
                              V
                         *----+---*
                         | Linker |
                         *----+---*
                      +-------+--------+
                      V                V
    +-------+  *------+------*    *----+----*  +----------+
    | F1.FD +->| Application |    |    RP   |<-+ Database |
    |       |  |             |    |         |  |          |
    |       |  |   Program   |    | Program |  |          |
    +-------+  *-------------*    *---------*  +----------+
```

Figure 3-1. Rapid Application Generator Interfaces

3.1.2.2 Detailed Interface Definition

Rapid Application Generator Interfaces

Form Definition Language

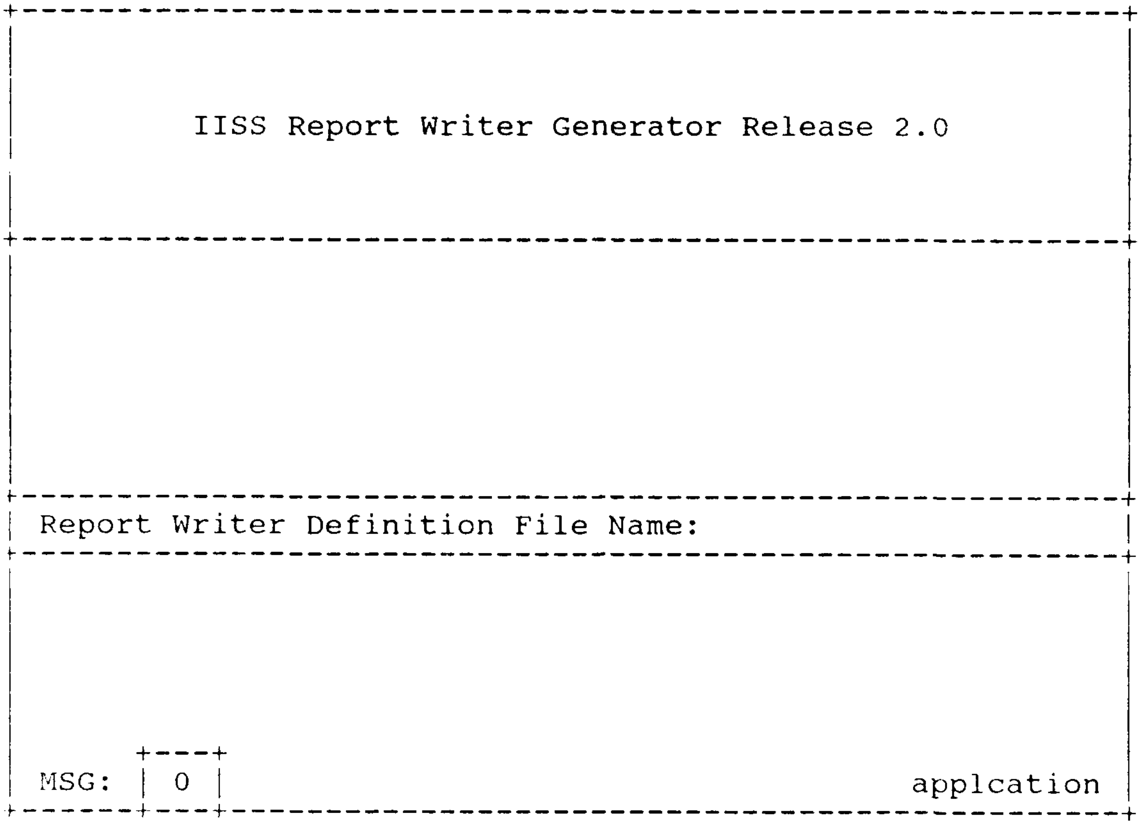
The syntax of the FDL accepted as input by RAP is documented in Appendix A of [4]. This language is intended to provide access to all Form Processor functionality. It is also intended to be a LALR(1) grammar for ease of parsing. A number of automatic parser generators are available for LALR(1) grammars, most notably the UNIX utility YACC.

Binary Form File Format

The format is specified in Section 3 of [4].

User Input to RAP

The following figure is the IISS form RAP uses to prompt the user for an input file.



```
IISS Report Writer Generator Release 2.0

Report Writer Definition File Name:

MSG: | 0 |                                  applcation
```

Figure 3-2. RAP screen

The host system invocation of RAP is system dependent. The user specifies a FDL source file to be compiled, the username and password of the CDM to be accessed. Error messages are directed to the user's terminal.

RAP Error Messages

RAP error messages are specified in Section 3 of [4].

Generated C and COBOL Code

The generated C code contains logic for control flow and Form Processor interactions and interfaces with the generated COBOL code. It is constrained to be syntactically and semantically acceptable to the DIGITAL VAX 11 C compiler.

The generated COBOL code contains NDML statements to access CDM databases and interfaces with the generated C code. It is constrained to be syntactically and semantically acceptable to the NDML Precompiler and the DIGITAL VAX COBOL compiler.

When the C code has been compiled and the COBOL code precompiled and compiled, the modules are linked to create an application program which runs under IISS.

GAPTRAN Interfaces

The invocation of GAPTRAN is system dependent. The user is prompted for the name of the FDL file to be translated and the name of the output file.

3.2 Detailed Functional Requirements

Application dialogues under the Form Processor are based on the forms or block mode of interaction. In this dialogue the user is presented with an electronic form and is requested to supply several data values. Upon entering these values the user presses a function key and the application processes all entered values. The cycle repeats with the application displaying another form for the user. The other major type of application dialogue is conversational, or tty. In this dialogue the application prompts the user for each data value.

Applications consist of two parts: a description of the forms to be displayed on the screen and the control flow to process the data. Forms are specified using the FDL. This language specifies locations and sizes of fields on the screen and is described in Appendix A of [4]. The control flow may be specified by a computer programming language like FORTRAN, COBOL, or C. Applications in these languages interact with the Form Processor by making calls tc procedures which change the forms diaplayed (e.g. addfrm, rmvpag), put or get data from the forms (e.g. pdata, gdata) and output the screen to the user and request input (e.g. outscr, oiscr).

The Application Definition Language (ADL) is a subset of FDL and may be used to specify the control flow of the application. This language is designed to eliminate many of the details normally associated with application specification. It does this by restricting the domain of applications that can be specified by:

o having a uniform model of computation for those applications
o making the application's state information easily accessible
o using forms
o using NDML

The ADL's restricted domain of applications consists of that set of applications which employ relation database access methods and use forms to display and interact with the database. The database access methods are the data manipulation operators of selection, insertion, deletion, and modification. ADL applications assume the data in the database can be used "as is" without performing complex algorithmic manipulations on it.

ADL has an interactive and a noninteractive model of computation. The latter is intended to be used primarily as a report writer. ADL's model of computation is based on the condition and its actions. A condition is syntactically and semantically similar to an if statement in a general purpose language. It tests for a particular computational state and, if true, the condition's actions are executed. This collection of activities is referred to by saying the condition is executed. ADL state information includes such things as function key press and the value of an item on a form. Actions are commands which change the state of computation and include such things as adding a form to the display list and setting an item field to a value.

State information is divided into that of the Form Processor and that of the application. Form Processor state information includes the following items.

- o value of an item
- o scrollable array index
- o display attribute of a field
- o which form is displayed in a window
- o number of pages in a window
- o appears if state
- o cursor position
- o IISS logon role
- o function key pressed
- o modification of an item's value

Application state information includes the following items.

- o application startup point
- o signal flag value
- o data read overflows form
- o data stream value changes
- o empty database query

Actions are commands which alter the state of an application. Possible actions include:

- o output a display
- o terminate the application
- o embedded C code
- o assign a value to an item
- o set a flag
- o a nested condition
- o call a procedure
- o display a help form or message
- o query a database
- o insert a row into a database
- o delete a row from a database
- o modify a row in a database

3.2.1 Language Semantics

The following paragraphs discuss the language semantics required.

3.2.1.1 Create Application Statement

The application_id is the name of the application and is used as the root for the file names of the generated C and COBOL code and the procedure names for the COBOL module.

The parameter_form_id is the name of the UIMS parameter form for this application. This form must also be defined to the User Interface DataBase in order to complete the definition.

The Keypad clause defines a mapping between a terminal's function key numbers and the function key names used in Pick condition expressions. These keypad definitions are in effect for the duration of the application and override those on the Create Form statement. A key_id may is bound to exactly one function key but a function key may be bound to multiple key_ids.

The c_code clause is any C code which might normally occur external to a C procedure. The code is placed into the generated C code after the #include statements and other generated global variable declarations and just before the main procedure.

The Create Application statement specifies that an interactive application is to be generated.

3.2.1.2 Create Report Statement

The application_id, parameter_form_id and c_code have identical semantics to the Create Application statement.

The Create Report statement specifies that a noninteractive application is to be generated.

3.2.1.3 Condition Statement

A condition is syntactically and semantically similar to an if then else statement in a general purpose language. The condition's boolean expression facilitates the checking of Form Processor and application state information. When the condition's expression is tested the condition is said to be evaluated. If the evaluation is true the condition's actions are executed in the order they appear in the source file. When the condition's actions are executed the condition is said to be executed.

In the interactive model, condition's that have expressions which refer to Form Processor state information are evaluated once per function key press, in the order they are listed in the source file and at most one is executed. In either model, conditions which refer to application state information are evaluated during the appropriate action and if true, the condition is executed and flow of control continues with the first action after the one that tripped the condition. In the noninteractive model, all conditions should refer to one of the application state information conditions: Startup, Overflow, Change, Empty, or flag_id. This is because there are no function key presses and the application is entirely driven by the data read from a database. When a condition is tripped the application state information is set to false prior to executing the condition.

Conditions which refer to the Startup condition are evaluated once at application startup. Conditions which refer to the Overflow or Change conditions are evaluated during an appropriate Present action. If tripped, the Present action terminates and the condition is executed. When the condition terminates the flow of control continues with the first action after the Present. A condition which refers to the Empty condition is evaluated following the corresponding Select action. If tripped the condition is executed. When the condition terminates the flow of control continues with the first action after the Select. A condition which refers to a flag_id is evaluated following the corresponding Signal action. If tripped, the condition is executed. When the condition terminates the flow of control continues with the first action after the Signal.

Condition Expressions

The semantics of expressions are similar to that of Value and Appears If clauses in the Form Processor. This section describes the application state conditions.

The Pick condition is true if the specified function key has been pressed. If the key was defined on a form, it also checks that the form the key was defined on is displayed. The Startup condition is true when the application starts. The Overflow condition is true if the number of rows selected from a database exceeds the number of array elements allotted on the target form. The Change condition is true if a column's value changes in the sequence of rows selected from a database. The Overflow and Change conditions occur during the Present action. The Empty condition is true when a Select action returns no rows. A flag_id is true if it has been set by the Signal action.

Form Keypad Definitions

Form Keypad definitions are part of the Create Form statement like Prompts and Background clauses. The Keypad clause defines a mapping between a terminal's function keys, represented by nonnegative integers and the function key names used in the Pick condition. Keypad definitions are placed on forms which are not subforms. The keypad bindings are in effect while the form is the top page in the window. Keypad definitions on the Create Form statement are overridded by those on the Create Application statement. A key_id is bound to exactly one function key but a function key may be bound to more than one key_id.

Qualified Names

Qualified names are used to refer to Form Processor fields. They are enclosed in single quotes. Qualified names may refer only to those fields contained in the source file. Forms referenced in qualified names must also be presented at some point in the application.

Two symbolic array indices are available. The use of a symbolic index implies a control flow structure loop and at application run time the symbolic index is replaced with an actual value. The part of the qualified name which prefixes a symbolic index has scope from that condition or action to all nested conditions and actions inclusive. Scope rules are based on an upper case comparison of the symbolic index prefix. The syntax for the universal (for all) quantifier index is a star. This means all elements of the array will be used sequentially in the condition or action. The for each quantifier index is a zero. This symbolic index is normally used in a qualified name which is the target of a Select action, or a condition or action which are nested in the Select.

3.2.1.4 Page Statement

The Page action outputs the current display list to the current logical device.

3.2.1.5 Exit Statement

The Exit action terminates the application.

3.2.1.6 Embedded C Code

This allows the user to reference the generated internal data structures and perform specialized computations. Only qualified name internal data structures should be referenced to ensure the integrity of the application.

3.2.1.7 Set Statement

The Set action assigns the value of the right hand side to the item field on the left.

3.2.1.8 Signal Statement

The Signal action allows the user to set or reset the state of a flag_id. Any top level conditions which refer to the flag_id will be evaluated.

3.2.1.9 Nested Condition Statement

Nested conditions differ from top level conditions in that they are evaluated if their parent condition or action is executed. They are evaluated in the order they appear in the source file. References to application state functions, Startup, Overflow, Change, Empty and to flag_id does not change the order of evaluation.

3.2.1.10 Help Statement

The Help action is used to display a message in the message line or display a help form.

3.2.1.11 Present Statement

The purpose of the Present action is to replace the current or top form in a window. If there is no form in the window the form is added. A form may be Presented in exactly one window.

The purpose of the Display action is to add a form to the top of a window. A form may be Displayed in exactly one window.

The purpose of the Redisplay action is to remove forms from a window until the specified form is the current form in the window. A form may be Redisplayed in exactly one window.

A secondary purpose of a Present action is to initiate or resume reading data from selects which target to the presented form or one of its subforms. Data reading may be supressed by using the Noselect option.

3.2.1.12 The Call Statement

The purpose of the call action is to allow the application to make calls to Form Processor procedures or other system services to perform specialized computations. The call action invokes the procedure procedure_id. The arguments are passed by reference. Arguments may be a string, integer, real or a qualified name of an item. Items default to data type character. Items may be coerced to an integer or real data type by using the type modifier Integer or Real. The character string representing the qualified name of the item itself will be passed to the procedure if the type modifier Path is used.

3.2.1.13 NDML Statements

NDML statements are similar to those of the CDM precompiler with provisions for referencing item fields. Item fields replace references to local variables. Error checking is limited to syntax and the existence of referenced tables and columns.

The Select Statement

The semantics of the Select action are similar to those of NDML selects. This section will describe those aspects which differ. Select describes a mapping from tables and columns to item fields. The From clause specifies which tables are involved. The Where clause specifies which rows are to be retrieved. Operands may be columns, strings, integers, real numbers or item fields. The Order By clause specifies the order in which the rows are to be retrieved.

The Select action establishes which data are to be read. It does not read the data nor are subactions executed until the data are read during a Present action. If reading is suspended due to the occurrence of an Overflow or Change condition another Present action will continue reading the data.

The Nodup Clause

The Nodup clause is part of an Item field on a form, like the size and location clauses. The item with the Nodup clause is the target of a Select action. In a sequence of rows selected from a database if the value of a column in one row is the same the previous row's, the item's value will be blanks. The use of the Nodup clause assumes the rows have been ordered by the column which targets the item.

The Picture Clause

The Picture clause is a part of the Domain clause of an Item field on a form, like the Left and Must Enter clauses. The item with the Picture clause is the target of a Select action.

The string is a COBOL PICTURE clause which describes how the data from the selected column is to be formatted. The Picture string must describe a data type which is character based and compatible with the External Schema data type of the source column. The Picture string may not contain a repeat specification.

The Insert Statement

The semantics of the Insert action are similar to those of NDML inserts. The Insert describes a mapping of values to a table and its columns.

The Modify Statement

The semantics of the Modify action are similar to those of NDML inserts. The Modify describes a mapping of values to a table and its columns.

The Delete Statement

The semantics of the Delete action are similar to those of NDML inserts. The Delete describes which rows in a table are to be removed.

3.2.2 Generate Code Specification

Unless otherwise specified the generated code is in C.

3.2.2.1 Create Application Statement

The application_id is used as the root for the file names of the generated C and COBOL code and the procedure names for the COBOL module. Under the VMS operating system the C file will have the file extension .C and the COBOL code .PRC.

The c_code clause is placed into the generated C code after the #include statements and other generated global variable declarations and just before the main procedure.

The Create Application statement specifies that an interactive application is to be generated. The interactive model begins by executing a condition with a Startup function. This condition's actions should display the first form and possibly assign values to some item fields or select some data from a database. Then a loop is started which contains a call to the Form Processor procedure OISCR to output and input the screen. Next, within the loop are a number of conditions. These conditions sequentially test the state information until a match is found and then the condition is executed. At most one condition is executed per iteration. The loop repeats until an Exit action is encountered at which time the application terminates.

3.2.2.2 Create Report Statement

The c_code has identical semantics to the Create Application statement.

The Create Report statement specifies that a noninteractive application is to be generated. The noninteractive model begins with a condition which contains the Startup function. The initial actions will include a Select action to establish which data are to be read from the database and a Present action to display a form and start reading from the database. Since there is no user interaction with the application, conditions are checked in a different manner than in the interactive model. While data are being read, conditions are checked which determine if one of the application states has occured (such as Overflow, Change and Empty). When these conditions are executed they typically display the screen (using the Page action) and start reading another screenful of data using another Present action. When all actions have finished executing and no more conditions are true then the application terminates.

3.2.2.3 Condition Statement

Each condition is represented in the generated C code by a procedure of the name condXXX where XXX is a number. The Form Processor procedure EVLINT is used to evaluate the condition's expression. All Form Processor state information can be handled directly by this procedure. Application state information is contained in the boolean array func[XXX] where XXX is a number. The current truth value is inserted in the condition expression prior to the call to EVLINT. If the condition is tripped the func[XXX] truth value is reset immediately after the call and before the actions are executed.

Each action is represented in the generated C code. These actions are contained inline with the condXXX procedure. Actions are generated in the order they appear in the source file.

Condition Expressions

In the generated code the expression is placed in a character string. At application runtime the values representing application state information are inserted in the string using the C function sprintf. The result is evaluated using the Form Processor procedure EVLINT.

Form Keypad Definitions

In the generated code the variable fldno[XXX] keeps track of which forms are currently the top page in their windows. This variable is modified when a Present action occurs.

Qualified Names

At compilation time, RAG constructs a psuedo display list to verify that qualified names will exist at run time. In the generated code for the symbolic indices, the variable indx[XXX], where XXX is a number contains the actual index value. The variable inmx[XXX] contains the maximum number of elements in the array. This maxinum number is determined by calling the Form Processor procedure NUMELM. A C for loop contains all conditions and actions which are in the scope of the qualified name using the symbolic array index.

At application runtime the index values are substituted into qualified names containing the symbolic array index using the C procedure sprintf. In general all the qualified names in an action must refer to fields which are on the display list of the current logical device. The exception to this are qualified names in the Select part of a Select action. These qualified names must be on the display list when the form they are on is Presented. These qualified names may also be in the Where clause of nested Selects.

Data structures which hold values for item fields are named: XXX1.XXX2...XXXn. Where XXX1 is the name of a Presented form or a form which is an open ended array. XXXi is further qualification of the C data structure. The internal data structures do not necessarily have the current field values as they are updated following a Set or Present action only.

3.2.2.4 Page Statement

In the generated code the _PAGENO item field on the form PSCREN is incremented with the Form Processor procedure EVLINT and the screen is output by calling OUTSCR.

3.2.2.5 Exit Statement

In the generated code the Form Processor procedure TERMFP is called, and the NTM service TRMNAT or the CDM procedure TRMDML if there are NDML statements.

3.2.2.6 Embedded C Code

In the generated code the C source code between the '%{' and '%}' is copied as an action. The C code may be any statements that would normally appear inside the control flow statements of a procedure. The embedded C code is not checked for syntax or semantics. The #line preprocessor statement is generated so C compile errors refer to lines in the ADL source file.

3.2.2.7 Set Statement

In the generated code the right hand side is evaluated by calling the Form Processor procedure EVLSTR. The internal data structure which represents the target item field is updated. If the qualified name references arrays, the internal data structure element indexed by nonsymbolic zero is updated.

3.2.2.8 Signal Statement

In the generated code the flag_id is set to true (or false if the Not option is specified). The flag_id is internally represented by the boolean array func[XXX] and its value is set according to the Not option. The XXX is an index to the func array for the flag_id. The conditions are evaluated by calling the condXXX procedure generated for conditions.

3.2.2.9 Nested Condition Statement

In the generated code nested conditions are generated inline with their parent condition's condXXX procedure. The condition's expression is evaluated using the Form Processor function EVLINT. If the expression is true, references to any application state information causes the corresponding flag to be reset and then the actions are executed.

3.2.2.10 Help Statement

In the generated code the help string format uses the Form Processor function PMSGLS to display the string in the message line. The help form_id format uses the Form Processor function ADDFRM to add the form_id to the window SCREEN on the form PSCREN. Next, is a call to OISCR and when the user presses an application function key, a call is made to RMVPAG to remove form_id from the window.

3.2.2.11 Present Statement

In the generated code the Present action calls the Form Processor procedure RMVPAG to remove the top page in the window window_name (or SCREEN if unspecified) and then ADDFRM to put the form form_id in the window.

In the generated code the Display action calls the Form Processor procedure ADDFRM to put the form form_id in the window window_name (or SCREEN if unpsecified).

In the generated code the Redisplay action repeatedly calls GPAGE to determine what pages, if any, must be removed from the window window_name (or SCREEN if unspecified) so the form form_id will be the current form in the window. If pages are to be removed then RMVPAG is called to remove them. If the form form_id is not found then all pages are removed from the window.

In the generated code if there is a select which targets to an item on a form, or one of its subforms, a call is made to the procedure xXXX, where XXX is the name of the presented form. This procedure calls a selXXX procedure, where XXX is the number of the select action. The selXXX procedure is reponsible for reading the data and is specified in section 3.2.2.13. To determine which was the last select to map to this form the selXXX procedure is called through the psno[XXX] variable. When the form is displayed and the xXXX procedure returns the internal data structure representing the form is updated by calling the Form Processor procedure GDATA. The Noselect option suppresses the generation of the call to xXXX. The variable fldno[XXX] is updated to indicate the form is currently the top page in its window and so updates the form/function key bindings.

3.2.2.12 The Call Statement

In the generated code the Form Processor procedure GDATA is used to obtain the values of the items. The procedure PDATA is used to place any modified values back into the items. Integer or real variables are generated for literal integers or reals or for items which are coerced to those types.

3.2.2.13 NDML Statements

NDML statements are similar to those of the CDM precompiler with provisions for referencing item fields. In the generated C code each NDML statement is represented by a procedure named sqlXXX, where XXX is a number. The C code is responsible for retrieving data from the forms and passing it to the COBOL code.

In the generated COBOL code each NDML statement is represented by a procedure named application_idXXX, where XXX is a number. The COBOL code contains the NDML statement in a form to be precompiled by the CDM precompiler.

The Select Statement

In the generated code the C procedures sqlXXX, selXXX, getdbXXX and putdbXXX and the COBOL procedure application_idXXX are generated. At the time the application is run sqlXXX and application_idXXX create a temporary file, using putdbXXX, with the results of the select statement. selXXX reads the data from the file using getdbXXX and moves it to the item fields on the form.

In sqlXXX, the Form Processor procedure GDATA is used to obtain the values of item fields, if any, referenced in the where clause. If the select has nested selects then the values of item fields used in their where clauses are also obtained. These values are stored in a data structure named "whrstruct.whrXXX.YYY", where XXX is the number of the select and YYY is the name of the item field. Next sqlXXX calls the precompiled module application_idXXX. The select is performed and for each row retrieved putdbXXX is called. The results are stored in a temporary file. sqlXXX also checks for the Empty condition function, sets the flag func[YYY] and calls the condition condZZZ which implements this function.

In selXXX each row is read using getdbXXX. Each column retrieved is in a data structure named dbrXXX.YYY, where XXX is the number of the select and YYY is the name of the target item field. The data are placed in the item fields by calling PDATA. selXXX also checks for Overflow and Change condition functions, sets the flag func[YYY] and calls the condition condZZZ which implement these functions. If there are subactions they are generated inline to the procedure selXXX and are executed for each row read.

The Nodup Clause

In the generated code for each Nodup clause, a variable named chgXXX is generated, where XXX is a number. In the procedure sqlXXX the variable is initialized to blanks. In the procedure selXXX the variable is checked against the current value read and if different the value is put in the item field as normal. Next the variable is updated with the current value.

The Picture Clause

In the generated COBOL code the Picture string is used as the declaration for the variable with receives the selected External Schema column value.

The Insert Statement

In the generated code an Insert action is represented by the procedure sqlXXX, where XXX is the number of the insert. The generated COBOL procedure is application_idXXX. The procedure sqlXXX obtains the values of any item fields referenced by calling the Form Processor procedure GDATA and places the value in the data structure dbrXXX.YYY, where XXX is the number of the insert and YYY is the name of the item field. Next, the procedure application_idXXX is invoked.

The Modify Statement

In the generated code the Modify action is represented by the procedure sqlXXX, where XXX is the number of the modify. The generated COBOL procedure is application_idXXX. The procedure sqlXXX obtains the values of any item fields referenced in the

Set clause by calling the Form Processor procedure GDATA and placing the value in the data structure dbrXXX.YYY, where XXX is the number of the insert and YYY is the name of the item field. Any item fields referenced in the Where clause are obtained using GDATA and are placed in the data structure whrstruct.whrXXX.YYY. Next, the procedure application_idXXX is invoked.

The Delete Statement

In the generated code the Delete action is represented by the procedure sqlXXX, where XXX is the number of the delete. The generated COBOL procedure is application_idXXX. The procedure sqlXXX obtains the values of any item fields referenced in the Where clause by calling the Form Processor procedure GDATA and placing the value in the data structure whrstruct.whrXXX.YYY. Next, the procedure application_idXXX is invoked.

3.2.3 Data Structures

Figures 3-3a and 3-3b show the major data structures created by RAP in generating the application. The data structure FIELD is created by modules specific to the FLAN CPCI and are illustrated in [4]. Pointers not pointing to anything point to another instance of the data structure.

```
   ACTLST                    PROCLST
  +---------+               +---------+
  | nxtact  +->             | nxtpro  +-
  +---------+               +---------+
  | acttyp  |               | string  |
  +---------+               +---------+
  | selptr  +-> SELECT      | integer |
  +---------+               +---------+
  | insptr  +-> INSERT      | double  |
  +---------+               +---------+
  | modptr  +-> MODIFY      | fld     +-
  +---------+               +---------+
  | delptr  +-> DELETE      | type    |
  +---------+               +---------+
  | proclst +-> PROCLST
  +---------+
  SELECT                    INSERT
+--------+                +--------+
| allsel +-->             | allins +->
+--------+                +--------+
| collst +-->  COLLST     | tbllst +->  TBLLST
+--------+                +--------+
| tbllst +-->  TBLLST     | collst +->  COLLST
+--------+                +--------+  +--------+
| selexp +-->  SELEXP                 | nxtcol +->
+--------+                            +--------+
| whrexp +-->  WHREXP                 | esdtyp +->  ESDTYP
+--------+                            +--------+
| ordlst +-->  ORDLST                 | pred   +->  PREDOPND
+--------+   +--------+               +--------+ +--------+
             | nxtord +-->                       | fld    +->
             +--------+                          +--------+
             | fld    +-->  FIELD                | str    |
             +--------+                          +--------+
             | esdtyp +-->  ESDTYP               | num    |
             +--------+   +--------+             +--------+
                          | nxtesd +-->          | esdtyp +->
                          +--------+             +--------+
                          | colnam |
                          +--------+
                          | type   |
                          +--------+
```

Figure 3-3a. Major Data Structures Created By RAP

```
  MODIFY                    DELETE
+---------+               +---------+
| allmod +-->             | alldel +-->
+---------+               +---------+
| tbllst +-->   TBLLST    | tbllst +-->             TBLLST
+---------+               +---------+           +---------+
| collst +-->   COLLST    | whrexp +-->   WHREXP | nxttbl +-->
+---------+               +---------+           +---------+
| whrexp +-->   WHREXP                          | tblnam  |
+---------+  +---------+                        +---------+
             | whrexp +-->                      | esdtyp +-->
             +---------+                        +---------+
             | pred   +-->  PREDOPND
             +---------+
             | type   +-->
             +---------+
    SELEXP
  +---------+
  | selexp +-->
  +---------+
  | select +-->
  +---------+
  | type    |
  +---------+
```

Figure 3-3b. Major Data Structures Created by RAP (Continued)

3.3 Performance Requirements

The following paragraphs discuss the RAP performance requirements.

3.3.1 Programming Methods

The RAP is written in C and NDML. A number of existing Form Processor and FLAN modules are used as the RAP shares some internal data structures with them. Generated applications are compiled using the C compiler, the NDML Precompiler, the COBOL compiler and then linked to create executable application.

3.3.2 Program Organization

RAP uses the callable interface to FLAN to parse the source input and create the data structures. The NDML module is used to verify the existence of tables and columns referenced in the NDML statements. Separate modules are used to generate the application's data structures, the control flow for interactive/noninteractive applications, the conditions and actions, and the NDML statements.

3.3.3 Modification Consideration

The use of existing Form Processor and FLAN modules ensures that changes made to FP data structures and procedures will require few changes be made to RAP's procedures.

3.4 Human Performance

Not applicable.

3.5 Data Base Requirements

The RAP accesses the CDM data dictionary information to verify the existence of tables and columns referenced in NDML statements and to generate application code for the External Schemas.

3.5.1 Sources and Types of Input

The RAP accesses the CDM user view: VIEW_META_DATA. It references the columns: DI_ID, ES_TYPE, ES_SIZE, and ES_ND.

3.5.2 Destinations and Types of Output

The generated code is returned to the External Schemas. COBOL, C, and FORTRAN codes are generated.

3.5.3 Internal Tables and Parameters

RAP shares internal data structures with the Form Processor and FLAN modules.

3.6 Adaptation Requirements

The following paragraphs define the program adaption requirements.

Portability

In order to ensure generated code portability, the generated C and COBOL code must satisfy the following constraints:

- The C code will include the ANSI C header <string.h>.
- The C code will not reference freed storage.
- The COBOL code will use only double quotes to specify character strings.
- The COBOL code will not contain tabs.
- The COBOL instruction EXIT PROGRAM will be within a separate paragraph.

In addition to these specific constraints, any future modifications to the code generator should ensure that the generated C code adheres to the current version of the IISS C Coding Standards.

Code Identification

The generated C code contains a comment line at the beginning identifying the name of the generated COBOL program. The format of this comment is:

```
THIS FILE IS ASSOCIATED WITH filename.PRC
```

if generated on a VAX/VMS host and

```
THIS FILE IS ASSOCIATED WITH PRC(member)
```

if generated on an IBM host.

3.7 Government-Furnished Property List

Not applicable.

SECTION 4

QUALITY ASSURANCE PROVISIONS

4.1 Introduction and Definition

"Testing" is a systematic process that may be preplanned and explicitly stated. Test techniques and procedures may be defined in advance and a sequence of test steps may be specified. "Debugging" is the process of isolation and correction of the cause cf an error.

"Antibugging" is defined as the philosophy of writing programs in such a way as to make bugs less likely to occur and when they do occur, to make them more noticeable to the programmer and the user. In other words, as much error checking as is practical and possible in each routine should be performed.

4.2 Computer Programing Test and Evaluation

The quality assurance provisions for test consist of the normal testing techniques that are accomplished during the construction process. These tests include design and code walk_throughs, unit testing, and integration testing.

4.2.1 Preliminary Qualification Tests

Preliminary qualification tests consist of design and code walk-throughs. These tests are performed by the design team. Structured design, design walk-through and the incorporation of "antibugging" facilitate this testing by exposing and addressing problem areas before they become coded "bugs".

Development and testing are done on the Air Force VAX.

4.2.2 Formal Qualification Tests

The first formal tests involve unit testing. These tests also are performed by the design team. These tests verify the functionality of each RAP routine. Each function is tested separately, then the entire subsystem is tested as a unit.

4.2.3 System Test Program

The integration testing entails generating from an ADL definition an application comprising C and COBOL code, compiling the C portion, precompiling the COBOL portion with the CDM Precompiler, compiling the resultant COBOL program, installing the application on the IISS UIS, and producing an interactive database application using a CDM data base via the NTM.

4.3 Verification Cross-Reference Matrix

A cross-reference matrix, or similar process, to assure all dependencies of each routine upon the other have been verified will be established and included in the unit test plan.

SECTION 5

PREPARATION FOR DELIVERY

The implementation site for the constructed software is the ICAM Integrated Information Support System (IISS) Test Bed site located at Arizona State University, Tempe, Arizona. The software associated with each CPCI release is delivered on a media which is compatible with the IISS Test Bed. The release is clearly identified and includes instructions on procedures to be followed for installation of the release. Integration with the other IISS CPCI's is done at the IISS Test Bed on a scheduled basis.

SECTION 6

NOTES

Please refer to the Software Availability Bulletin, Volume III, Part 16, CI# SAB620326000, for current IISS software and documentation availability.