

WRDC-TR-90-8007
Volume VIII
Part 25

AD-A248 932



INTEGRATED INFORMATION SUPPORT SYSTEM (IISS)
Volume VIII - User Interface Subsystem
Part 25 - Application Generator User's Manual

S. Barker

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

DTIC
ELECTE
APR 23 1992
S D D

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

92-10424



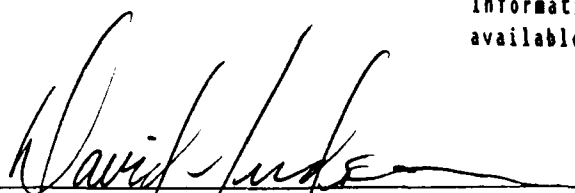
92 4 22 109

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.

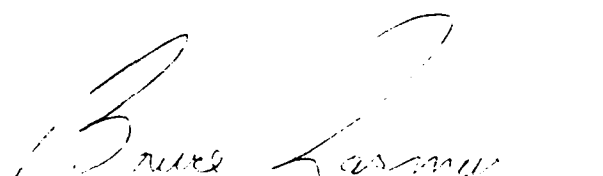
This technical report has been reviewed and is approved for publication.

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

25 July 91
DATE

FOR THE COMMANDER:


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 from 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.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution is Unlimited.	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UM 620344501		5. MONITORING ORGANIZATION REPORT NUMBER(S) WRDC-TR- 90-8007 Vol. VIII, Part 25	
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.	
11. TITLE See block 19		PROGRAM ELEMENT NO. 78011F	PROJECT NO. 595600
		TASK NO. F95600	WORK UNIT NO. 20950607
12. PERSONAL AUTHOR(S) Structural Dynamics Research Corporation: Barker, S., et al.			
13a. TYPE OF REPORT Final Report	13b. TIME COVERED 4 / 1 / 87 - 12 / 31 / 90	14. DATE OF REPORT (Yr., Mo., Day) 1990 September 30	15. PAGE COUNT 68
16. SUPPLEMENTARY NOTES WRDC/MTI Project Priority 6203			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify block no.)	
FIELD	GROUP	SUB GR.	
1308	0905		
19. ABSTRACT (Continue on reverse if necessary and identify block number) This document describes the syntax of the Application Definition Language (ADL). BLOCK 11: INTEGRATED INFORMATION SUPPORT SYSTEM Vol VIII - User Interface Subsystem Part 25 - Application Generator User's Manual			
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	

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

	<u>Page</u>
SECTION 1. INTRODUCTION	1-1
1.1 Manual Objectives	1-1
1.2 Intended Audience	1-1
1.3 Related Documents	1-1
SECTION 2. DOCUMENTS	2-1
2.1 Reference Documents	2-1
2.2 Terms and Abbreviations	2-2
SECTION 3. APPLICATION DEFINITION LANGUAGE (ADL)	3-1
3.1 ADL Format Notation	3-1
3.2 Application Definition Language	3-2
3.3 Application Definition Language Syntax	3-3
3.3.1 CREATE APPLICATION Statement	3-6
3.3.2 Parameter_form_id	3-8
3.3.3 KEYPAD Clause	3-8
3.3.4 Embedded C Code	3-9
3.3.5 Condition Definition	3-9
3.3.5.1 Expression Parameter	3-10
3.3.5.2 Action Parameter	3-15
3.3.5.3 Value Parameter	3-20
3.3.5.4 SELECT Statement	3-21
3.3.5.4.1 Statistic Functions	3-26
3.3.5.4.2 Col_Spec Parameter	3-27
3.3.5.4.3 Query Expression	3-28
3.3.5.4.4 Where Expression	3-31
3.4 Qualified Names	3-35
SECTION 4. HOW TO CREATE AN APPLICATION DEFINITION	4-1
4.1 Statement Format	4-1
4.2 Restrictions	4-1
4.3 Abbreviations	4-1
4.4 Including Comments	4-1
4.5 Reserved Words	4-1
4.6 How to Define a Report Application	4-2
4.6.1 A Simple One Page Report	4-3
4.6.1.1 Specifying the Report Format	4-3
4.6.1.2 Retrieving the Database Information	4-5
4.6.2 The Multi-Page Report	4-5
4.6.2.1 Specifying the Multi-Page Report Format.	4-7
4.6.2.2 Paging the Report	4-8
4.6.2.3 Grouping Database Information	4-9
4.6.2.4 Using Nested SELECT Commands	4-9
SECTION 5. HIERARCHICAL REPORT WRITER	5-1
5.1 Defining a Hierarchical Report	5-2
5.2 Accessing the Hierarchical Report Writer ...	5-6
APPENDIX A. Steps For Executing The Rapid Application Generator	A-1

LIST OF ILLUSTRATIONS

<u>Figure</u>		<u>Page</u>
4-1	Report Format	4-3
4-2	Structure of EMPDATA Table	4-3
4-3	Form Hierarchy for the Payroll Report	4-4
4-4	New Table Structure for EMPDATA	4-6
4-5	Format of Payroll Report from Combined Tables	4-6
4-6	Form Hierarchy for the Multi-Page Report	4-7
4-7	Table Structures for DEPTDATA and EMPDATA	4-10
5-1	Format of a Hierarchical Report	5-1
5-2	Format of an Assembly Hierarchy	5-2
5-3	Format of a Printed Hierarchical Report	5-3
5-4	Empdata Table	5-3
5-5	Format of the Initial Report	5-5

Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



SECTION 1

INTRODUCTION

The Rapid Application Generator (RAP) is a tool for translating a textual definition of an interactive database application or a report application into the C and COBOL programs that are required to access selected data base information resident in the Common Data Model (CDM). These data are accessible through the Integrated Information Support System (IISS) Neutral Data Manipulation Language (NDML). To generate an application, you must first use the Application Definition Language (ADL) to define the application. You then go through several steps defined in Appendix A to generate the executable application program from the application definition.

1.1 Manual Objectives

The main objective of this manual is to describe the syntax of the ADL. Topics covered include:

- o How to define an application.
- o The syntax of the language.
- o How to run the application generator and produce the executable.

1.2 Intended Audience

The RAP is intended for use by application programmers in the IISS environment. Knowledge of User Interface forms, the CDM, and the Neutral Data Manipulation Language (NDML) is assumed.

1.3 Related Documents

The Form Editor User Manual describes how to define the User Interface forms. Information you should be familiar with includes:

- o item, form and window fields
- o repeating fields

The Form Processor User Manual explains several concepts you need to help you in defining and using forms for application software. These include:

- o form hierarchies
- o qualified names

The Neutral Data Manipulation Language User Manual describes the syntax of the NDML. This language provides the capability of communicating with the IISS testbed databases. You need to be familiar with the SELECT, DELETE, INSERT, and MODIFY commands of the NDML.

SECTION 2

DOCUMENTS

2.1 Reference Documents

- [1] Structural Dynamics Research Corporation, IISS Application Generator Development Specification, DS 620344502, 31 May 1988.
- [2] Systran, ICAM Documentation Standards, 15 September 1983, IDS150120000C.
- [3] Systran Corporation, Embedded NDML Programmer's Reference Manual, PRM620341001, 31 May 1988.
- [4] Systran, User's Manual for the ICAM Integrated Support System (IISS) Neutral Data Manipulation Language (NDML), February, 1983.
- [5] Systran, Implementation of Enhancements of NDML SELECT COMMAND, 25 July 1984, revised 9 September 1984.
- [6] Systran, Discussion of Function Implementation NDML SELECT COMMAND, 25 July 1984, revised 4 September 1984.

This manual is one of a set of user manuals that together describe how to operate in the IISS environment. The complete set consists of the following manuals listed here for reference:

- [1] Structural Dynamics Research Corporation, IISS Form Editor User Manual, UM 620344400, 31 May 1988.

Explains how to define and maintain electronic forms. It is intended to be used by programmers writing application programs that use the Form Processor.
- [2] Structural Dynamics Research Corporation, IISS Form Processor User Manual, UM 620344200, 31 May 1988.

Describes the set of callable execution time routines available to an application program to process electronic forms. It is intended to be used by programmers writing application programs for the IISS environment.
- [3] Structural Dynamics Research Corporation, IISS Terminal Operator Guide, OM 620344000, 31 May 1988.

Explains how to operate the generic IISS terminal when running an IISS application program. The IISS end user environment, function selection and some predefined applications are also described.

- [4] Structural Dynamics Research Corporation, IISS Text Editor User Manual, UM 620344600, 31 May 1988.

Explains how to use the file editing functions including: inserting, deleting, moving and replacing text.

- [5] Structural Dynamics Research Corporation, IISS Application Generator User Manual, UM 620344502, 31 May 1988.

Describes the Application Definition Language and the process used for translating textual definitions of interactive database applications into programs that access selected data base information resident in the Common Data Model. This information is accessible through the IISS Neutral Data Manipulation Language.

- [6] Structural Dynamics Research Corporation, IISS Virtual Terminal User Manual, UM 620344300, 31 May 1988.

Explains the program callable interface to the IISS Virtual Terminal. The callable routines, Virtual Terminal commands and the implementation of additional terminals are described. It is intended for application and system programmers working in the IISS environment.

2.2 Terms and Abbreviations

Abbreviation id: A user-defined abbreviation of a database table that may be used as a qualifier for a data item instead of the table name itself.

Application Definition Language (ADL): A subset of the Forms Definition Language (FDL) which provides for the definition of interactive, form-based applications. ADL is part of FDL. The statements are included in the same source file as the form definitions.

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

Application Generator (AG): Subset of the IISS User Interface that consists of software modules that generate IISS application code and associated form definitions based on a language input. The part of the AG that generates report programs is called the Report Writer. The part of the AG that generates interactive applications is called the Rapid Application Generator.

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.

Attribute: A field characteristic such as blinking, highlighted, black, etc. and various other combinations. Background attributes are defined for forms and window fields only. Foreground attributes are defined for item fields. Attributes may be permanent (they remain the same unless changed by the application program), or they may be temporary (they remain in effect from the application's first call to OISCR after a call to PUTATT or PUTBAK to the next call to OISCR).

Background Attribute: A color which is applied to the background of forms, graphs, window fields, and item fields. Specifying a background attribute is analogous to specifying what color of paper to print a form, etc. on.

Bar Graph: A graph which represents dependent data values correlated to an independent variable by horizontal or vertical rectangles which are proportional to the dependent data values.

C Code: A user-defined sequence of C language statements which are to be included in the generated application.

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.

Communication Services: Allows on host interprocess communication and inter-host communication between the various Test Bed subsystems.

Communication Subsystem (COMM): IISS subsystem that provides communication services to the Test Bed and subsystems.

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 modeling.

Condition Action: An action to be taken when a specified condition is true.

Condition Definition: Specifies pre-defined actions that will occur as the result of user interaction with the software via the terminal.

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

Dependent Data: Data correlated to a dependent variable.

Dependent Variable: A mathematical variable whose value is determined by that of one or more other variables in a function.

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 and the user.

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

Field: An area on a form which is a place-holder for data. A field may be an item, a form, a window, or a graph.

Field Name: A qualified name enclosed in single quotes which denotes a field in the Form Processor display list.

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

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

Foreground Attribute: A characteristic which is applied to the text of a form, graph, or item. Specifying a background attribute is similar to specifying what color of ink to use to print a form on paper. Foreground attributes may also include such characteristics as blinking, highlighted, etc.

Form: A 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, windows, and graphs.

Form Definition (FD): Form Definition Language after compilation. It is read at run-time by the Form Processor.

Form Definition Language (FDL): The language in which electronic forms and the reports and applications which interact with them are defined.

Form Driven Form Editor (FD FE): Subset of the FE which consists of a forms driven application used to create Form Definition files interactively.

Form Editor (FE): Subset of the IISS User Interface that is used to create definitions of forms. The FE consists of the Forms Driven Form Editor and the Forms Language Compiler.

Form Field: A place-holder on a form for a sub-form which is to be supplied in the form definition. A sub-form should contain a group of logically related data. A form field is

static, that is, each time the form which contains the form field appears, the same sub-form will be displayed in the form field.

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

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

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.

Form Processor Text Editor (FPTE): Subset of the Form Processor that consists of software modules that provide text editing capabilities to all users of applications that use the Form Processor.

Forms Language Compiler (FLAN): A translator that accepts a series of Forms Definition Language statements and produces form definition files as output.

Graph: A picture correlated with data that alters as the data changes; by necessity, this is a dynamic (not pre-defined) picture. A graph may be imposed on windows or forms.

Graph Field: An area on a form into which a pre-defined graph will be placed at run-time.

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.

Independent Data: Data that is correlated to an independent variable.

Independent Variable: A mathematical variable whose value is specified first and determines the value of one or more other values in an expression or function. For example, In a business graph of sales versus month, month is the independent variable and sales is the dependent variable, because sales varies by month.

Item Field: A non-decomposable area of a form in which 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.

Line Graph: A graph which represents dependent data values correlated to an independent variable by points connected with a broken line.

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. NOTE: a single application can have more than one logical device. To the end user this also appears as multiple applications running simultaneously.

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 have been and are currently open for an application process.

Operand: A value which is operated on by some function.

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.

Operator: A mathematical or logical symbol denoting an operation to be performed (e.g., +, -, /, *, etc.).

Page: An instance of a form in a window that is 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 at one time with provisions for viewing any portion of the data buffer.

Physical Device: A hardware terminal.

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

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

Pie Graph: A graph which represents data values on a circle which is cut into sections by radii. Each section of the circle represents a data value as a percentage of the total of all the values represented by the graph.

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

Primitive: The lowest level of definition for an attribute. Primitives define the characteristics of an attribute.

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

Rapid Application Generator (RAP): A translator which accepts ADL as input and generates an interactive application as output.

Report Writer (RW): A translator which accepts ADL as input and generates a report application as output.

Sub-form: A form that is used within another form.

Text Editor (TE): Subset of the IISS User Interface that consists of a file editor that is based on the text editing functions built into the Form Processor.

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 Development System (UIDS): Collection of IISS User Interface subsystems that are used by applications programmers as they develop IISS applications. The UIDS includes the Form Editor and the Application Generator.

User Interface Management System (UIMS): The run-time 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.

Virtual Terminal (VT): subset of the IISS User Interface that performs the interfacing between different terminals and the UI. This is done by defining a specific set of terminal features and protocols which must be supported by the UI software which constitutes the virtual terminal definition. Specific terminals are then mapped against the virtual terminal software by specific software modules written for each type of real terminal supported.

Virtual Terminal Interface (VTI): The callable interface to the VT.

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

Window Field: A place-holder on a form for sub-forms which are to be supplied at run-time. A window field is dynamic, that is, based upon data which may change, a different sub-form may be placed in the window each time the containing form appears.

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

APPLICATION DEFINITION LANGUAGE (ADL)

The Application Definition Language (ADL) provides a precise and flexible method for defining applications. It is an extension of the Form Definition Language that allows you to:

- o Define the interactive environment that the application user will use to access the database (i.e., CDM data).
- o Define the non-interactive environment necessary for producing electronic or hard-copy reports.
- o Retrieve, delete, insert and modify the data in the database.
- o Perform simple statistical calculations on the information such as counts, sums, and averages as a function of the NDML retrieval process.

3.1 ADL Format Notation

This manual uses the following notation to describe the syntax of the ADL subset of Form Definition Language (FDL):

UPPER-CASE	identifies reserved words that have specific meanings in ADL. These words are generally required unless the portion of the statement containing them is itself optional.
lower-case	identifies names, numbers, or character strings that the user must supply.
Initial upper-case	identifies a statement or clause that is defined later on.
_ Underscores	identify reserved words or portions of reserved words that are optional.
{ } Braces	enclosing vertically stacked options indicate that one of the enclosed options is required.
[] Brackets	indicate that the enclosed clause or option is optional. When two or more options are vertically stacked within the brackets, one or none of them may be specified.
... Ellipsis	indicates that the preceding statement or clause may be repeated any number of times.
' ' Single Quotes	indicates that the literal representation of the element's characters is to be used.

_id a suffix which specifies an identifier.
_name a suffix which specifies a qualified name.
string specifies a character string.
int specifies an integer.
real specifies a real number.

All other punctuation is to be considered part of the language.

Application Definition

3.2 Application Definition Language

The Application Definition Language (ADL) subset of FDL provides a precise and flexible method for defining applications. ADL allows you to:

- o Define the interactive environment that the application user will use to access the database.
- o Define the non-interactive environment necessary for producing electronic or hard-copy reports.
- o Retrieve, delete, insert and modify the data in the database.
- o Perform simple statistical calculations on the information such as counts, sums, and averages as a function of the NDML retrieval process.

The database interaction functions of ADL are designed for use with relational databases. For this reason, a basic understanding of relational database concepts is very helpful. If you feel you need some background material on the subject before reading further, the following are suggested. The references are listed in order of the level of advance knowledge necessary.

Martin, James, Computer Data-base Organization

This book is an excellent reference for those unfamiliar with database concepts. Part 1 is a good introduction for a new user.

S. Bing Yao, ed., Principles of Database Design, Volume I: Logical Organizations

Chapter 6 specifically addresses relational databases.

Wiederhold, Gio, Database Design

Chapters 7, 8, 9, and 10 specifically address relational databases.

Ullman, Jeffrey D, Principles of Database Systems

Chapters 2, 5, 6, 7, 8, and 9 specifically address relational databases.

3.3 Application Definition Language Syntax

The collection of ADL statements that define an application is an application definition. An application definition is created by writing ADL statements directly to an ADL source file with any text editor you might use to prepare a program source file. The ADL source file is processed by the Rapid Application Generator to produce an interactive application executable. The Form and Condition definitions may be written in any order. The complete ADL Syntax is listed in this section. Sections 3.3.1 - 3.3.5.4.4 contain a detailed explanation of the syntax by statement and clause.

Application

```
CREATE ( APPLICATION )
      {      } application_id
      ( REPORT      )

      [ (parameter_form_id) ]

      [ KEYPAD( { key_id = int }... ) ]

      [ '%{' c_code '%}' ]

      Condition ...
```

Condition

```
ON ( Expression ) '{' Action ... '}'
```

Expression

```
( string
  item_name
  int
  flag_id
  ( Expression )
  - Expression
  Expression || Expression
  Expression + Expression
  Expression - Expression
  Expression * Expression
  Expression / Expression
  Expression < Expression
  Expression <= Expression
  Expression = Expression
  Expression != Expression
  Expression > Expression )
```

```

Expression >= Expression
Expression AND Expression
Expression OR Expression
NOT Expression
Expression ? Expression : Expression
INDEX(field_name)
BETWEEN(Expression, Expression,
        Expression)
IN(Expression, Expression,...)
GETATT(field_name, type)
GPAGE(window_name)
GWINDO(window_name, page)
APPEARS(field_name)
CURSOR(field_name)
ROLE(Expression)
PICK(Expression)
MODIFY(field_name)
STARTUP()
OVERFLOW(field_name)
CHANGE(item_name)
( EMPTY(item_name) )

```

Action

```

( PAGE
EXIT
'%' c_code '%'
SET item_name = Expression
SIGNAL [ NOT ] flag_id
Condition

( PRESENT )
( DISPLAY ) [ NOSELECT ] form_id [ IN window_name ]
( REDISPLAY )

( string )
HELP {
( form_id )

CALL procedure_id [ ( { ( PATH ) value
( { ( INTEGER ) item_name } ,... )
( { REAL )
]

SELECT [ '{' action ... '}' ]

INSERT INTO table_id (Col_spec ...) VALUES ( Value ... )

MODIFY table_id [ abbreviation_id ]
[ USING { table_id [ abbreviation_id ] } ,... ]
SET { Col_spec = Value } ...
WHERE where_expr

DELETE FROM table_id [ abbreviation_id ]
[ USING { table_id [ abbreviation_id ] } ,... ]
WHERE where_expr
)

```

Value

```
( string )
( int )
( real )
( item_name )
```

Select

```
( SELECT { Statistic ( [ DISTINCT ] item_name ) } ...
  FROM ( query_expr )
  [ WHERE where_expr ]

SELECT [ DISTINCT ] item_name ...
  FROM ( query_expr )
  [ WHERE where_expr ]

  [ ORDER BY { item_name ( ASCENDING )
                ( DESCENDING ) } } ... ]

SELECT { Statistic ( [ DISTINCT ] item_name = Col_spec ) }...
  FROM { table_id [ abbreviation_id ] } ,...
  [ WHERE where_expr ]

SELECT [ DISTINCT ] { item_name = Col_spec } ...
  FROM { table_id [ abbreviation_id ] } ,...
  [ WHERE where_expr ]

  [ ORDER BY { Col_spec ( ASCENDING )
                ( DESCENDING ) } } ... ] )
```

Statistic

```
( MAXIMUM )
( MINIMUM )
( AVG )

( AVERAGE )
( SUM )
( COUNT )
```

Col_spec

```
[ table_id .
  abbreviation_id . ] column_id
```

Query_expr

```
( Query_expr DIFFERENCE Query_expr
  Query_expr INTERSECT Query_expr
  Query_expr UNION      Query_expr
  ( Query_expr )

  SELECT ( Statistic ( [ DISTINCT ] Col_spec ) ) ...
    FROM ( Query_expr )
    [ WHERE Where_expr ]

  SELECT DISTINCT Col_spec ...
    FROM ( Query_expr )
    [ WHERE Where_expr ]

  SELECT ( Statistic ( [ DISTINCT ] Col_spec ) ) ...
    FROM ( table_id [ abbreviation_id ] ) ,...
    [ WHERE Where_expr ]

  SELECT DISTINCT Col_spec ...
    FROM ( table_id [ abbreviation_id ] ) ,...
    [ WHERE Where_expr ]
)
```

Where_expr

```
( Where_expr AND Where_expr
  Where_expr OR  Where_expr
  Where_expr XOR Where_expr
  NOT Where_expr
  Where_expr =  Where_expr
  Where_expr != Where_expr
  Where_expr >  Where_expr
  Where_expr >= Where_expr
  Where_expr <  Where_expr
  Where_expr <= Where_expr
  Where_expr == Where_expr
  Col_spec IS [ NOT ] NULL
  Col_spec [ NOT ] BETWEEN Value AND Value
  Col_spec
  Value
( ( Where_expr )
)
```

CREATE APPLICATION Statement

3.3.1 CREATE APPLICATION Statement

Every application definition must begin with the CREATE APPLICATION statement. This tells the compiler that what follows is an application definition. If the keyword REPORT is substituted for APPLICATION, then a report application is defined instead of an interactive application. The syntax for this statement is:

```
CREATE ( APPLICATION )
      { application_id
      ( REPORT ) }
```

CREATE	is the statement keyword.
REPORT	is a keyword which specifies that you are defining an output report (either hardcopy or to the screen).
APPLICATION	is a keyword which specifies that you are creating an interactive application. APPLICATION may be abbreviated AP.
application_id	is a unique name of up to 6 letters and/or numbers associated with each application or report. An application_name is required. The application_name cannot begin with a number. An application may have the same name as any form in the application definition. It is incorporated by the Rapid Application Generator into the Application name which is used to invoke the executable at run-time. The exact form of the Application name created is system dependent and therefore the length of application_name may be further restricted by the system you are running on.

CREATE APPLICATION Statement, parameter_form_id

3.3.2 Parameter form id

A parameter form is specified when you wish to have certain essential information entered before the application begins to run (for example, certain parameters for a report you wish to produce). A parameter form is defined, and when the user runs SYSGEN to set up the UI database for the specific application, the parameter form is added to the definition. The syntax for specifying a parameter form is:

(parameter_form_id)

parameter_form_id is the qualified name of the form which you wish to appear before the application begins processing. Refer to section 3.4 for further explanation of qualified names.

CREATE APPLICATION Statement, KEYPAD Clause

3.3.3 KEYPAD Clause

This clause allows you to give names to any of the programmable function keys so that they are accessible to the application. A keypad clause associated with a CREATE APPLICATION statement overrides all KEYPAD clauses which are associated with any of the forms accessed by the application. Once a function name has been assigned to a key, that function name is bound to that key, that is, it may not be applied to a different key within the same application. It is possible to apply more than one name to the same key. You may not apply the same name to more than one key. The syntax for the KEYPAD clause is:

KEYPAD({ key_id = int } ...)

KEYPAD is the clause keyword.

key_id is the name you choose to apply to the function key. The name can be a maximum of 10 alphabetic characters. You will use this name in ON PICK condition statements to specify the key that determines the action. The key_id = int phrase may be repeated.

int is the number of the programmable function key. int can be any number in the set {0, 4 through 20}.

CREATE APPLICATION Statement, Embedded C Code

3.3.4 Embedded C Code

When functionality beyond that provided by FDL is required, C Code can be embedded in the FDL source file in the following manner. To signal the Rapid Application Generator that what follows is C code, you enclose the code within the characters %{ and %} as follows:

```
'%{' C Code '%}'
```

You may then write the C code in the normal manner.

CREATE APPLICATION Statement, Condition

3.3.5 Condition Definition

The Condition portion of an application definition specifies pre-defined actions that will occur as the result of the evaluation of the specified expression as TRUE. The expressions include evaluation of: cursor positioning, function key selection, change of value in an item field, and the occurrence of certain values in an item field. Some of the possible pre-defined actions that can be triggered by a true evaluation include: deletion of data in a table, insertion of data into a table, modification of data in a table, and selection of data from a table. The syntax for the Condition is:

```
ON ( Expression ) '{' Action ... '}'
```

ON is the statement keyword.

Expression is a truth-valued mathematical or symbolic statement which must evaluate to TRUE before the specified Action will take place. The syntax for the Expression parameter is contained in section 3.3.5.1 of this manual.

Action specifies the action(s) to be performed when the specified condition evaluates to TRUE. The syntax for the Action parameter is defined in section 3.3.5.2 of this manual.

Condition Definition, Expression Parameter

3.3.5.1 Expression Parameter

The expression parameter is used with the VALUE clause to specify the default value of a field, the APPEARS IF clause to specify when a field appears on its containing form, or as a condition definition to determine when to take a pre-defined action. The syntax for expression is:

```
( string
  item_name
  int
  flag_id
  ( Expression )
  - Expression
  Expression || Expression
  Expression + Expression
  Expression - Expression
  Expression * Expression
  Expression / Expression
  Expression < Expression
  Expression <= Expression
  Expression = Expression
  Expression != Expression
  Expression > Expression
  Expression >= Expression
  { Expression AND Expression
    Expression OR Expression
    NOT Expression
    Expression ? Expression : Expression
    INDEX(field_name)
    BETWEEN(Expression, Expression,
      Expression)
    IN(Expression, Expression,...)
    GETATT(field_name, type)
    GPAGE(window_name)
    GWINDO(window_name, page)
    APPEARS(field_name)
    CURSOR(field_name)
    ROLE(Expression)
    PICK(Expression)
    MODIFY(field_name)
    STARTUP()
    OVERFLOW(field_name)
    CHANGE(item_name)
  ( EMPTY(item_name) )
```

Condition Definition, Expression Parameter

String is a character string enclosed in double quotes ("default value").

item_name is a qualified name enclosed in single quotes ('field') identifying an item field whose value you want to use. (Refer to section 3.4 for a further explanation of qualified names.)

int is an integer value.

flag_id is a user-defined identifier which contains a boolean valued state indicator which may be used in condition and set expressions.

(Expression) specifies the use of parenthesis for grouping when combining expressions.

- Expression specifies the negative of an expression as a value.

NOT Expression specifies the logical negation of a boolean expression.

Expression Binop Expression

specifies default values, appears if criterion, and condition definitions as the result of combining expressions using binary operators. In many cases, these combinations are truth-valued expressions with values of 1 or 0. The binary operators are:

(| |)
+
-
*
/
=
{ != }
<
<=
>
>=
AND
(OR)

Condition Definition, Expression Parameter

These operators have the usual meanings and precedences. The operator `||` is used for string concatenation and has the same precedence as `+` and `-`. Operands of the wrong type (i.e., character vs. integer) are automatically converted.

`Expression ? Expression : Expression`

is a conditional assignment expression. If the expression to the left of the `?` evaluates to true then the expression to the right of the `?` is evaluated. If it is false, then the expression to the right of the `:` is evaluated.

`INDEX (field_name)`

specifies that the value is the name of the first displayed element of the array `"field_name"`. The array must be on the same form as `field` being defined and must be enclosed in single quotes (i.e., `INDEX('myfield')`). For example, if `"myfield"` is a two dimensional array that has been scrolled once horizontally and twice vertically, `INDEX('myfield')` might return `"myfield(2,3)"`.

`GETATT (field_name, type)`

specifies the value as an item field's attribute.

`GPAGE (field_name, page)`

is specifying the value as the qualified name of the form in a specified page of a window.

`GWINDO (field_name)`

is specifying the value as the number of pages in a window.

`APPEARS (field_name)`

is a boolean expression that is true if the specified field is displayed and false if the field is not displayed.

Condition Definition, Expression Parameter

BETWEEN (Expression, Expression, Expression)

is a boolean expression that is true if the first expression value is between the value of the second and third expression values and false if it is not.

CURSOR (field_name)

is a boolean expression that is true if the cursor is in the specified field and false if it is not.

IN (Expression, Expression, ...)

is a boolean expression that is true if the first expression value is in the remaining set of expression values.

ROLE (Expression)

is a boolean expression that is true if the role of the current logged on user is the specified value and false if it is not.

OVERFLOW (field_name)

allows you to specify one or more actions to be performed when the size of a field is exceeded. Field_name is the prefix of an item which is the target of a SELECT statement. A repeating field can cause the overflow of its containing field by repetition in either the horizontal or vertical direction. You specify the name of the field that causes the condition action to be triggered. For example, a set of nested forms with a one row repeating item field at the lowest level could be output with interruptions at the line level or at a higher level. This allows the application to require that a sub-form be output either in its entirety or not at all. OVERFLOW specifies that one or more actions will occur if the size of a named field is exceeded. Field name is the qualified name of the field to be evaluated.

Condition Definition, Expression Parameter

CHANGE(item_name)

allows you to define one or more actions to be performed when the value of a named item field changes. If the item field being tested for a changing value contains database values, then it may be appropriate for these values to have been previously sorted so that all similar values are grouped together. This sorting can be achieved by using the "ORDER BY" option of the SELECT statement (see section 3.3.5.4 in this manual). If the test for a changing value is positive, the actions are taken after the item field is substituted with the changed value. CHANGE specifies that one or more actions will be performed when the value of a named item field changes.

STARTUP ()

allows you to define one or more actions to be performed at the beginning of the application. This condition is required and should include as one of its actions the presentation of an initial form.

PICK (Expression)

allows you to define one or more actions to be performed when the user presses a programmable function key which you defined in the form definition KEYPAD clause.

MODIFY(field_name)

allows you to define one or more actions to be performed when the value of an item field or item fields of a form are modified by the user. The modify condition checks to see if the field has been modified since the last function key pick. The MODIFY condition is different from the CHANGE condition. The CHANGE condition checks for changes in an item field which has been targeted by a select. The MODIFY condition checks for user changes to an item or item fields of a form between function key picks.

Condition Definition, Expression Parameter

EMPTY(item_name)

is true if the select statement to the specified item name returns zero rows from the database.

Condition Definition, Action Parameter

3.3.5.2 Action Parameter

Actions are to be taken when the specified ON condition is true. The condition action syntax is:

```
( PAGE
  EXIT
  '%{' c_code '%'
  SET item_name = Expression
  SIGNAL [ NOT ] flag_id
  Condition

  ( PRESENT )
  { DISPLAY } [ NOSELECT ] form_id [ IN window_name ]
  ( REDISPLAY )

  HELP { string }
  { form_id }

  CALL procedure_id [ ( { ( PATH ) value
                        { ( INTEGER ) item_name } ,... )
                      ( REAL )
                    ) ]

  SELECT [ '(' action ... ')' ]

  INSERT INTO table_id (Col_spec ...) VALUES ( Value ... )

  MODIFY table_id [ abbreviation_id ]
    [ USING { table_id [ abbreviation_id ] } ,... ]
    SET { Col_spec = Value } ...
    WHERE where_expr

  DELETE FROM table_id [ abbreviation_id ]
    [ USING { table_id [ abbreviation_id ] } ,... ]
    WHERE where_expr )
```

Condition Definition, Action Parameter

PAGE	The PAGE action outputs the current display list to the current logical device.
'%{' 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. C code is inserted directly into the FDL source file. The characters %{ signal the start of the C code and %} signals the end of the C code.
SET item_name = Expression	This action sets the value of a item to another field or a specified expression.
EXIT	This action terminates the application.
SIGNAL [NOT] flag_id	This action allows the boolean valued state information (flag_id) to be set to true (or false if the NOT option is used). Any top level conditions using the flag_id are evaluated immediately following this action and if true are executed. Upon termination of the condition the state information is set to false and flow of control continues with the action following the SIGNAL.
Condition	A condition (defined in section 3.3.5) can be nested within another condition and used as an action. Thus, the action of one true condition is to evaluate another condition. 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.

Condition Definition, Action Parameter

```
( PRESENT      )
( DISPLAY      ) [NOSELECT] form_id [IN window_name]
( REDISPLAY    )
```

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.

The 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 suppressed by using the NOSELECT option.

```
HELP ( string )
HELP {         }
      ( form_id )
```

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

```
CALL procedure_id [ ( ( ( PATH      ) value
                      ( INTEGER ) item_name ) ,... )
                   ( REAL      ) ) ]
```

Condition Definition, Action Parameter

The purpose of the CALL action is to allow the application to make calls to Form Processor procedures, other system services, or procedure written by the user in order to perform specialized computations. The CALL action invokes the procedure specified by `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 modifiers `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.

```
INSERT INTO table_id (Col_spec ...) VALUES (Value ...)
```

This action inserts data into the database table. The values specified in the `VALUE` clause will be inserted into the specified column of the specified database table. The syntax for the `Value` parameter is presented in section 3.3.5.3.

```
DELETE FROM table_1 [ abbreviation_1 ]  
  [ USING { table_2 [ abbreviation_2 ] } ,... ]  
  WHERE where_expr
```

This action deletes data from the database table. `Table_1` is the full name of the database table from which the data is to be deleted. `Abbreviation_1` allows you to define an abbreviated name for the database table. You may use this name to refer to the table throughout the delete action.

The `USING` clause specifies the table(s) referenced in the `WHERE` clause. These tables will not be affected by the delete. `Table_2` specifies the name of the table(s) referenced in the `WHERE` clause. `Abbreviation_2` allows you to specify an abbreviated name for the table(s) referenced in the `WHERE` clause.

Condition Definition, Action Parameter

The WHERE clause allows you to specify criteria which determines which rows in the FROM table_1 will be deleted.

Where_expr specifies the criteria for the delete. The syntax for the Where expression is contained in section 3.3.5.4.4 of this manual.

```
MODIFY table_1 [ abbreviation_1 ]  
  [ USING [ table_2 [ abbreviation_2 ] ] ,... ]  
  SET { Col_spec = Value } ...  
  WHERE where_spec
```

This action modifies data in the database table specified by table_1. Abbreviation_1 allows you to define an abbreviated name for the database table. You may use this name to refer to the table throughout the modify action.

The USING clause allows you to specify the table(s) which are referenced in the WHERE clause. Those tables referenced by the WHERE clause will not be affected by the MODIFY. Table_2 is the full name of the table referenced in the WHERE clause. Abbreviation_2 allows the specification of an abbreviated name to refer to the table referenced in the WHERE clause.

SET is a keyword which allows you to set the specified column equal to the value specified. Col_spec is the name of the database column you wish to modify. The syntax for col_spec is listed in section 3.3.5.4.2 of this manual. Value specifies the value which is to be mapped to the column specified by col_spec. The syntax for the Value parameter is listed in section 3.3.5.3 of this manual.

Condition Definition, Action Parameter

The WHERE clause allows you to specify criteria which determine which rows of the MODIFY table_1 are to be changed.

where_expr specifies the criteria required for the modify. The syntax for the Where expression is listed in section 3.3.5.4.4 of this manual.

Value Parameter

3.3.5.3 Value Parameter

The Value parameter is used to specify specific values to be used as arguments in the CALL action, to be inserted into a database table, or placed into a database table via the MODIFY action. The syntax for the Value parameter is:

```
( string      )  
  int         )  
( real       )  
  item_name   )  
(             )
```

string specifies a character string enclosed in double quotes (" ").

int specifies an integer.

real specifies a real number.

item_name is the qualified name of an item whose value is to be referenced.

Condition Definition, SELECT

3.3.5.4 SELECT Statement

The SELECT statement is used to retrieve information from either a database table or from the set of data which results from a previous SELECT. There are several different options so that the SELECT may be customized to meet the specific needs of the user. The syntax for the SELECT statement is:

```
( SELECT { Statistic ( [ DISTINCT ] item_name ) } ...
    FROM ( query_expr )
    [ WHERE where_expr ]

SELECT [ DISTINCT ] item_name ...
    FROM ( query_expr )
    [ WHERE where_expr ]
    ( ASCENDING )
    [ ORDER BY { item_name {
    ( DESCENDING )
} } ... ]

SELECT { Statistic ( [ DISTINCT ] item_name = Col_spec ) }...
    FROM { table_id [ abbreviation_id ] } ,...
    [ WHERE where_expr ]

SELECT [ DISTINCT ] { item_name = Col_spec } ...
    FROM { table_id [ abbreviation_id ] } ,...
    [ WHERE where_expr ]
    ( ASCENDING )
    [ ORDER BY { Col_spec {
    ( DESCENDING )
} } ... ]
)
```

Each SELECT option shall be addressed separately. The first option to be addressed is:

```
SELECT { Statistic ( [ DISTINCT ] item_name ) } ...
    FROM ( query_expr )
    [ WHERE where_expr ]
```

This option is chosen when you wish to perform a statistic function on the set of data resulting from set operators in the query expression and return the resulting value into the item specified by the qualified name item_name. For example, a possible use of this option would be:

```
SELECT MAXIMUM('forma.salary')
    FROM (SELECT DISTINCT employees.salary
          FROM employees emp
          WHERE emp.job = "programmer")
```

Condition Definition, SELECT

This example chooses all unique salary columns from the employees database table where the job column in the same row holds the value "programmer". It then chooses the maximum salary from among the columns selected and places that value into the item specified by the qualified name 'forma.salary'.

Statistic	is a statistical function to be performed on the values resulting from the select. The statistic functions are listed in section 3.3.5.4.1 of this manual.
DISTINCT	is a keyword used to specify that duplicate rows are to be ignored. Item_name is the qualified name which specifies the item to receive the resulting value.
item_name	is the qualified name of the item into which the result of the statistic function is to be placed. Qualified names are discussed in section 3.4 of this manual.
FROM	is a keyword which specifies that what follows is an expression which determines where the data is selected. The syntax for the Query expression is listed in section 3.3.5.4.3 of this manual.
WHERE	is a keyword which specifies a condition which must be true in order for the select to take place. The syntax for the Where expression is listed in section 3.3.5.4.4 of this manual.

The second option is:

```
SELECT [ DISTINCT ] item_name ...  
FROM ( query_expr )  
[ WHERE where_expr ]  
[ ORDER BY { item_name { ASCENDING } } ... ]  
[ ORDER BY { item_name { DESCENDING } } ... ]
```

Condition Definition, SELECT

This option allows you to SELECT a set of data from the table(s) specified by the nested selects in the query_expression and sort the resulting set of data by a specific item_name. An example usage of this option is:

```
SELECT DISTINCT 'forma.salary(0)'  
FROM (SELECT DISTINCT employees.salary  
      FROM employees emp  
      WHERE emp.job = "programmer")  
ORDER BY 'forma.salary' ASCENDING
```

This example chooses all unique salary columns from the employees database table where the job column in the same row holds the value "programmer". It then places the resulting values into the item array specified by the qualified name 'forma.salary(0)' and then sorts 'forma.salary(0)' in ascending order.

DISTINCT	is a keyword used to specify that duplicate rows are to be ignored.
Item_name	is the qualified name which specifies the item to receive the resulting value. Qualified names are discussed in section 3.4 of this manual.
FROM	is a keyword which specifies that what follows is an expression which determines where the data is selected. The syntax for the Query expression is listed in section 3.3.5.4.3 of this manual.
WHERE	is a keyword which specifies a condition which must be true in order for the select to take place. The syntax for the Where expression is listed in section 3.3.5.4.4 of this manual.
ORDER BY	are keywords which specify that the results of the select are to be sorted. ASCENDING and DESCENDING are keywords to specify the order in which to sort the values.

The third option is:

```
SELECT { Statistic ( [ DISTINCT ] item_name = Col_spec ) }...  
FROM { table_id [ abbreviation_id ] } ,...  
[ WHERE where_expr ]
```

Condition Definition, SELECT

This option is very similar to the first option discussed. The difference is that the first option included the use of nested SELECTs to choose the database table(s). This option selects directly from a database table. For example, you might choose to use this select:

```
SELECT MAXIMUM(DISTINCT 'forma.salary' = employees.salary')
FROM employees emp
WHERE emp.job = "programmer"
```

This example chooses all unique salary columns from the employees database table where the job column in the same row holds the value "programmer". It then chooses the maximum salary from among the columns selected and places that value into the item specified by the qualified name 'forma.salary'.

statistic	is a statistical function to be performed on the values resulting from the select. The statistic functions are listed in section 3.3.5.3 of this manual.
DISTINCT	is a keyword used to specify that duplicate rows are to be ignored.
Item_name	is the qualified name which specifies the item to receive the resulting value. Qualified names are discussed in section 3.4 of this manual.
Col_spec	is the full name of the column of the database table from which to select the data. The syntax for col_spec is listed in section 3.3.5.4.2 of this manual.
FROM	is a keyword which specifies that what follows are tables the data are to be selected from.
table_id	is the name of the database table from which to select the data.
abbreviation_id	is a method of abbreviating the name of the database table. Once an abbreviation_id has been assigned to a table, the table may be referred to by that abbreviation for the duration of the select.

Condition Definition, SELECT

WHERE is a keyword which specifies a condition which must be true in order for the select to take place. The syntax for the Where expression is listed in section 3.3.5.4.4 of this manual.

The final SELECT option to be discussed is:

```
SELECT [ DISTINCT ] { item_name = Col_spec } ...  
  FROM { table_id [ abbreviation_id ] } , ...  
  [ WHERE where_expr ]  
    [ ORDER BY { Col_spec { ASCENDING } } ... ]  
      { DESCENDING }
```

This option is very similar to the second option discussed. The difference is that this option does not use nested selects; it selects directly from the database table. For example:

```
SELECT DISTINCT 'forma.salary(0)' = employees.salary  
  FROM employees emp  
  WHERE emp.job = "programmer"  
  ORDER BY emp.salary DESCENDING
```

This example chooses all unique salary columns from the employees database table where the job column in the same row holds the value "programmer". It then sorts the columns in descending order and places the values in the item array specified by the qualified name 'forma.salary(0)'.

DISTINCT is a keyword used to specify that duplicate rows are to be ignored.

Item_name is the qualified name which specifies the item to receive the resulting value. Qualified names are discussed in section 3.4 of this manual.

Col_spec is the full name of the column of the database table from which to select the data. The syntax for col_spec is listed in section 3.3.5.4.2 of this manual.

FROM is a keyword which specifies that what follows are tables the data are to be selected from.

Condition Definition, SELECT

Table_id	is the name of the database table from which to select the data.
Abbreviation_id	is a method of abbreviating the name of the database table. Once an abbreviation_id has been assigned to a table, the table may be referred to by that abbreviation for the duration of the select.
WHERE	is a keyword which specifies a condition which must be true in order for the select to take place. The syntax for the Where expression is listed in section 3.3.5.4.4 of this manual.
ORDER BY	are keywords which specify that the results of the select are to be sorted. ASCENDING and DESCENDING are keywords to specify the order in which to sort the values.

Statistic Functions

3.3.5.4.1 Statistic Functions

The following statistic functions are provided for use with the SELECT statement:

```
( MAXIMUM )  
| MINIMUM |  
{ AVERAGE }  
| SUM     |  
( COUNT  )
```

MAXIMUM	The maximum value returned by the SELECT is placed in the item specified by the qualified name item_name. A discussion of qualified names appears in section 3.4 of this manual. MAXIMUM may be abbreviated MAX.
MINIMUM	The minimum value returned by the SELECT is placed in the item specified by the qualified name item_name. A discussion of qualified names appears in section 3.4 of this manual. MINIMUM may be abbreviated MIN.
AVERAGE	all of the values returned by the SELECT statement are totaled and divided by the number of non-null occurrences. This number is placed in the item specified by the qualified name item_name. AVERAGE may be abbreviated AVG. A discussion of qualified names appears in section 3.4 of this manual.

SUM all of the values returned by the SELECT statement are totaled and this number is placed in the item specified by the qualified name `item_name`. A discussion of qualified names appears in section 3.4 of this manual.

COUNT the number of values returned by the SELECT statement is placed in the item specified by the qualified name `item_name`. A discussion of qualified names appears in section 3.4 of this manual.

Col_spec Parameter

3.3.5.4.2 Col spec Parameter

A `col_spec` is the full name of the column in the specified table which contains the value which is to be operated upon. The syntax for a `col_spec` is:

```
[ table_id . ] column_id  
[ abbreviation_id . ]
```

`table_id` is the name of the table which contains the column.

`abbreviation_id` is an abbreviated name for the table which was specified previous to using the abbreviation.

.

is used to separate the column name from the table name or abbreviation.

`column_id` is the name of the column in the table.

Query Expression

3.3.5.4.3 Query Expression

A query expression is an expression which allows the user to specify where the application is to select the data from. Query expressions are used to enable the user to specify set operators, that is, a select within a select. Query expressions are recursive. The syntax for query_expression is:

```
( Query_expr DIFFERENCE Query_expr
  Query_expr INTERSECT Query_expr
  Query_expr UNION      Query_expr
  ( Query_expr )

  SELECT { Statistic ( [ DISTINCT ] Col_spec ) } ...
    FROM ( Query_expr )
    [ WHERE Where_expr ]

  SELECT DISTINCT Col_spec ...
  { FROM ( Query_expr )
    [ WHERE Where_expr ] }

  SELECT { Statistic ( [ DISTINCT ] Col_spec ) } ...
    FROM { table_id [ abbreviation_id ] } ,...
    [ WHERE Where_expr ]

  SELECT DISTINCT Col_spec ...
    FROM { table_id [ abbreviation_id ] } ,...
    [ WHERE Where_expr ] )
```

Query_expr DIFFERENCE Query_expr

specifies that any two of the SELECT statements below may be executed. A DIFFERENCE function is then performed on the results of the two SELECTs. This means that the results of query 1 are compared to the results of query 2. The rows which were returned by query 1 but NOT returned by query 2 are called the DIFFERENCE. For example, If query 1 returned A, B, and C, and query 2 returned C, D, and E, taking the DIFFERENCE of the queries returns A and B.

Query Expression

Query_expr INTERSECT Query_expr

specifies that any two of the SELECT statements below may be executed. An INTERSECT function is then performed on the results of the two SELECTs. This means that the results of query 1 are intersected with the results of query 2. The rows which were returned by query 1 AND query 2 are called the INTERSECTION. For example, if query 1 returned A, B, and C, and query 2 returned C, D, and E, the INTERSECTION of the two returns C.

Query_expr UNION Query_expr

specifies that any two of the SELECT statements below may be executed. An UNION function is then performed on the results of the two SELECTs. This means that the results of query 1 are unioned with the results of query 2. All of the rows which were returned by both of the queries minus the duplicate rows are called the UNION. For example, if query 1 returned A, B, and C, and query 2 returned C, D, and E, the UNION of the two returns A, B, C, D, and E.

(Query_expr)

indicates that parentheses may be used to group query_expressions.

The following SELECTs function just as those described in section 3.3.5.4 of this manual. The difference is that these statements specify the Column Specification directly instead of depending upon a nested SELECT to perform that function. The query expression always specifies a SELECT which is subordinate to the SELECT which specified the query expression.

Query Expression

```
SELECT { Statistic ( [ DISTINCT ] Col_spec ) } ...  
      FROM ( Query_expr )  
      [ WHERE Where_expr ]
```

in this case, the query expression in the FROM clause returns a set of rows from the specified table(s). Then the SELECT statement selects specific column(s) out of the set of rows. The selected columns are passed back to the containing SELECT statement (one of those described in section 3.3.5.4) which performs a specified statistical function (described in section 3.3.5.4.1 of this manual).

```
SELECT DISTINCT Col_spec ...  
      FROM ( Query_expr )  
      [ WHERE Where_expr ]
```

in this case, the query expression in the FROM clause returns a set of rows from the specified table(s). Then the SELECT statement selects specific column(s) out of the set of rows. The selected columns are passed back to the containing SELECT statement (one of those described in section 3.3.5.4) which selects further.

```
SELECT { Statistic ( [ DISTINCT ] Col_spec ) } ...  
      FROM { table_id [ abbreviation_id ] } ,...  
      [ WHERE Where_expr ]
```

this option SELECTs data directly from the specified table and column, performs a statistical function (described in section 3.3.5.4.1 of this manual) on the result.

```
SELECT DISTINCT Col_spec ...  
      FROM { table_id [ abbreviation_id ] } ,...  
      [ WHERE Where_expr ]
```

this option SELECTs data directly from the specified table and column.

Where Expression

3.3.5.4.4 Where Expression

The where expression is used to specify the circumstances under which a piece of data is to be retrieved. A SELECT statement retrieves data from database table rows where the condition(s) specified in the WHERE clause are true. Where expressions can be recursive. The most basic where expressions are Col_spec and Value. It is generally not useful to specify either Col_spec or Value alone. They are intended to be specified in combination with each other and with specific binary operators. The syntax for a Where expression is:

```
( Where_expr AND Where_expr
  Where_expr OR  Where_expr
  Where_expr XOR Where_expr
  NOT Where_expr
  Where_expr =   Where_expr
  Where_expr !=  Where_expr
  Where_expr >   Where_expr
  Where_expr >=  Where_expr
  Where_expr <   Where_expr
  Where_expr <=  Where_expr
  Where_expr ==  Where_expr
  Col_spec IS [ NOT ] NULL
  Col_spec [ NOT ] BETWEEN Value AND Value
  Value
  ( ( Where_expr ) ) )
```

Where_expr AND Where_expr

This where expression specifies that the criteria specified by BOTH where expressions must be TRUE for a row of data in order for it to be selected. An example is:

```
WHERE emp.salary > 10000 AND emp.job = "clerk"
```

In order for a row of data to be selected, the salary column of that row must have a value greater than 10000 AND the job column must have the value "clerk".

Where Expression

Where_expr OR Where_expr

This where expression specifies that the criteria specified by either one or BOTH of the where expressions must be TRUE for a row of data in order for it to be selected. For example:

```
WHERE emp.salary > 10000 OR emp.job = "clerk"
```

In order for a row of data to be selected, either the salary column of that row must have a value greater than 10000 OR the job column must have the value "clerk". If both are true, the expression is considered true.

Where_expr XOR Where_expr

This where expression specifies that the criteria specified by either one of the where expressions but not both must be TRUE for a row of data in order for it to be selected. An example is:

```
WHERE emp.salary > 10000 XOR emp.job = "clerk"
```

In order for a row of data to be selected, either the salary column of that row must have a value greater than 10000 OR the job column must have the value "clerk". If both are true, the expression is considered false.

NOT Where_expr

This where expression specifies that the logical negation of the criteria specified by the where expression must be true for a row of data in order for it to be selected. For example:

```
WHERE NOT emp.salary > 10000
```

In order for a row of data to be selected, the value contained in the salary column of that row must not be greater than 10000.

Where Expression

The following expressions depend upon binary operators. These operators have the usual meanings and precedences. An example will be given for each.

Where_expr = Where_expr

Example:

WHERE emp.salary = 10000

Where_expr != Where_expr

Example:

WHERE emp.salary != 10000

Where_expr > Where_expr

Example:

WHERE emp.salary > 10000

Where_expr >= Where_expr

Example:

WHERE emp.salary >= 10000

Where_expr < Where_expr

Example:

WHERE emp.salary < 10000

Where_expr <= Where_expr

Example:

WHERE emp.salary <= 10000

Where Expression

Where_expr == Where_expr

Example:

WHERE emp.dept_no == dept.dept_no

Col_spec IS [NOT] NULL

This where expression allows you to check for NULL (empty) columns. For example:

emp.salary IS NULL

allows you to select rows where the salary column is empty.

emp.salary IS NOT NULL

allows you to select rows where the salary column is not empty.

Col_spec [NOT] BETWEEN Value AND Value

This where expression allows you to check to see if the value contained in a column falls within a specific range. For example:

emp.salary BETWEEN 10000 AND 17500

You might also use this expression to select for rows which fall outside of a specific range. For example:

emp.salary NOT BETWEEN 15000 and 25000

Col_spec

The column specification is used in combination with other where expressions to form compound expressions. In all of the previous examples, emp.salary is a column specification. The syntax for Col_spec is contained in section 3.3.5.4.2 of this manual.

Where Expression

Value

Value is used in combination with other where expressions to form compound where expressions. In the previous examples, 10000, 15000, "clerk", etc. are examples of Values. The syntax for the Value expression is contained in section 3.3.5.3 of this manual.

(Where_expr)

This expression uses parentheses to change the precedence of where expressions.

Qualified Names

3.4 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 an asterisk * (e.g., 'myarray(*)'). This means all elements of the array will be used sequentially in the condition or action.

The "for each" quantifier index is a zero (e.g., 'myarray(0)'). This symbolic index is normally used in a qualified name which is the target of a SELECT action, or a condition or action which is nested in the SELECT. For each row retrieved by the select the current index is accessed in the qualified name.

SECTION 4

HOW TO CREATE AN APPLICATION DEFINITION

You create an application definition by writing ADL statements directly to a text file using any text editor.

4.1 Statement Format

ADL statements can be entered in free format. Free format means that keywords and numbers can be separated by any number of spaces. The compiler treats tabs, comments, and carriage returns as spaces.

Application definition statements may be in any order.

Form field names used in the application definition syntax are pseudo qualified names in the form hierarchy. They are denoted by strings of the following type: 'form.item'. The single quotes are required. (Refer to section 3.4 for more information on qualified names).

4.2 Restrictions

Every application definition must begin with the CREATE APPLICATION or CREATE REPORT statement.

There must be at least one space before and after every keyword in the syntax.

4.3 Abbreviations

Underscores in the ADL syntax indicate reserved words or portions of reserved words that are optional.

4.4 Including Comments

You can include comments in an application definition by enclosing the comment text between /* and */. Comments are treated as spaces by the ADL compiler. For example:

```
CREATE APPLICATION appl1      /* inventory application */
```

4.5 Reserved Words

This is an alphabetized list of the reserved words in the Application Definition Language subset of FDL.

ABOVE	ABS	ABSOLUTE
ADD	ADDITIVE	ALL
AND	AP	APPEARS
APPLICATION	AS	ASC

ASCENDING	ASSIGN	AT
ATTRIBUTE	AVERAGE	AVG
AXIS	BACKGROUND	BAR
BELOW	BETWEEN	BOTTOM
BOX	BY	CALL
CENTER	CHANGE	CNT
COL	COND	CONDITIONAL
COLOR	COLUMN	COUNT
CURVE	DELETE	DESC
DESCENDING	DIFFERENCE	DISPLAY
DISTINCT	DOMAIN	END
ENTER	EVERY	EXIT
EXPLODE	FILL	FINE
FONT	FORM	FREQ
FROM	GRAPH	GRID
H	HELP	HORIZONTAL
IF	IN	INSERT
INSIDE	INTEGER	INTERSECT
INTO	IS	ITEM
KEY	KEYPAD	LABEL
LEFT	LEGEND	LINE
LINEAR	LINE TYPE	LINE WIDTH
LOG	LOWER	MARKER
MAX	MAXIMUM	MIN
MINIMUM	MODIFY	MUST
NILL	NODUP	NOSELECT
NOT	NULL	NUM
NUMERIC	OF	ON
OR	ORDER	OUTSIDE
OVERFLOW	PAGE	PATH
PATTERN	PERCENT	PIC
PICTURE	PIE	PRESENT
PROMPT	QUANTITY	REAL
REDISPLAY	RELATIVE	REPORT
RIGHT	ROW	SCALE
SELECT	SET	SHADE
SIGNAL	SIZE	SPACE
SPACES	START	STARTUP
STYLE	SUM	SUMMARY
SYMBOL	SYMBOL_FREQUENCY	TICK
TO	TOP	UNION
UPPER	UPVECTOR	USING
V	VALUE	VALUES
VERSUS	VERTICAL	WHERE
WINDOW	WITH	XOR

4.6 How to Define A Report Application

This section explains how to use the ADL to define a report. First, the simple case of a one page report using a one table database is described. The simple case is then expanded to a report that contains more than one page and a database that contains two tables.

4.6.1 A Simple One Page Report

Your company requires each department to produce an employee payroll report. The format of the report is shown in Figure 4-1. The employee data is stored in the database table EMPDATA. The structure of the table is shown in Figure 4-2.

```

DEPARTMENT PAYROLL REPORT
*****
"Deptname" Department
*****

EMPNO      NAME      MONTHLY      COMM      ANNUAL
-----      -      -      -      -
XXXXX      Name      $9999.99    $9999.99    $99,999.99
.           .           .           .           .
.           .           .           .           .
.           .           .           .           .

```

Figure 4-1 Report Format

EMPDATA					

Name	Salary	Comm	Compen	Empno	*****

*** Denotes Key Field

Figure 4-2 Structure of EMPDATA Table

4.6.1.1 Specifying the Report Format

You specify the report format using form definitions. Report forms are similar to electronic forms in that they can contain item and form fields. To determine how to define these forms, you need to think of the report as it will appear in its final form (i.e., after it has been produced on the output medium). Ask the questions:

- o What information appears only once?
- o What information is repeated?
- o Does any of the information repeat in groups?

With the answers to these questions, you will be able to develop a form hierarchy for your report. The form hierarchy for the Payroll Report is shown in Figure 4-3.

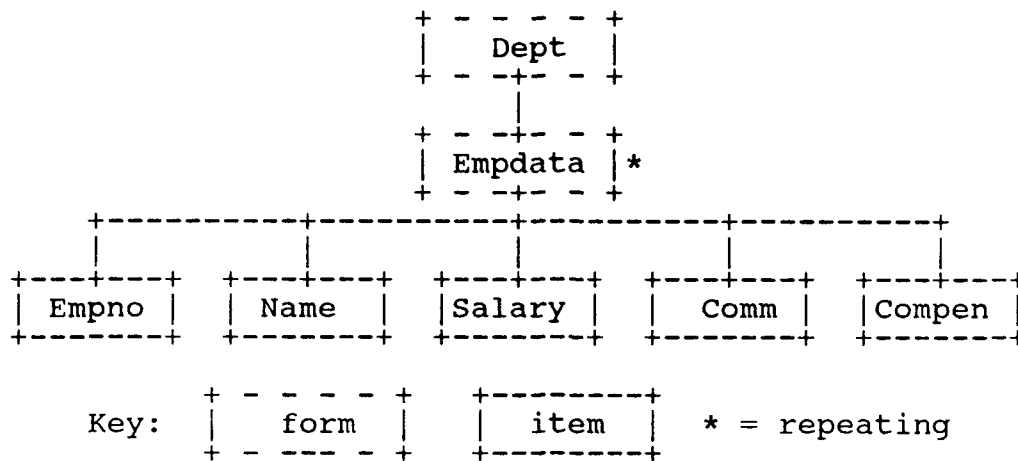


Figure 4-3 Form Hierarchy for the Payroll Report

The form Dept represents the finished report. The titles, headings, and other formatting characters are defined as prompts since they only appear once. The employee data is defined as the repeating form field Empdata on Dept. Empdata will be repeated for every record in the database table. The form Empdata contains the item fields to display the actual database information for each employee. The FDL source for this portion of the report definition is:

```

CREATE REPORT Payroll
CREATE FORM Dept
  SIZE 80 by 23
  PROMPT center at 2 40 "DEPARTMENT PAYROLL REPORT"
  PROMPT at 4 20
    "*****"
  PROMPT center at 6 40 "SALES DEPARTMENT"
  PROMPT at 8 20
    "*****"
  PROMPT at 10 10 "EMPNO"
  PROMPT at 10 21 "NAME"
  PROMPT at 10 33 "MONTHLY"
  PROMPT at 10 43 "COMM"
  PROMPT at 10 54 "ANNUAL"
  PROMPT at 11 33 "SALARY"
  PROMPT at 11 54 "COMPENSATION"
  PROMPT at 12 10 "-----"
  PROMPT at 12 16 "-----"
  PROMPT at 12 33 "-----"
  PROMPT at 12 43 "-----"
  PROMPT at 12 54 "-----"
  FORM Empdata (* v 0) at 14 9 SIZE 55 by 1
CREATE FORM Empdata
  ITEM Empno at 1 2
    DISPLAY AS text
    DOMAIN (pic "99999")

```

```
ITEM Name at 1 13
      DISPLAY AS text SIZE 15 by 1
ITEM Salary at 1 25
      DISPLAY AS text
      DOMAIN (pic "$9999v99")
ITEM Comm at 1 35
      DISPLAY AS text
      DOMAIN (pic "$9999v99")
ITEM Compen at 1 46
      DISPLAY AS text
      DOMAIN (pic "$99,999v99")
```

4.6.1.2 Retrieving the Database Information

You retrieve the database information using NDML SELECT commands. Selection of data takes place as a result of a Condition statement. First you must specify the Condition which in this case is STARTUP. Then you write the select statement specifying the item fields on the form Empdata as the fields that receive the data retrieved by the SELECT. Because Empdata is a repeating form, you must use the subscript (0) to indicate the current occurrence of the form. You must also specify that the form must be presented with a Present statement. The FDL source for this portion of the report definition is:

```
ON (STARTUP())
{
  SELECT 'dept.empdata(0).empno'    = Empno
        'dept.empdata(0).name'    = Name
        'dept.empdata(0).salary'  = Salary
        'dept.empdata(0).comm'    = Comm
        'dept.empdata(0).compen'  = Compen
        FROM Empdata
        ORDER BY Empno
  PRESENT Dept
  Page
}
```

NOTE: Qualified names must be enclosed in single quotes. If a name is qualified by a repeating form, that form name must end with the subscript 0 as shown above in 'dept.empdata(0).empno'. This specifies the first occurrence of the repeating form.

This SELECT will make the employee data available as needed by the repeating form Empdata.

4.6.2 The Multi-Page Report

The previous example suggests that the employee data for each department is in a separate table. These separate tables could be combined into one table by changing the structure of EMPDATA. The new structure of the EMPDATA table is shown in Figure 4-4. From the one table you can now generate one report that is formatted as shown in Figure 4-5. Items that have been added to the previous example are shown in bold.

EMPDATA

Deptno	Deptname	Name	Salary	Comm	Compen	Empno
*****						*****

*** Denotes Key Field

Figure 4-4 New Table Structure for EMPDATA

***** Page 1				
DEPTNO: XXX - - DEPTNAME: "DEPT"				

EMPNO	NAME	MONTHLY SALARY	COMM	ANNUAL COMPENSATION
XXXXX	Name	\$9999.99	\$9999.99	\$99,999.99
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
***** Page 2				
DEPTNO: XXX - - DEPTNAME: "DEPT"				

EMPNO	NAME	MONTHLY SALARY	COMM	ANNUAL COMPENSATION
XXXXX	Name	\$9999.99	\$9999.99	\$99,999.99
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

Figure 4-5 Format of Payroll Report from Combined Tables

4.6.2.1 Specifying the Multi-Page Report Format

The form hierarchy for this report is little changed from the previous example because most of the additions are achieved using ON Conditions. The new Payroll Report form hierarchy is shown in Figure 4-6.

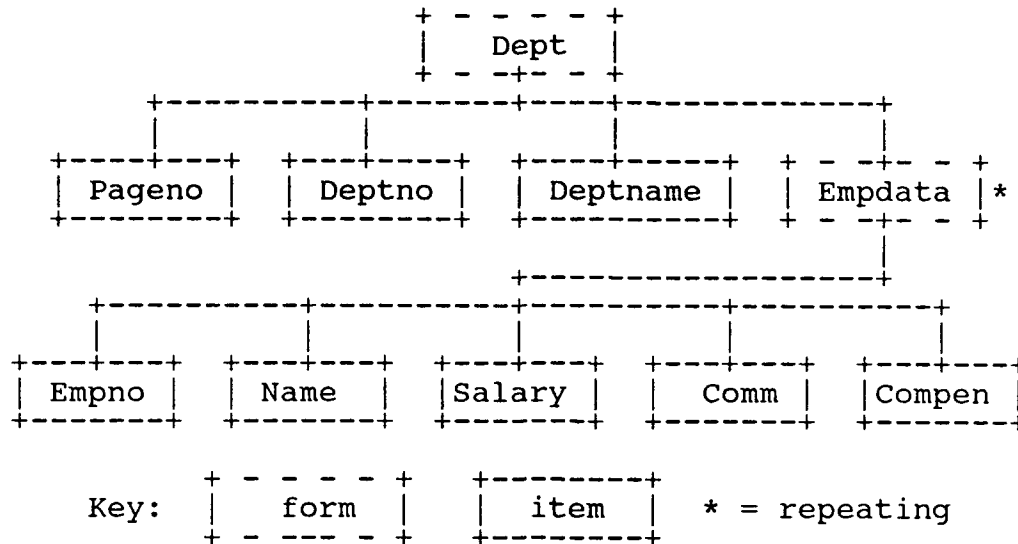


Figure 4-6 Form Hierarchy for the Multi-Page Report

This report now contains data for more than one department and it may contain more than one page. To accommodate this, the item fields Deptno, Deptname and Pageno are added to the form Dept. In order for the employee information to overflow on the form Dept, Dept must be a fixed size form. Empdata will be repeated on Dept for every employee in that department. The FDL source for this portion of the report definition is:

```
CREATE REPORT Payroll
CREATE FORM Dept SIZE 80 by 23
  ITEM Pageno at 1 70 VALUE '._PAGENO'
  PROMPT at 5 20
    "*****"
    DISPLAY AS text
    PROMPT AT left "Page"
  PROMPT at 1 20
    "*****"
  ITEM Deptno at 3 27
    DISPLAY AS text
    DOMAIN (pic "999")
  PROMPT at LEFT "DEPTNO: "
  ITEM Deptname at 3 42
    DISPLAY AS text
  PROMPT at LEFT "- - DEPTNAME: "
  PROMPT at 7 10 "EMPNO"
  PROMPT at 10 21 "NAME"
  PROMPT at 10 33 "MONTHLY"
  PROMPT at 10 43 "COMM"
  PROMPT at 10 54 "ANNUAL"
  PROMPT at 11 33 "SALARY"
  PROMPT at 11 54 "COMPENSATION"
  PROMPT at 12 10 "-----"
  PROMPT at 12 16 "-----"
  PROMPT at 12 33 "-----"
  PROMPT at 12 43 "-----"
  PROMPT at 12 54 "-----"
  FORM Empdata (* v) at 14 9 SIZE 55 by 1
CREATE FORM Empdata
  ITEM Empno at 1 2
    DISPLAY AS text
    DOMAIN (pic "99999")
  ITEM Name at 1 13
    DISPLAY AS text SIZE 15 by 1
  ITEM Salary at 1 25
    DISPLAY AS text
    DOMAIN (pic "$99999v99")
  ITEM Comm at 1 35
    DISPLAY AS text
    DOMAIN (pic "$99999v99")
  ITEM Compen at 1 46
    DISPLAY AS text
    DOMAIN (pic "$99,999v99")
```

4.6.2.2 Paging the Report

To page the report you define an ON OVERFLOW condition. This allows the overflow of one item or form, on an item or a form that is higher in the form hierarchy, to trigger a specified action. In this case, the action you want to occur is paging. For this report, you want the overflow of the form Empdata to trigger a new page. Because Empdata is a repeating form, you use the subscript (0) to indicate the current occurrence of the form. You must also specify that the form

Dept will be presented on the top of the new page to provide the headings to the Empdata information. The FDL source for this portion of the report definition is:

```
ON (OVERFLOW ('dept. Empdata(0)') )
{
  PAGE
  PRESENT Dept
}
```

The item field Pageno on the form Dept is given the value '._PAGENO'. This value is incremented by 1 every time paging occurs.

4.6.2.3 Grouping Database Information

The payroll information on this report needs to be grouped by department. This is done by defining an ON CHANGE condition. An ON CHANGE condition allows the change in an item field value to trigger an action or group of actions. For this report, whenever the value of the item field Deptno changes, there should be a page eject and the form Dept should be repeated for the next department. This ON CHANGE condition allows each department to start on a new page. The FDL source for this portion of the report definition is:

```
ON (CHANGE ('dept.deptno') )
{
  PAGE
  PRESENT Dept
}
```

This condition implies that the database information is sorted by department number. This is accomplished by using the ORDER BY option of the SELECT command. The FDL source for this portion of the report definition is:

```
ON (STARTUP())
{
  SELECT 'dept.deptno'      = Deptno
        'dept.deptname'    = Deptname
        'dept.empno'       = Empno
        'dept.name'        = Name
        'dept.salary'      = Salary
        'dept.comm'        = Comm
        'dept.compen'      = Compen
  FROM Empdata
  ORDER BY Empdata.Deptno Empdata.Empno
  Present Dept
  PAGE
}
```

4.6.2.4 Using Nested SELECT Commands

The database for the Combined Payroll Report could contain two separate tables. The structures of these tables is shown in Figure 4-7.

DEPTDATA	
Deptno	Deptname

EMPDATA					
Deptno	Name	Salary	Comm	Compen	Empno
*****					*****

*** Denotes Key Field

Figure 4-7 Table Structures for DEPTDATA and EMPDATA

These two tables have a "one to many" relationship. This means that for every record in DEPTDATA, there are several records in EMPDATA. This database configuration maps to the relationship between the Dept and Empdata forms in the form hierarchy. By retrieving the information from the database using nested SELECT commands, the data will be presented as defined by the form hierarchy. The FDL source for this portion of the report definition is:

```

SELECT 'dept.deptno'    = Deptno
      'dept.deptname'  = Deptname
FROM Deptdata
ORDER BY Deptdata.Deptno
{
  SELECT 'dept.empdata(0).empno'    = Empno
        'dept.empdata(0).name'     = Name
        'dept.empdata(0).salary'   = Salary
        'dept.empdata(0).comm'     = Comm
        'dept.empdata(0).compen'   = Compen
FROM Empdata
WHERE Empdata.Deptno = 'Deptno'
ORDER BY Empdata.Empno
}

```

Data retrieval and data presentation go hand in hand and are not independent of one another. The nested selects acquire the data in the proper way. The nesting means that the data in the inner loop(s) are periodically exhausted thus stopping the occurrence of the form(s) using that data (e.g., Empdata). Without the nested SELECT, Empdata would continue repeating whether or not Empdata.Deptno matched Deptdata.Deptno.

SECTION 5

HIERARCHICAL REPORT WRITER

A hierarchical report consists of a set of boxes which are connected into a tree structure. The Hierarchical Report Writer can be used to produce two types of hierarchical reports:

- o A true hierarchy such as is shown in an organization chart for a management structure where each employee has a single manager.
- o A network hierarchy such as a parts list where one part may be used in different assemblies which in turn are part of a single larger assembly.

The Hierarchical Report Writer is a post-processor which takes a report generated by the Application Generator and rearranges it into an appropriate tree structure. The data to be displayed can be either a true hierarchy where each box appears only once (as shown in Figure 5-1), or a network where a box may appear more than once (as shown in Figure 5-2). The boxes appearing below a box are referred to as its expansion.

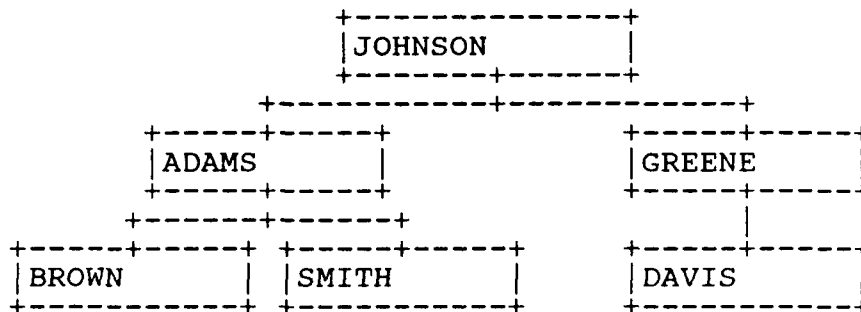


Figure 5-1 Format of a Hierarchical Report

When a network is displayed, the expansion of an item is only displayed below the first occurrence of the box. If the report is paged, the page number of the expansion is displayed below subsequent occurrences of the item. If the report is not paged, then a "*" is displayed in place of the expansion. This is illustrated in Figure 5-2 where "PART 1" occurs twice.

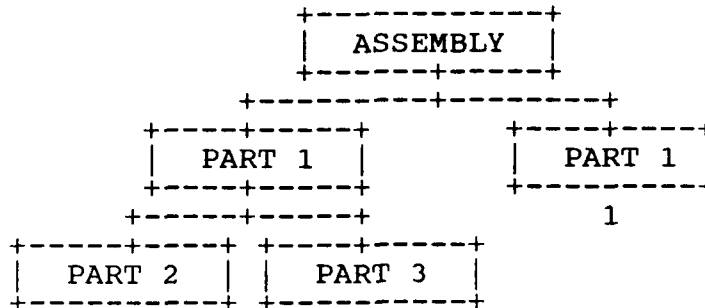


Figure 5-2 Format of an Assembly Hierarchy

5.1 Defining a Hierarchical Report

A Hierarchical Report contains a separate page for each box in the hierarchy. A single page contains the graphic representation of the box, the name of the box, and the names of the boxes which are to be connected below it. The format of the printed hierarchical report that is produced is shown in Figure 5-3.

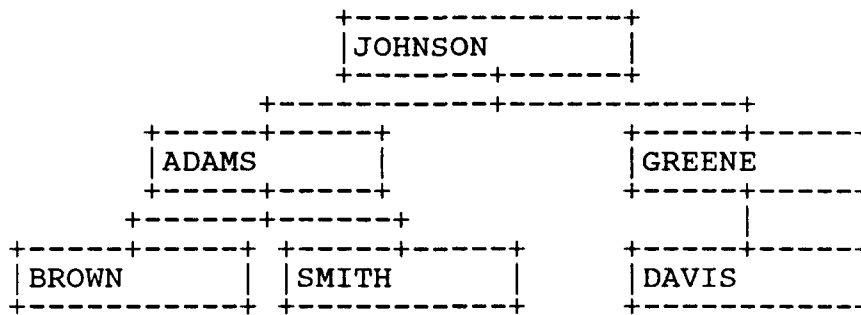


Figure 5-3 Format of a Printed Hierarchical Report

The table used as the source of the data is shown in Figure 5-4.

Supervisor	Name
-----	Johnson
Johnson	Adams
Johnson	Greene
Adams	Brown
Adams	Smith
Greene	Davis

Figure 5-4 Empdata Table

The FDL source for the initial report is:

Create Report Orgchart

```

On (Startup())
{
  SELECT 'orgchart.abox(0).name' = name
        'orgchart.abox(0).empno' = empno
  FROM Empdata
  {
    SELECT 'orgchart.abox(0).refs(0).name' = name
    FROM Empdata
    WHERE supervisorno = 'orgchart.abox(0).empno'
  }
  PRESENT orgchart
  page
}

on (overflow('orgchart.abox(0)'))
{
  page
  present orgchart
}
  
```

CREATE FORM orgchart
SIZE 60 BY 23

FORM abox (* vertical with 0 spaces)
AT 2 1
SIZE 60 BY 22

CREATE FORM abox
SIZE 60 BY 22
PROMPT AT 1 23 "|"
PROMPT AT 2 2
"+-----+"
PROMPT AT 3 2 "|"
PROMPT AT 3 44 "|"
PROMPT AT 4 2
"+-----+"
PROMPT AT 5 23 "|"
PROMPT AT 6 2 ")"

ITEM name
DISPLAY AS text
AT 3 4
SIZE 30

ITEM empno
DISPLAY AS text
AT 4 RIGHT OF name
SIZE 6
PROMPT AT LEFT "-"

FORM refs (* VERTICAL WITH 0 SPACES)
AT 8 1
SIZE 40
PROMPT AT 1 below and column 2 ")"

CREATE FORM refs

ITEM name
DISPLAY AS text
AT 1 2
SIZE 30

The initial report that is produced by the Report Writer is shown in Figure 5-5.

```
      |  
+-----+  
| JOHNSON |  
+-----+  
      |  
)  
JOHNSON  
ADAMS  
GREENE  
)  
.  
.  
.  
<NEW PAGE>  
      |  
+-----+  
|      ADAMS      |  
+-----+  
      |  
)  
ADAMS  
BROWN  
SMITH  
)  
.  
.  
.  
<NEW PAGE>  
      |  
+-----+  
|      GREENE      |  
+-----+  
      |  
)  
GREENE  
DAVIS . .
```

Figure 5-5 Format of the Initial Report

This initial report has the following characteristics which are required in a report that is used as input to a hierarchical report:

- o The first line of the page is blank (the form ABox starts on line 2 of the form Orgchart).
- o The first repeating form (ABox) is the same size as the space reserved for it on the page (Orgchart), forcing each box to appear on a separate page (this may also be accomplished by using an ON CHANGE condition).
- o The first line of the box contains a single character indicating the upper connection point (this character will be blanked out for a box which does not have any boxes above it).
- o The last line of the box contains a single character indicating the lower connection point (this character will be blanked out for a box which does not have any boxes below it).
- o A line containing a single right parenthesis (")") separates the box from the connection information.
- o The first line of connection information contains the name of this box.
- o Following lines of connection information contain the names of the boxes that are connected below this box, one per line.
- o The end of the connection information is indicated by another line containing a single right parenthesis. Anything between this line and the beginning of the next page (e.g. the message line) will be ignored.

5.2 Accessing the Hierarchical Report Writer

The Hierarchical Report Writer is available as an application in the IISS environment. Accessing an application is explained in the IISS Terminal Operator Guide.

Type "SDHRWZZZZZ" in the function field of the IISS Function Screen to display the Hierarchical Report Writer form. Then enter the following information:

Input File	Enter the name of the file that contains the generated report to be postprocessed into a tree structure.
------------	--

Output File Enter the name of the file (or device) that is to contain the reformatted report.

Width Enter the desired width of the reformatted report pages in characters.

Depth Enter the number of lines desired on the reformatted report pages. Alternatively, if a zero is entered, the reformatted report will not be divided into pages, but will instead be divided into a number of strips which can be pasted side-by-side to form a single large page containing the tree.

New Pages Enter a "Y" if you have a network and want the expansion of each box which appears more than once to be on a page by itself rather than appearing under the first occurrence of the box.

Invert Enter a "Y" if you want to turn the tree structure "upside down" (i.e. the connection information will be interpreted as the boxes above this box, rather than the boxes below). For example, if you have a report detailing a parts list (i.e. the parts and subassemblies which make up each assembly), you can get a parts usage list (i.e. the assemblies which contain each part) by inverting the original report.

When you are satisfied that you have entered the correct information in these items, press the <ENTER> key and the reformatted report will be written to the selected output file.

APPENDIX A

STEPS FOR EXECUTING THE RAPID APPLICATION GENERATOR

Below is the procedure to use in invoking the Rapid Application Generator for this release. This procedure assumes that the NTM is up and running and the user is logged on to IISS. Refer to the Terminal Operator Guide for the procedure for logging on to IISS. The following conventions are used to document this section:

- o Text in angle brackets is to be replaced with appropriate information by the user.
- o Single upper case words enclosed in angle brackets represent terminal keys (e.g. <ENTER>).
- o Text in upper case is to be entered as shown.

1 \$ @GENAP This starts the application generator. Respond to the prompts as follows:

Option Number?	6
FDL File?	TESTXX
CDM Username/Password?	CDM/CDM
Logical Unit of Work?	TESTXX
Host?	VAX
Delete Obsolete Code?	Y
IBM Databases?	N

Replace TESTXX with TESTAP for an interactive application or TESTRW for a report.

Steps 2 and 3 update the UI database.

2 Return to IISS and on the IISS Function Screen fill in the fields as follows:

Function: SYSGEN
Press <ENTER>

- 3 When the initial SYSGEN screen is displayed:

Press <PF7>

When the next screen is displayed, enter the Function:

DBMOD

Press <PF7>

When the Function Definition screen is displayed, fill in the fields as follows:

Description: Database Modification Program

AP Name: SDDBMODZZZ

Press <ENTER>

Enter the authorized role as follows:

SYSMGR

Press <ENTER>

Press <QUIT> to return to the initial SYSGEN screen.

Press <QUIT> to end SYSGEN and return to the IISS Function

To run the generated application program, perform step 13.

- 4 On the IISS Function Screen, fill in the item as follows:

FUNCTION: DBMOD Press <ENTER>

When the generated application is complete, another function may be selected.