For: conn
Printed on: Thu, Jun 18, 1992 07:48:38
From book: Ada LRM SDD
Document: SDD
Last saved on: Tue, May 12, 1992 08:17:45

1. Scope

The Ada LRM Reader is a tool for browsing through an online copy of the Ada Language Reference Manual (LRM). This tool allows a user to interactively view the Ada LRM, search for strings, and move through the Ada LRM with ease. Ease of human interface is a chief concern.

The target user is assumed to have a VT100−style display terminal or VT100 emulation capabilities. The user will be using this tool in one of several modes:

1. As a user on a UNIX workstation running in a VT100 emulator window,

2. As a user accessing a UNIX workstation remotely, also running a VT100 or VT100 emulator on a PC, and

3. As a user on a PC running the Ada LRM Reader on the PC.

The Ada LRM Reader is a single program written in Ada and will be considered to be a single CSCI. This CSCI includes:

1. The source code, in Ada, of the Ada LRM Reader

2. All data files needed by the Ada LRM Reader

3. All source files and programs used to create the data files needed by the Ada LRM Reader

4. All documentation associated with the Ada LRM Reader

5. Installation instructions for compiling the setting up the Ada LRM Reader for a PC or UNIX platform

6. A complete executable version of the Ada LRM Reader with its associated data files and installation instructions which is ready to run on a PC under MSDOS 3.3 or higher

2. References

2.1. Documents

Conn, Richard, **Software Requirements Specification for the Ada LRM Reader**, University of Cincinnati, Department of Electrical and Computer Engineering, Mail Location 30, Cincinnati, Ohio 45221

This document contains other references which may be useful.

Conn, Richard, **Software User's Manual for the Ada LRM Reader**, University of Cincinnati, Department of Electrical and Computer Engineering, Mail Location 30, Cincinnati, Ohio 45221

2.2. Terminology

The following application–specific terms are defined below in order to better follow this document:

Citation – A body of text in the Ada LRM or an LRM Support File which is uniquely identified by a numeric reference or a keyword (these are called citation labels). For example, the citation identified by 4 is:

```
                     4. Names and Expressions


    The rules applicable to the different forms of name and expression, and
    to their evaluation, are given in this chapter.
```

Citation Label – A numeric reference or a keyword which identifies a citation. For example, 4 is the citation label for the above citation. Valid citation labels take the following forms:

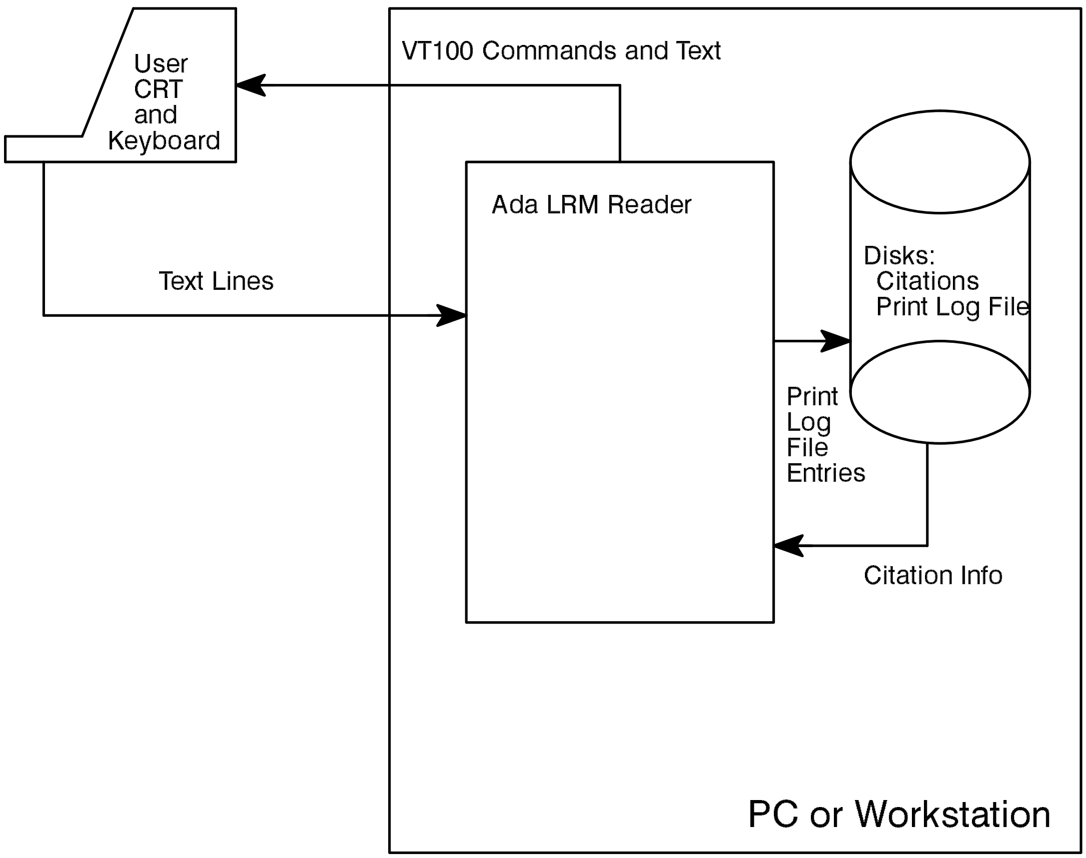
Label	*Refers to*
n	Chapter (1–14)
n.n	Chapter and Section
n.n.n	Chapter, Section, and Subsection
letter	Appendix (A–F)
CONTENTS	Table of Contents
INDEX	Index
FOREWARD	Foreward
POSTSCRIPT	Postscript
HELP	Online help screens for the Ada LRM Reader
ABOUT	Online program description of the Ada LRM Reader

Citation ID – An alphabetic reference used in the CITATION_ID enumeration type which maps to a citation label. Citation IDs are discussed in the Software Design Document for the Ada LRM Reader. Each citation label has one and only one citation ID.

3. Design

3.1. CSCI overview

The Ada LRM Reader is a program (composed of a single CSCI) which interacts with a user through a command–line and a line–oriented editor to display information in the Ada Language Reference Manual (LRM) to the user one screen at a time. The major components of this environment are:



VT100 commands are sent to the user's VT100–compatible CRT to clear screen and position cursor. Text sent to the user's CRT is simply displayed. The SUM shows the two basic screen displays.

The text lines sent from the user's keyboard are strings of ASCII characters terminated by a newline. The operating system's or shell's input line editor allows the user to edit these lines before the Ada LRM Reader "sees" them.

Entries in the Print Log File are created by the user when he issues the PRINT and PS commands (described in the SUM). An entry created by the PRINT command is the ASCII text of an entire citation. An entry created by the PS command is the ASCII text of the current screen.

The citation comes from files on disk. These citations include the numbered paragraphs, the table of contents, the foreward, the postscript, and the index. The HELP and ABOUT screens are also viewed as "citations."

3.1.1. CSCI architecture

The Ada LRM Reader CSCI consists of the following CSCs:

1. The LRM_Reader, the mainline procedure; this is one of the three active tasks in the CSCI

2. The MAKE_DAF procedure, which reads the *.doc files that contain the text of the Ada LRM and creates a corresponding set of *.daf files (DAF stands for Direct Access File); this is one of the three active tasks in the CSCI

3. The MAKE_CIT procedure, which reads the *.daf files and creates a CIT.ADA file which contains the enumeration type CITATION_ID (that contains values corresponding to every citation label in all the *.daf files) and a Citation_Location_Vector (CLV) array that contains a mapping between each citation and a record of data that contains an ID of the corresponding *.daf file, the starting line number of the citation in the *.daf file, and the ending line number of the citation in the *.daf file; this is one of the three active tasks in the CSCI

4. The Citation_Definition package, which contains the definition of the type CITATION_ID and the CLV array; the type CITATION_ID is an enumerated type that contains names for each citation (such as C4P1 for Chapter 4 Section 1) and the CLV array contains records associated with each CITATION_ID value that identifies the *.daf file containing the citation and the starting and ending record numbers in the *.daf file of the citation; the Citation_Definition package is created by the MAKE_CIT procedure

5. The SYSDEP package, which contains all the System Dependency information

6. The DAF_Handler package, which is used to create and access the information in the *.daf files

7. The Print_Log_Handler package, which is used to create and write information to the Print_Log file

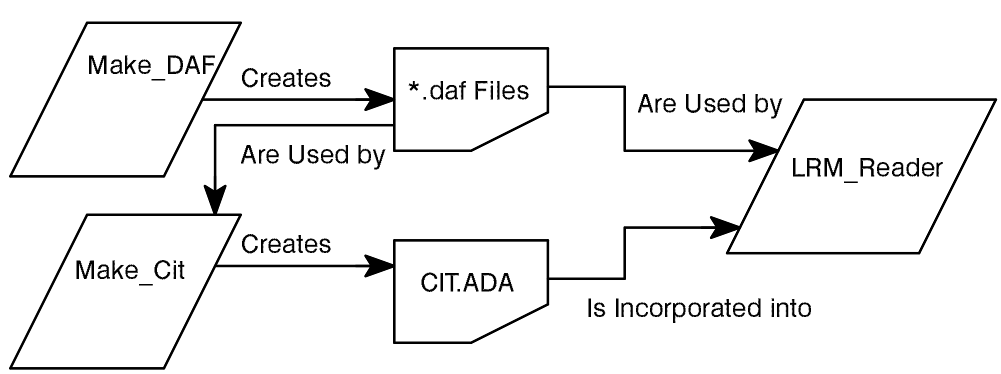
8. The Screen_Display_Controller package, which is used to display information to the user's screen

9. The Command_Dispatcher package, which is used to start the Ada LRM Reader and interface with the user's keyboard

10. The Citation_Handler package, which is used by the Command_Dispatcher to access the desired citation

11. The Primitive_Citation_Handler package, which provides low-level functions for accessing citations and is used by Citation_Handler, Screen_Display_Controller, and Print_Log_Handler

Of these CSCs, the first three are active objects (tasks) and the rest are definitions (Citation_Definition and SYSDEP) and passive objects which are invoked by the three active objects:

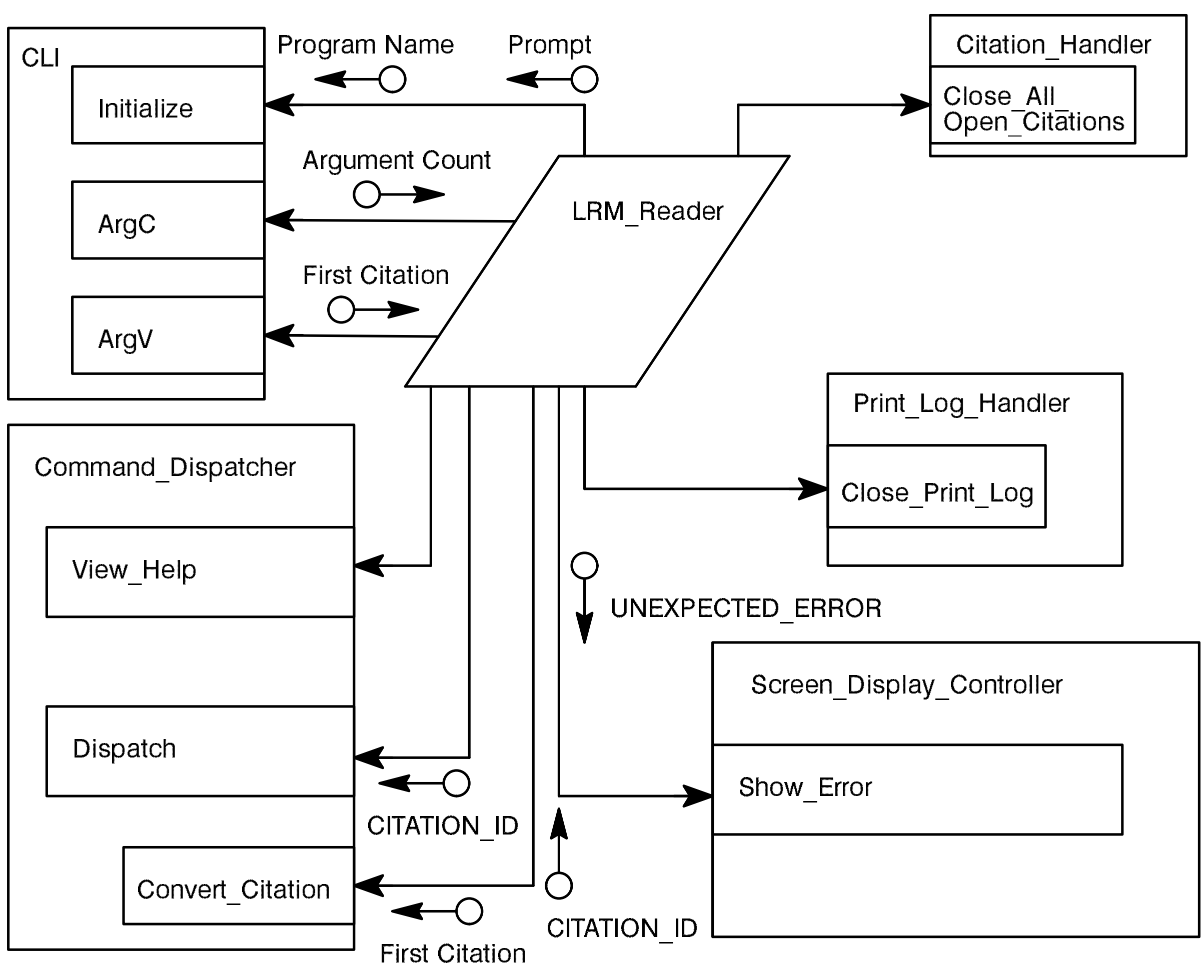


In addition, the following reusable CSCs provide some needed functionality:

1. Package CLI (part of CS Parts), which is a Command Line Interface
2. Package Console (part of CS Parts), which is a VT100 interface
3. Packages Input_File and Output_File (part of CS Parts), which support ASCII text file manipulation
4. Package System (part of the standard Ada environment), which provides address manipulation
5. Package Direct_IO (part of the standard Ada environment), which provides direct access file manipulation
6. Procedure Unchecked_Conversion (part of the standard Ada environment), which provides the ability to map addresses to pointers

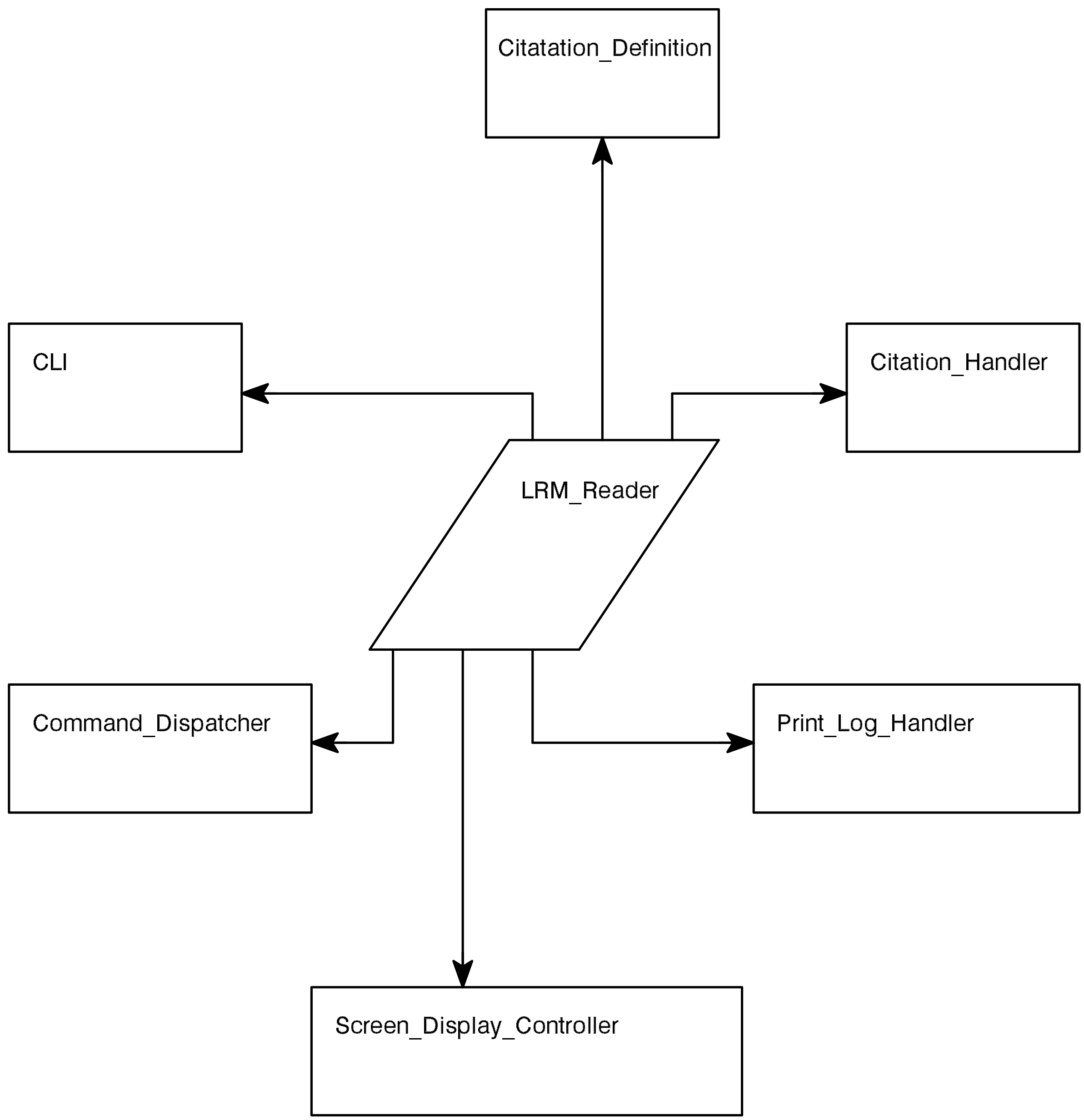
3.1.1.1. Object Interaction Diagram for the LRM_Reader Task

The following Object Interaction Diagram (OID) shows the principal CSC (the LRM_Reader) and the objects (also CSCs) with which it interfaces. Note that only the pertinent parts of the object interfaces are shown, as opposed to the entirety of the object interfaces.



3.1.1.2. Dependency Diagram for the LRM_Reader Task

The following Dependency diagram (simplified Booch diagram) shows the top−level view of the dependencies of the LRM_Reader CSC:

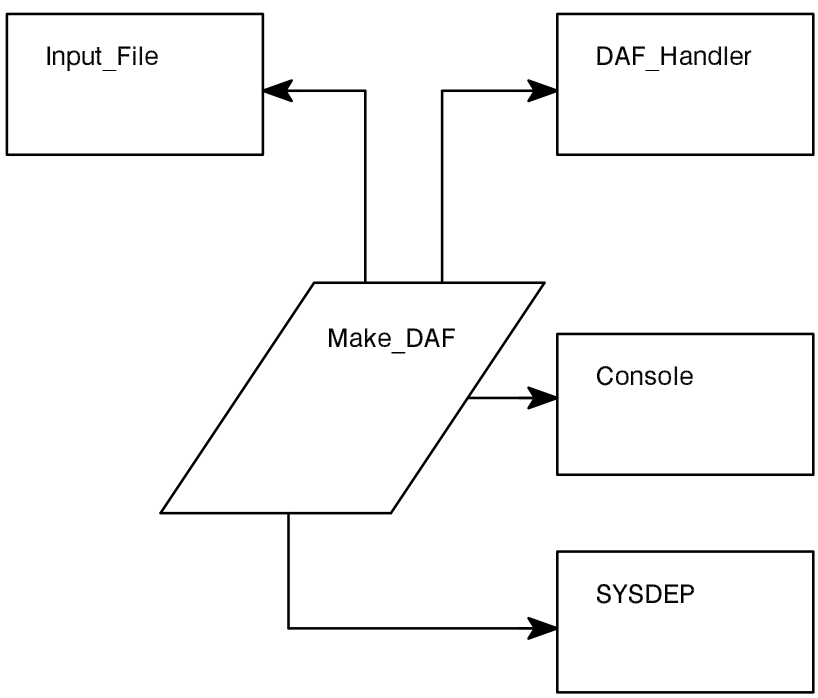


The key CSCs shown in this dependency diagram are:

1. The LRM_Reader, the mainline procedure
2. The Citation_Definition package, which contains the definition of the type CITATION_ID and the CLV (Citation_Location_Vector) array
3. Package CLI (part of CS Parts), which is a Command Line Interface
4. The Print_Log_Handler package, which is used to create and write information to the Print_Log file
5. The Screen_Display_Controller package, which is used to display information to the user's screen
6. The Command_Dispatcher package, which is used to start the Ada LRM Reader and interface with the user's keyboard
7. The Citation_Handler package, which is used by the Command_Dispatcher to access the desired citation

3.1.1.3. Dependency Diagram for the Make_DAF Task

The following Dependency diagram (simplified Booch diagram) shows the top-level view of the dependencies of the Make_DAF CSC:



The key CSCs shown in this dependency diagram are:

1. The MAKE_DAF procedure, which reads the *.doc files that contain the text of the Ada LRM and creates a corresponding set of *.daf files (DAF stands for Direct Access File)

2. The SYSDEP package, which contains all the System Dependency information

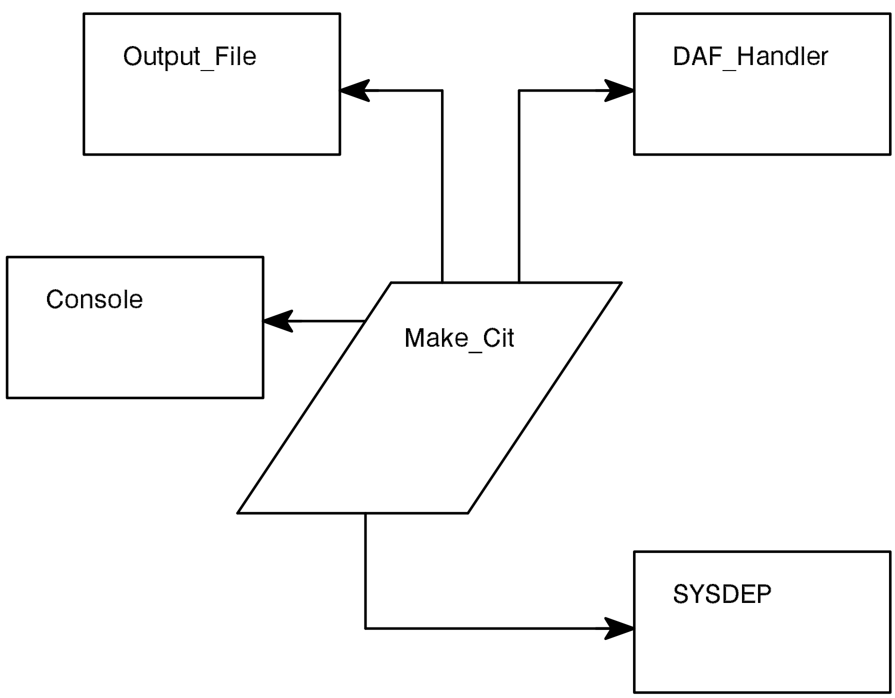
3. The DAF_Handler package, which is used to create and access the information in the *.daf files

4. Package Input_File (part of CS Parts), which provides text file manipulation

5. Package Console (part of CS Parts), which provides the ability to display to the console

3.1.1.4. Dependency Diagram for the Make_Cit Task

The following Dependency diagram (simplified Booch diagram) shows the top–level view of the dependencies of the Make_Cit CSC:



The key CSCs shown in this dependency diagram are:

1. The MAKE_CIT procedure, which reads the *.daf files and creates a CIT.ADA file which contains the enumeration type CITATION_ID (that contains values corresponding to every citation in all the *.daf files) and a Citation_Location_Vector (CLV) array that contains a mapping between each citation and a record of data that contains an identification of the corresponding *.daf file, the starting line number of the citation in the *.daf file, and the ending line number of the citation in the *.daf file

2. The SYSDEP package, which contains all the System Dependency information

3. The DAF_Handler package, which is used to create and access the information in the *.daf files

4. Package Output_File (part of CS Parts), which provides text file manipulation
5. Package Console (part of CS Parts), which provides the ability to display to the console

3.1.2. System States for LRM_Reader Task

This SDD only documents the LRM_Reader task in terms of its states because the other two tasks are far less complex and do not require an elaborate design model.

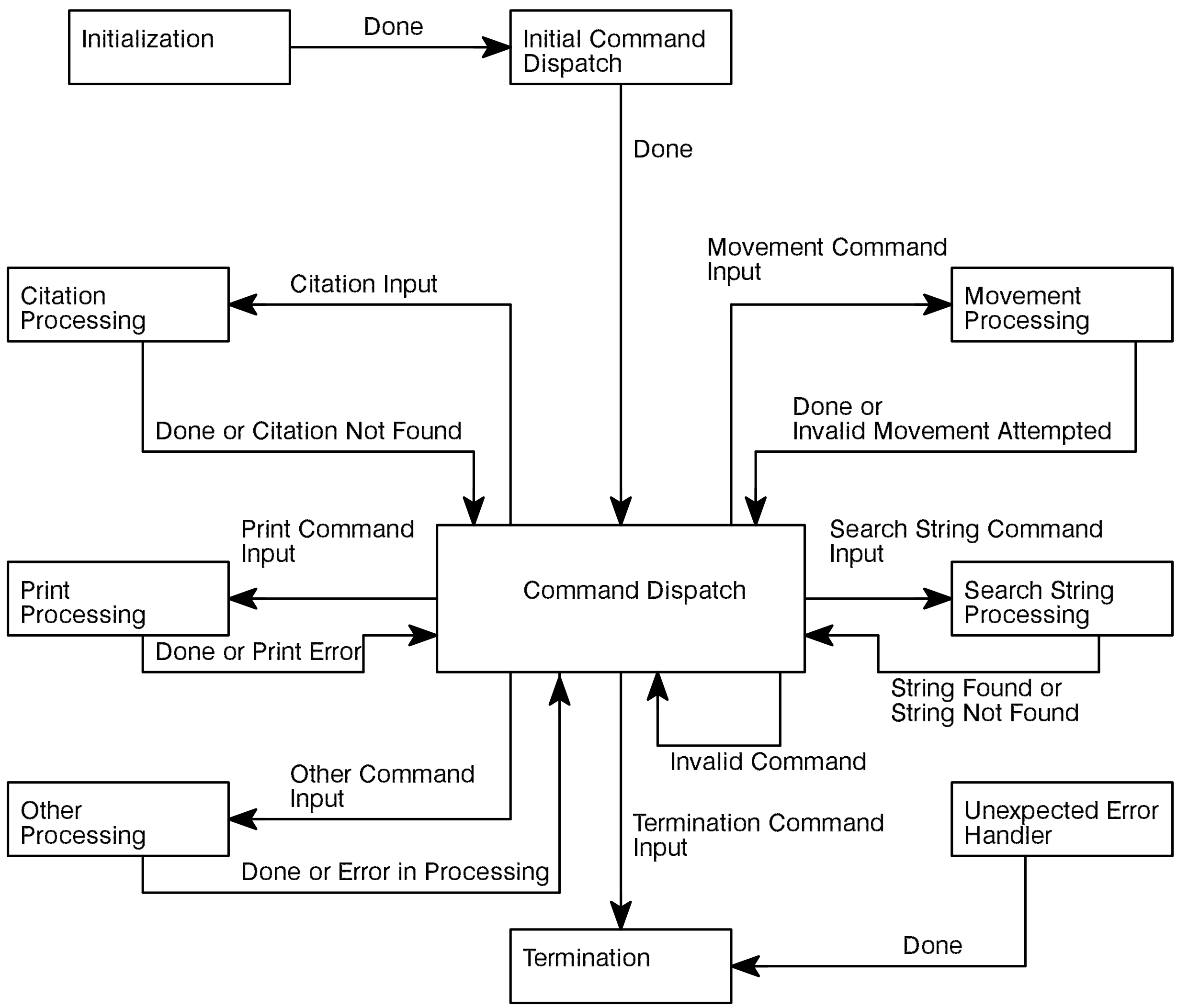
3.1.2.1. State Table

The following table itemizes the states of the LRM_Reader CSC:

State	*Meaning*	*Events*
Initialization	Startup initialization	Done Unexpected Error
Initial Command Dispatch	Command Line Argument is processed	Done Unexpected Error
Command Dispatch	User Input is Acquired and Processed	Citation Input Movement Command Input Print Command Input String Search Command Input Other Valid Command Input Termination Command Input Invalid Command Input Unexpected Error
Citation Processing	Locate and Display Citation	Done Citation Not Found Unexpected Error
Movement Processing	Move Between Screens and Citations	Done Invalid Movement Attempted Unexpected Error
Print Processing	Print Screen or Citation	Done Print Error Unexpected Error
String Search Processing	Search for String	String Found String Not Found Unexpected Error
Other Processing	Miscellaneous Commands Processed	Done Error in Processing Unexpected Error
Unexpected Error Handler	Recover from Unexpected Errors	Done
Termination	Close Down System and Exit to OS	

3.1.2.2. State Transition Diagram

The following State Transition Table shows these states and the events that cause transition between them. For each state except the Unexpected Error Handler, an Unexpected Error event causes transition to the Unexpected Error Handler; these events and transitions are not shown in order to reduce clutter in the diagram.



3.1.2.3. State/Associated CSC Table

The following table shows each state and the CSCs that are principally executing in that state.

State	*Associated CSCs*
Initialization	LRM_Reader
Initial Command Dispatch	LRM_Reader
Command Dispatch	Command Dispatcher, Screen Display Controller
Citation Processing	Citation_Handler, Primitive_Citation_Handler
Movement Processing	Citation_Handler, Primitive_Citation_Handler
Print Processing	Print_Log_Handler
String Search Processing	Citation_Handler, Primitive_Citation_Handler
Other Processing	Citation_Handler, Primitive_Citation_Handler

Unexpected Error Handler	LRM_Reader
Termination	LRM_Reader

3.1.3. Memory and processing time allocation

These allocations are presented for guidance only and are not binding.

3.1.3.1. LRM_Reader–based Task

CSC	*Memory*	*Time*
LRM_Reader Procedure	5%	1%
SYSDEP Package	0%	0%
Citation_Definition Package	5%	1%
Print_Log_Handler Package	5%	2%
Screen_Display_Controller Package	20%	80%
Citation_Handler Package	30%	6%
Primitive_Citation_Handler Package	35%	10%

3.1.3.2. Make_Cit–based Task

CSC	*Memory*	*Time*
Make_Cit Procedure	5%	5%
Console Package	15%	15%
SYSDEP Package	0%	0%
DAF_Handler Package	30%	40%
Output_File Package	50%	40%

3.1.3.3. Make_DAF–based Task

CSC	*Memory*	*Time*
Make_DAF Procedure	5%	5%
Console Package	15%	15%
SYSDEP Package	0%	0%
DAF_Handler Package	30%	45%
Input_File Package	50%	50%

3.2. CSCI design description

3.2.1. LRM_Reader Procedure

The LRM_Reader procedure is the mainline procedure for the Ada LRM Reader. Its purpose is to initialize the system, invoke the Command_Dispatcher, clean up when the Command_Dispatcher is finished, and trap any unexpected errors.

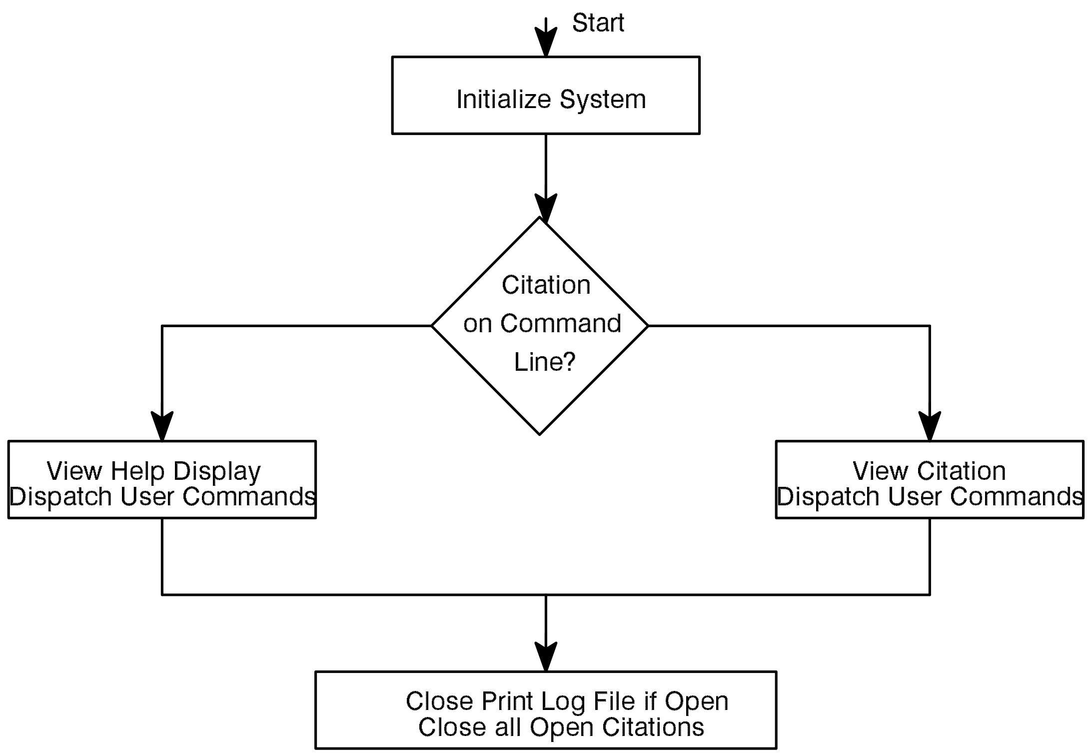
Mapping to Requirements

The LRM_Reader Procedure is the wrapper encapsulating the body of the Ada LRM Reader entity in the ERD in Section 3.3 of the SRS. It also meets the command line interface requirement in the SUM to start up with an optional citation specified on the command line.

Design

The OID in Section 3.1.1.1 of this SDD shows the data flow associated with the LRM_Reader Procedure.

The functional flow of the LRM_Reader Procedure is:



3.2.2. Make_DAF Procedure

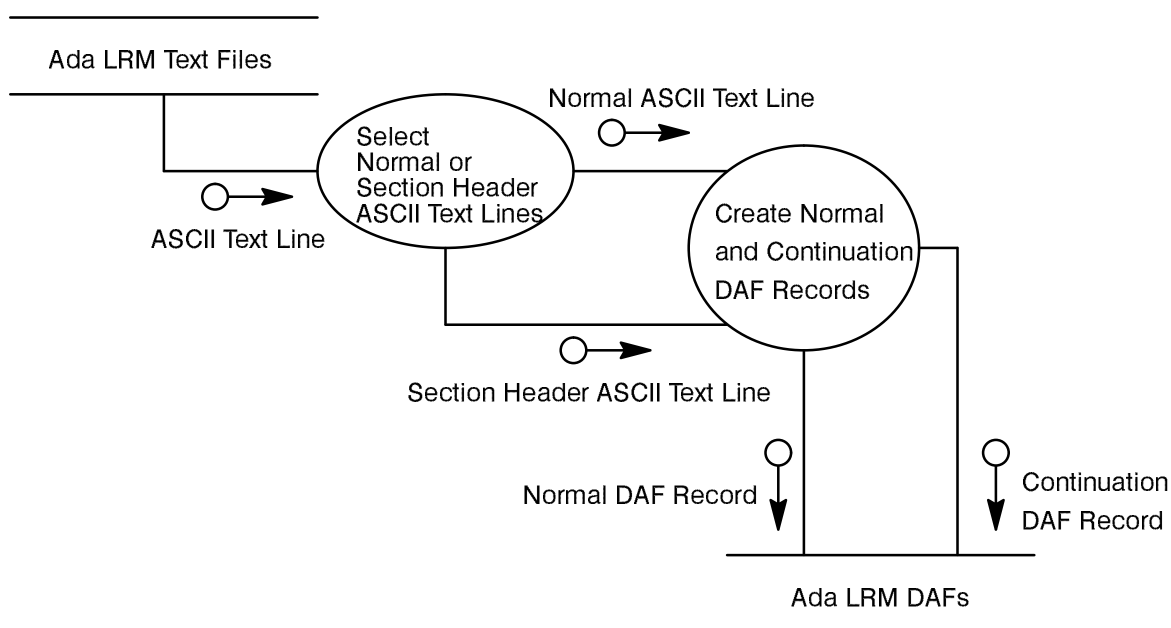
The Make_DAF Procedure is used to convert the ASCII text files which comprise the Ada LRM into Direct Access Files (DAFs). Placed into DAFs, the information in the Ada LRM can be located and accessed much more quickly than if it is stored as conventional ASCII text.

Mapping to Requirements

The Make_DAF Procedure addresses the requirements to contain the data in LRM Data Files in Sections 3.3 and 3.4 of the SRS. It also addresses the timing requirements in Section 3.6 of the SRS.

Design

The data flow of the Make_DAF Procedure is as follows:



The ASCII Text Lines from the Ada LRM Text Files are of two basic formats: (1) Section Header ASCII Text Lines, which begin with a ">" in the first column followed by a section number, and (2) Normal ASCII Text Lines, which begin with a character other than a ">". The DAF records do not contain these flags, but are structured as follows:

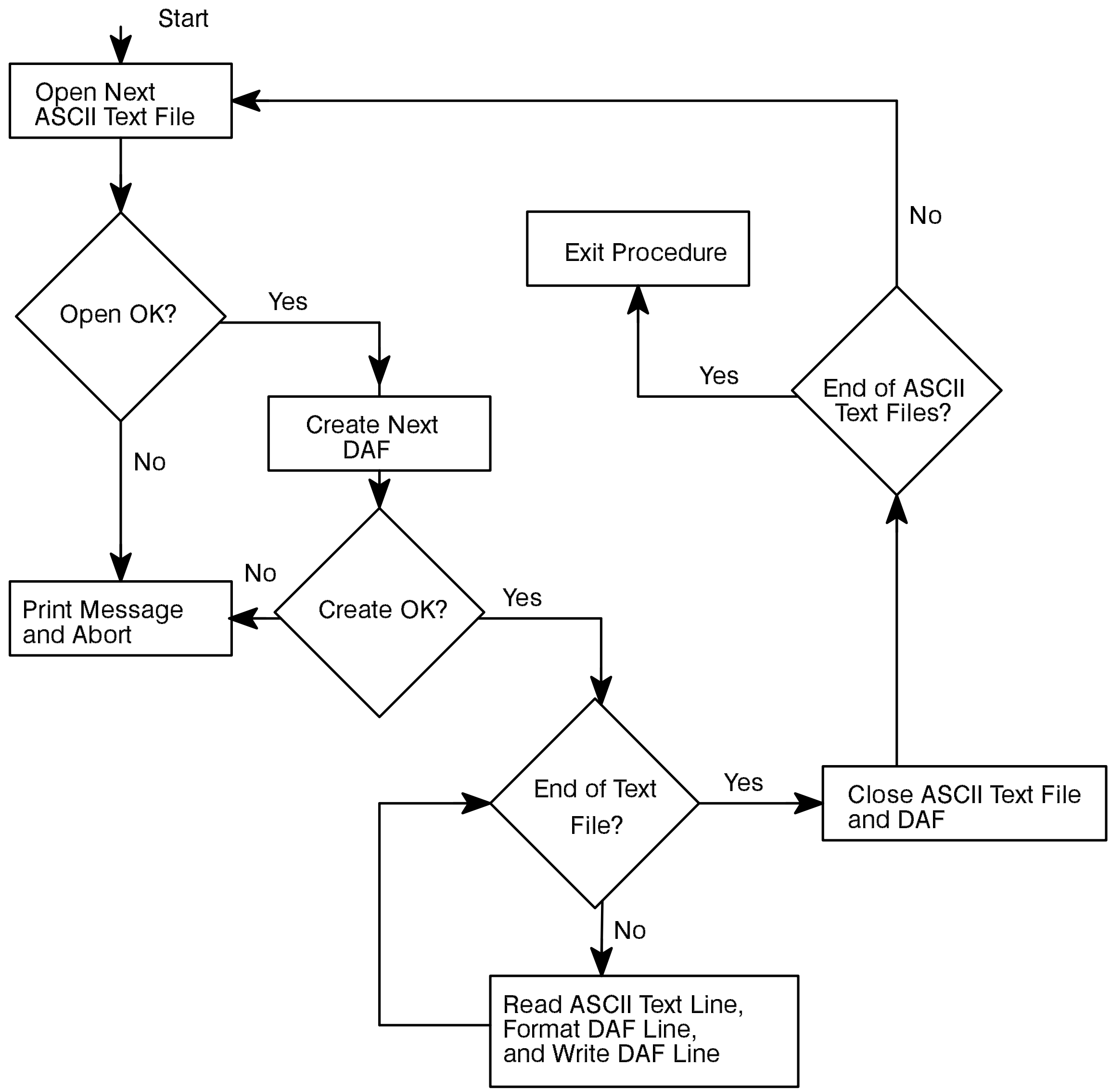
```
type LINE_TYPE is (NORMAL, SECTION, UNUSED);

type LINE is record
  Str      : STRING (1..SYSDEP.Screen_String_Length);
  Str_Last : NATURAL := 0;  -- index of last character in Str
  Kind     : LINE_TYPE := NORMAL;
end record;
```

Normal DAF Records differ from Continuation DAF Records in that the Str of Normal DAF Records begins with two leading spaces while the Str of Continuation DAF Records begin with a continuation mark, which is a vertical bar (|) followed by a space. The Str fields within the DAF Records are ready to be displayed on a VT100 screen without concern for exceeding the width of the screen.

The names of the ASCII text files of the Ada LRM are all of the form "chapNN.doc," and the names of the DAFs will be similar: "chapNN.daf." NN will take on the following values:

1. 01 to 14 – Ada LRM Chapters
2. aa to af – Ada LRM Appendices
3. co – Ada LRM Contents
4. in – Ada LRM Index
5. fo – Ada LRM Foreward
6. po – Ada LRM Postscript
7. he – Help screens for the HELP command
8. xx – Information screens for the ABOUT command

The functional flow of the Make_DAF Procedure is as follows:



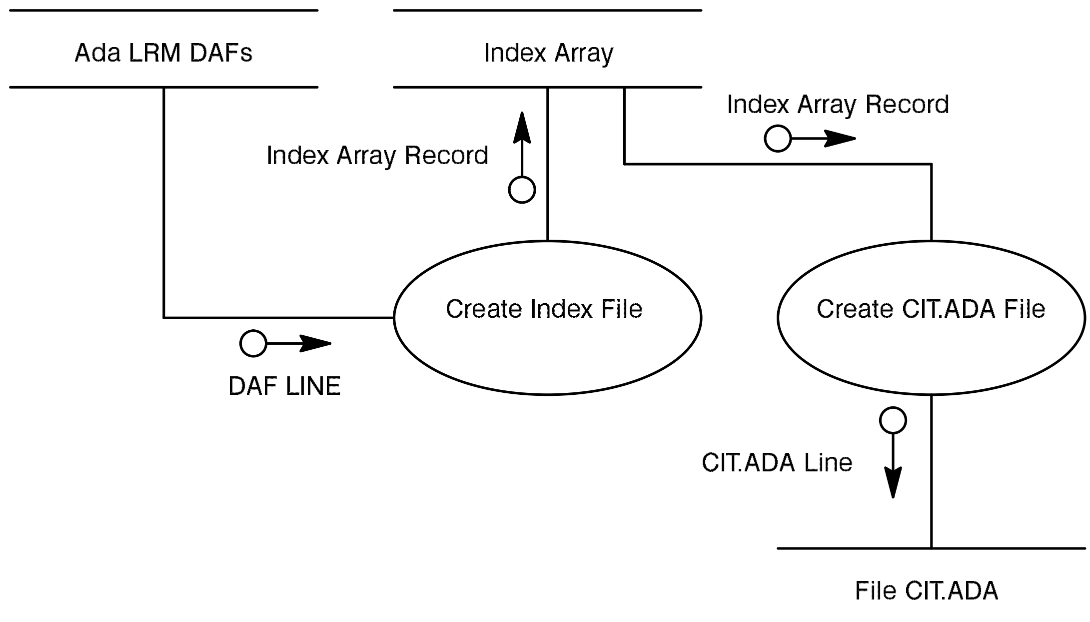
3.2.3. Make_Cit Procedure

The Make_CIT Procedure is used to create the CIT.ADA file from the *.daf files (created by the Make_DAF procedure). The CIT.ADA file contains two main elements: (1) the type CITATION_ID, which maps to each citation in each *.daf file, and (2) the CLV (Citation Location Vector), which identifies the associated *.daf file, the starting DAF Record number, and the ending DAF Record number for each citation specified in type CITATION_ID. CIT.ADA, therefore, is a hard-coded data collection which provides a very fast way to index into the *.daf files to locate and load a citation of interest.

Mapping to Requirements

The Make_Cit Procedure addresses the timing requirements in Section 3.6 of the SRS by creating the CIT.ADA file, which will serve as a memory-resident index to the citations.

The data flow of the Make_Cit Procedure is as follows:



The DAF records are structured as follows:

```
type LINE_TYPE is (NORMAL, SECTION, UNUSED);

type LINE is record
  Str      : STRING (1..SYSDEP.Screen_String_Length);
  Str_Last : NATURAL := 0;   -- index of last character in Str
  Kind     : LINE_TYPE := NORMAL;
end record;
```

The DAF LINEs whose Kind are SECTION trigger the new entries in the Index Array. The Index Array is simply an array of records that contain the following information:

```
subtype FILE_ID_STRING is STRING (1..2);

type CITATION_RECORD is record
  Citation_ID      : STRING (1..20);
  Citation_ID_Last : NATURAL;
  File_ID          : FILE_ID_STRING;
  Start            : DAF_Handler.LINE_NUMBER;
  Stop             : DAF_Handler.LINE_NUMBER;
end record;
```

where Citation_ID contains strings like ”C1P1” and ”C1P2” to identify citations (Chapter 1, Sections 1 and 2 in this case) and to ultimately form the CITATION_ID enumeration values in the CIT.ADA file.

The CITATION_ID type in the CIT.ADA file is an enumeration type which contains these citation IDs. For example:

```
type CITATION_ID is (C1P1, C1P2);
```

would the type declaration of the type CITATION_ID if C1P1 and C1P2 were the only citation IDs. The CLV array is an array of CITATION_LOCATION records, and, for our simple example, would resemble the following if C1P1 and C1P2 were contained in the file ”chap01.daf”:

```
type CITATION_LOCATION is record
  Chapter : STRING(1..2);
  Start   : DAF_Handler.LINE_NUMBER;
  Stop    : DAF_Handler.LINE_NUMBER;
end record;

type CITATION_LOCATION_VECTOR is array (CITATION_ID) of CITATION_LOCATION;

CLV : constant CITATION_LOCATION_VECTOR := (
  C1P1 => ("01", 1, 23),
  C1P2 => ("01", 24, 43)
);
```

The formats for the file names were discussed in the section on the preliminary design of the Make_DAF Procedure.

In addition to the conventional citation IDs (C1 to CF), the type CITATION_ID includes the following entries to provide mappings to the Table of Contents, Index, Foreward, and Postscript parts of the Ada LRM and the on-line help and about ”citations”:

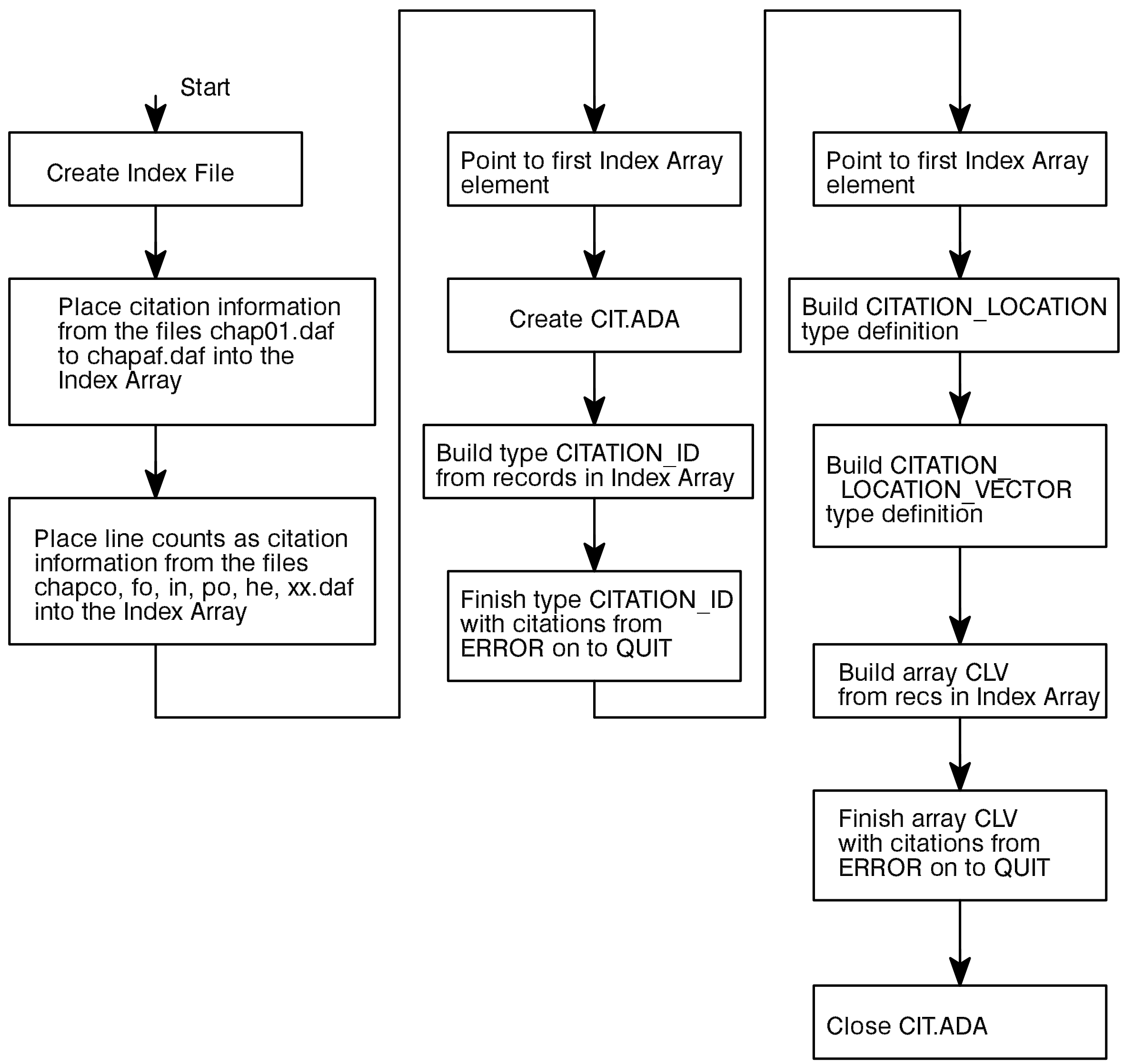
Citation	*Associated File*	*CLV Chapter*
CONTENTS	chapco.daf	co
FOREWARD	chapfo.daf	fo
INDEX	chapin.daf	in
POSTSCRIPT	chappo.daf	po
HELP	chaphe.daf	he
ABOUT	chapxx.daf	xx

Finally, the type CITATION_ID includes the following entries to provide mappings to the user commands (processed within the Command_Dispatcher package and documented extensively in the SUM), since citations and commands are both processed as commands:

Citation ID	*Associated Command*
ERROR	Invalid command
N	N (next screen)
P	P (previous screen)
USER_INPUT	input from the user
PRINT	PRINT (print current citation)
PS	PS (print current screen)
NEXT	NEXT (next citation)

PREVIOUS	PREVIOUS (previous citation)
PAUSE	PAUSE (delay further processing)
PUSH	PUSH (push current citation onto location stack and select new citation)
POP	POP (return to last citation PUSHed)
SEARCH_FIRST	/ (search from beginning of current citation for string)
SEARCH_NEXT	// (search from current position for string)
REFRESH	REFRESH (redisplay the current screen)
QUIT	QUIT (terminate the program)

The functional flow of the Make_Cit Procedure is as follows:



3.2.4. SYSDEP Package

The SYSDEP Package is a package which serves as a collection for the system dependency information. It serves to isolate all system dependencies in one single location so that future adaptation of the Ada LRM Reader to different environments will be expedited. Writing the Ada LRM Reader in Ada is a first step toward generating a system which ports to a variety of platforms, and during development the Ada LRM Reader will be compiled on several different systems to test its portability. However, there are some aspects of the design of such a system which Ada cannot by itself support the portability of the software. Directory names are a good example, which may take the form of "/major_dir/sub_dir" on one system (UNIX), "drive:\major_dir\sub_dir" on another system (MSDOS), or "disk:[major_dir.sub_dir]" on yet another system (VMS). The SYSDEP Package (short for SYStem DEPendency Package) acts as a container to limit these dependencies to only one CSC of the CSCI.

Mapping to Requirements

The SYSDEP Package meets requirements 3.5 and 3.5.1 in the SRS.

Design

The SYSDEP Package will contain only constants the the associated type definitions. The constants will be as reusable as possible:

1. Constant STRING objects will be explicitly unconstrained, allowing a greater ease of modification. For example, the form

   ```
   LRM_Files_Directory : constant STRING := "/reader/ada_lrm";
   ```

   will be used so that changing this string to some other value will require only editing the string.

2. Integer-like constants will be universal integers so they may map to any type. The following is a universal integer:

   ```
   Max_String_Length : constant := 110; -- chars
   ```

The SYSDEP Package contains the following constants:

1. LRM_Files_Directory – This STRING is the name of the directory containing the LRM DAFs ("chapNN.daf") of a form that a file name may simply be appended to it.

2. Program_Name – This STRING is the name of the Ada LRM Reader with a version number.

3. Print_File_Name – This STRING is the name of the file created in the user's current directory when the PRINT and PS commands are executed.

4. Citation_Stack_Depth – This universal integer is the size of the location stack within the LRM_Reader; it is equal to the maximum number of citations which may be PUSHed (recommended value: 20).

5. Screen_Width, Text_Line_Count, Command_Line_Number, Error_Message_Line_Number, Screen_String_Length, and Search_Pointer_Column – These universal integers define various attributes of the VT100 display screen. In addition, Screen_String_Length is an attribute of the Str field a DAF record.

6. Max_String_Length – This universal integer is the maximum length of a string from the console or a text data file. Examination has shown that the Ada LRM ASCII text files have lines as long as 102 characters, so this value should be a little larger than 102 in order to assure that buffers are not overflowed (recommended value: 110).

7. Max_Number_of_Citations – This universal integer is the maximum number of citations within a single DAF. Examination has shown that the Ada LRM ASCII text files have a maximum of 34 cita-

tions in a single file, and these files will map directly to the DAFs, so this value should be a little larger than 34 in order to assure that buffers are not overflowed (recommended value: 40).

8. Total_Number_of_Citations – This universal integer is the total number of citations in all the *.daf files. Examination has shown that the *.daf files have 214 citations, so this value should be a little larger than 214 in order to assure that buffers are not overflowed (recommended value: 230).

9. Max_Number_of_Screens – This universal integer is the maximum number of screens which may be allocated for a single citation. Examination has shown that the Ada LRM ASCII text files have a maximum of 5,800 lines in a single file (the index), so, rounding this up to 6,000 for safety, this number should be no less than 6000 lines/22 lines per screen (recommended value: 273).

10. Full_Copyright_Notice and Intro_Copyright_Notice – These STRING constants are copyright notices which are (1) simply embedded in the code (that is the only purpose of the Full_Copyright_Notice aside from acting as a copyright in the SYSDEP package itself) or (2) displayed to the user when a task (such as the LRM_Reader, Make_Cit, or Make_DAF) starts.

3.2.5. Citation_Definition Package

The Citation_Definition Package was discussed in some detail earlier when the Make_Cit Procedure was discussed. See the section on Make_Cit (Section 3.2.3.) for further information.

Mapping to Requirements

The Citation_Definition Package was created to address the timing requirements in Section 3.6 of the SRS. It provides a program–resident index into the *.daf files for each citation, greatly speeding program startup and routine operation by not requiring this information to be accessed from disk on a periodic basis.

Design

The Citation_Definition Package is created by the Make_Cit Procedure. It consists of three basic components:

1. Type CITATION_ID, an enumerated type identifying each citation in the Ada LRM, the table of contents, the index, the foreward, the postscript, the HELP citation for online documentation in the use of the Ada LRM Reader, the ABOUT citation, the ERROR condition, the USER_INPUT condition, and the various commands (N, P, etc.) to which the Command_Dispatcher responds. Type CITATION_ID is of the general form:

```
type CITATION_ID is (
  C1, -- chapter, section, subsection references
  C1P1,
  C1P1P1,
  -- detail omitted
  CA, -- appendices
  CB,
  CC,
  CD,
  CE,
  CF,
  CONTENTS,  -- special parts of the Ada LRM
  FOREWARD,
  INDEX,
  POSTSCRIPT,
```

```
    HELP, -- HELP and ABOUT citations
    ABOUT,
    ERROR, -- ERROR condition
    N, P, -- commands, including USER_INPUT
    -- detail omitted
    QUIT
  );
```

2. Types CITATION_LOCATION and CITATION_LOCATION_VECTOR, which are used to define the CLV array. These types are:

```
type CITATION_LOCATION is record
  Chapter : STRING(1..2);
  Start   : DAF_Handler.LINE_NUMBER;
  Stop    : DAF_Handler.LINE_NUMBER;
end record;

type CITATION_LOCATION_VECTOR is array (CITATION) of CITATION_LOCATION;
```

3. The CLV array, which identifies the *.daf file and the starting and ending record numbers of that citation within the indicated *.daf file. The general format of the CLV array is:

```
CLV : constant CITATION_LOCATION_VECTOR := (
  C1 => ("01", 23, 46), -- chapters, sections, subsections
  C1P1 => ("01", 47, 55),
  C1P1P1 => ("01", 56, 120),
  -- detail omitted
  CA => ("aa", 24, 506), -- appendices
  CB => ("ab", 23, 175),
  -- detail omitted
  CONTENTS => ("co", 1, 284),  -- special parts of the Ada LRM
  FOREWARD => ("fo", 1, 81),
  INDEX => ("in", 1, 5833),
  POSTSCRIPT => ("po", 1, 90),
  HELP => ("he", 1, 38), -- HELP and ABOUT citations
  ABOUT => ("xx", 1, 12),
  ERROR => ("  ", 1, 1), -- the commands start here
  N     => ("  ", 1, 1),
  -- detail omitted
  QUIT  => ("  ", 1, 1)
);
```

The Chapter field in the CLV array was designed to easily allow the creation of the *.doc and *.daf file names associated with a given citation. Note that the *.doc and *.daf file names associated with a citation are easily created using the CLV array: DOC_File_Name = ”chap” & CLV(Citation).Chapter & ”.doc”

3.2.6. DAF_Handler Package

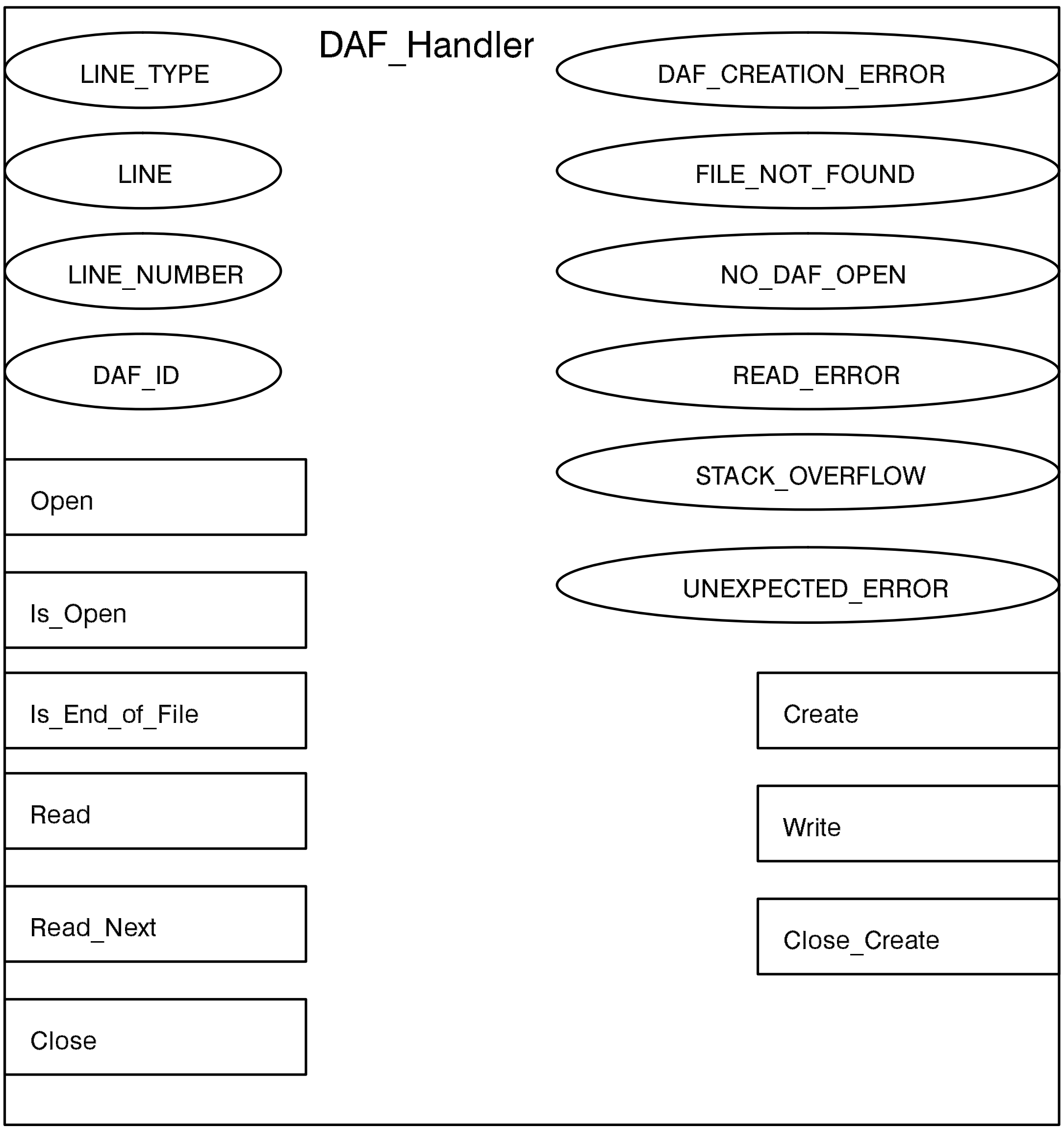
The DAF_Handler Package implements a passive object which provides methods to create and manipulate DAFs (Direct Access Files).

Mapping to Requirements

DAFs are employed primarily to address the timing requirements in Section 3.6 of the SRS. DAFs implement the LRM Data Files and the Ada LRM Reader Support Files identified in Section 3.4 of the SRS.

Design

The DAF_Handler Package presents the following sets of methods, types, data, and exceptions in its interface:



In this OID symbol, the short ovals represent data types, the long ovals represent exceptions, and the small rectangles represent subprograms. Details on the data types are presented in this SDD, Section 4.3.

CSC Specification

The following is the specification of the DAF_Handler Package:

```
with SYSDEP;
package DAF_Handler is
-- Handler for Direct Access Files (DAFs)

  -- Types of LINEs (records) in DAFs
  type LINE_TYPE is (NORMAL, SECTION, UNUSED);

  -- The LINE is the record of a DAF
  type LINE is record
    Str      : STRING (1..SYSDEP.Screen_String_Length);
    Str_Last : NATURAL := 0; -- index of last char in Str
    Kind     : LINE_TYPE := NORMAL;
  end record;

  subtype LINE_NUMBER is NATURAL range 1..NATURAL´LAST;

  subtype DAF_ID is NATURAL range 0..SYSDEP.Citation_Stack_Depth;

-- Exceptions
  DAF_CREATION_ERROR : exception;
  FILE_NOT_FOUND     : exception;
  NO_DAF_OPEN        : exception;
  READ_ERROR         : exception;
  WRITE_ERROR        : exception;
  STACK_OVERFLOW     : exception;
  UNEXPECTED_ERROR   : exception;

-- Subprograms to create a DAF file
  procedure Create (File_Name : in STRING);
  -- Create a DAF file

  procedure Write (Item : in STRING);
  -- Write a string to a DAF file

  procedure Close_Create;
  -- Close a DAF file

-- Subprograms to read DAF files
  function Open (File_Name : in STRING) return DAF_ID;
  -- Open an existing DAF file

  function Is_Open (ID : in DAF_ID) return BOOLEAN;
  -- Determine if the DAF file is currently open

  function Is_End_of_File (ID : in DAF_ID) return BOOLEAN;
  -- Determine if the end of a DAF file has been reached

  function Read (ID   : in DAF_ID;
                 Lnum : in LINE_NUMBER) return LINE;
  -- Read a specified line from a DAF file
```

```
  function Read_Next (ID : in DAF_ID) return LINE;
  -- Read the next line from a DAF file

  procedure Close (ID : in DAF_ID);
  -- Close a DAF file

end DAF_Handler;
```

Required Program Units

The following program units must be withed by the body of DAF_Handler:

```
Direct_IO
```

The following program units are needed by the body DAF_Handler but are already withed by the specification:

```
SYSDEP
```

Internal Global Code, Types, and Objects

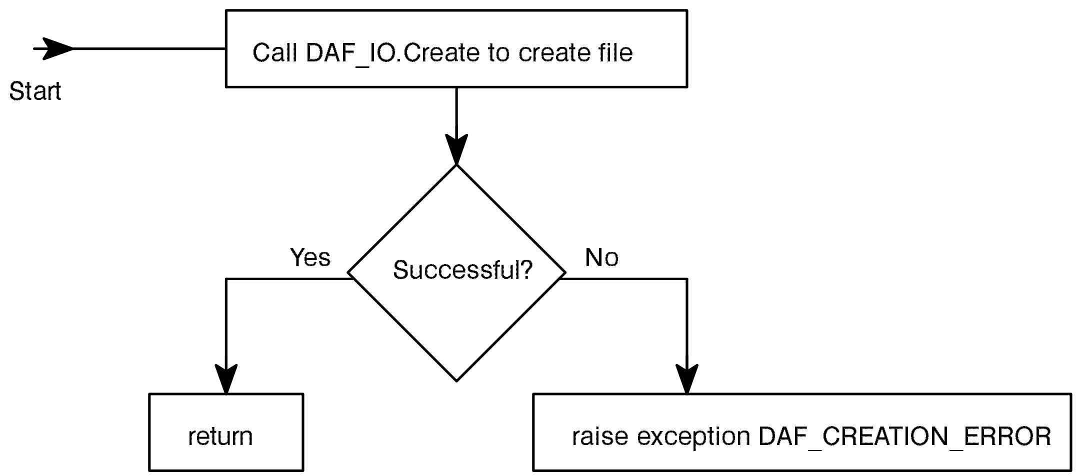
The following global code, types, and objects reside within the DAF_Handler.

```
package DAF_IO is new Direct_IO (LINE);

type FILE_ID_STACK is array (1..DAF_ID´LAST) of DAF_IO.FILE_TYPE;
Stack : FILE_ID_STACK;
-- Stack of FILE_TYPE objects associated with DAF_IDs

type USE_FLAG is (UNAVAILABLE, AVAILABLE);
-- Flag to mark a file ID as available or not

type FILE_USE_STACK is array (1..DAF_ID´LAST) of USE_FLAG;
Use_Stack : FILE_USE_STACK := (others => AVAILABLE);
-- Stack of USE_FLAGs to mark DAF_IDs as available or not

Create_File_ID : DAF_IO.FILE_TYPE;
-- Global file type object used for an output DAF
```

3.2.6.1. Create Procedure

The Create procedure works with the Create_File_ID data and the DAF_IO Package.

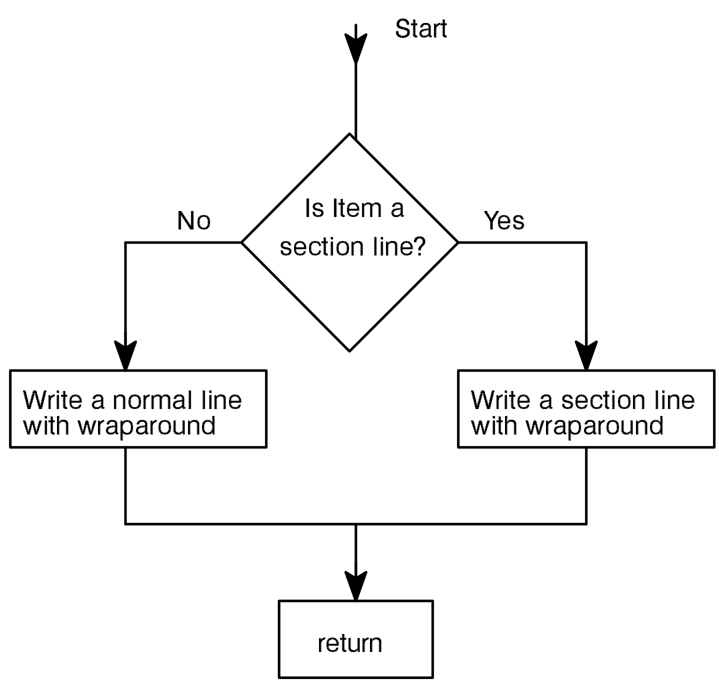
The algorithm for this subprogram is:



3.2.6.2. Write Procedure

The Write Procedure works with the Create_File_ID data and the DAF_IO Package.

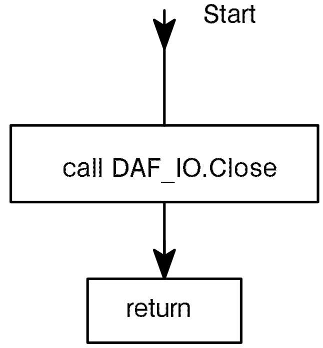
The algorithm for this subprogram is:



3.2.6.3. Close_Create Procedure

The Close_Create Procedure works with the Create_File_ID data and the DAF_IO Package.

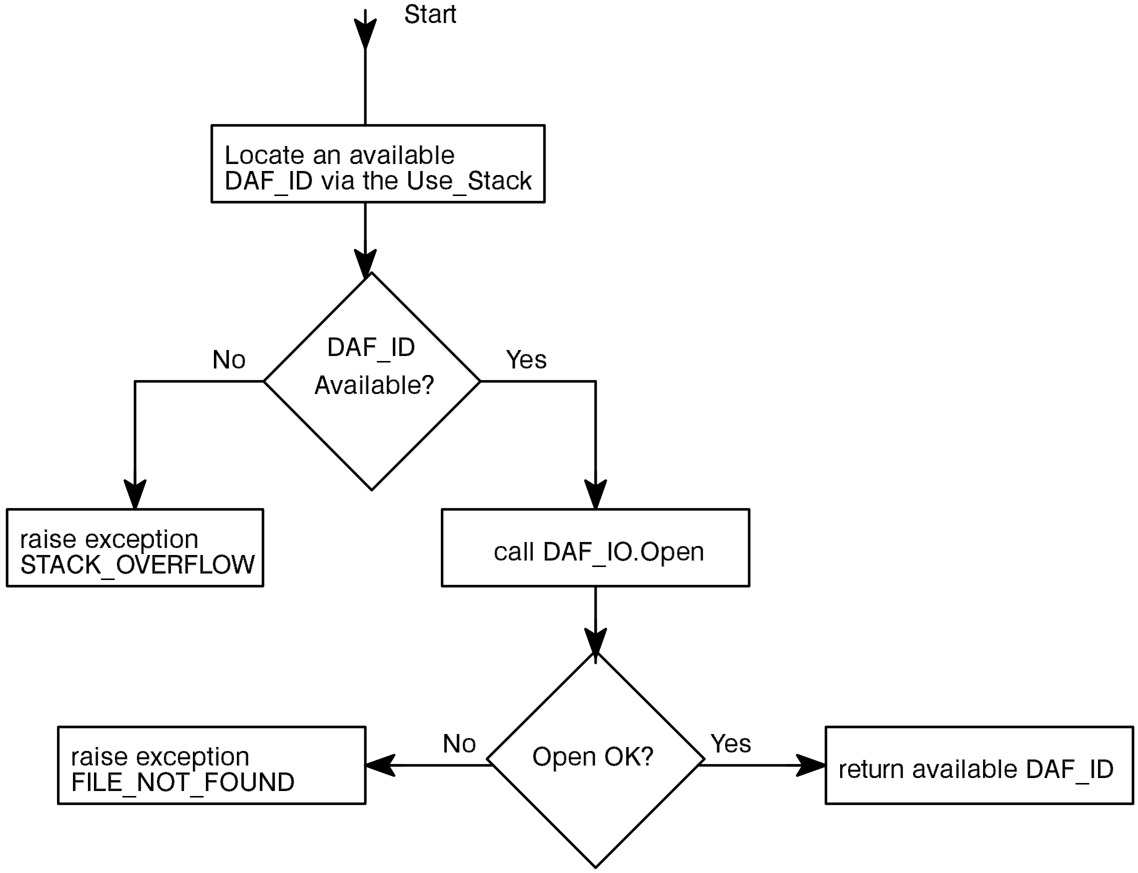
The algorithm for this subprogram is:



3.2.6.4. Open Function

The Open Function works with the DAF_IO Package and the Use_Stack and Stack vectors.

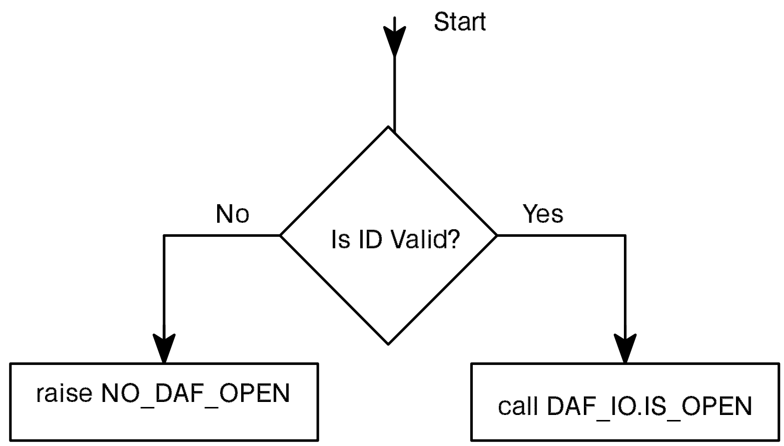
The algorithm for this subprogram is:



3.2.6.5. Is_Open Function

The Is_Open function works with DAF_IO and the Stack vector.

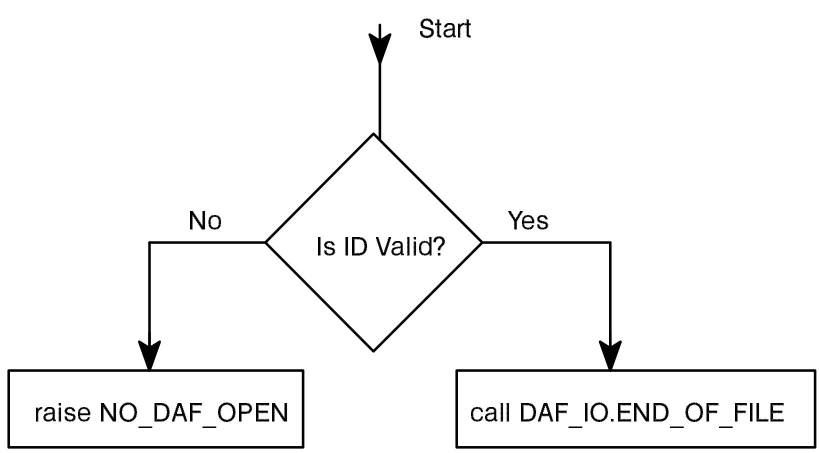
The algorithm for this subprogram is:



3.2.6.6. Is_End_of_File Function

The Is_End_of_File function works with DAF_IO and the Stack vector.

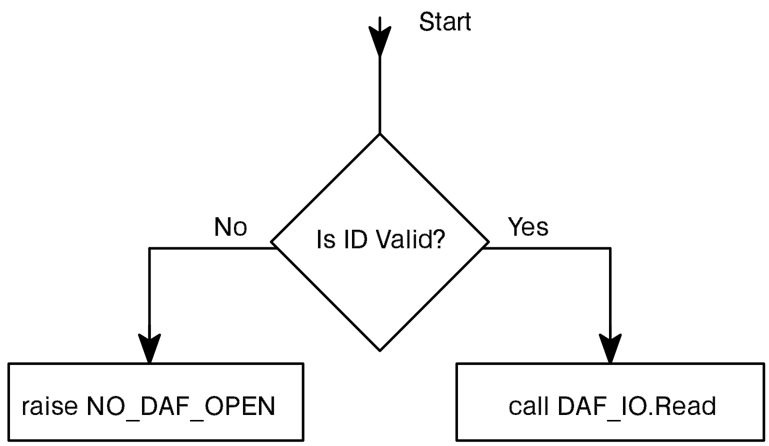
The algorithm for this subprogram is:



3.2.6.7. Read Function

The Read function works with DAF_IO and the Stack vector.

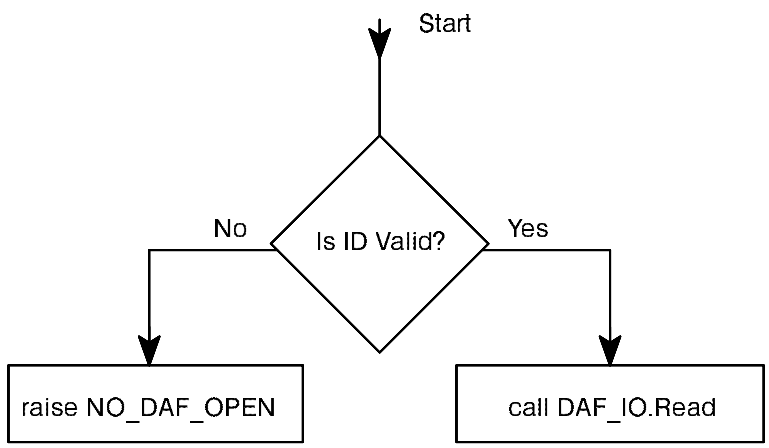
The algorithm for this subprogram is:



3.2.6.8. Read_Next Function

The Read_Next function works with DAF_IO and the Stack vector.

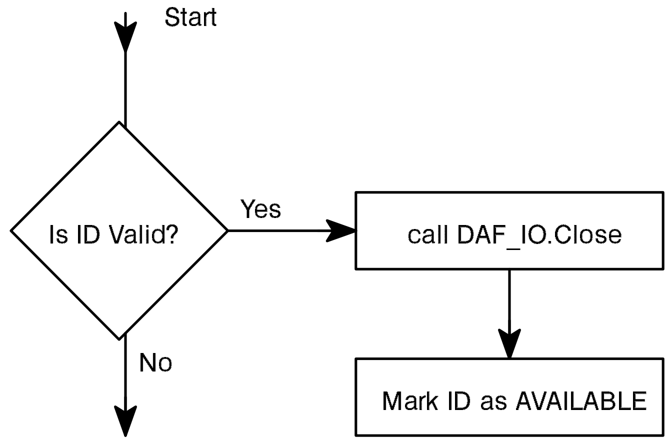
The algorithm for this subprogram is:



3.2.6.9. Close Procedure

The Close function works with DAF_IO, the Use_Stack vector, and the Stack vector.

The algorithm for this subprogram is:



3.2.7. Command_Dispatcher Package

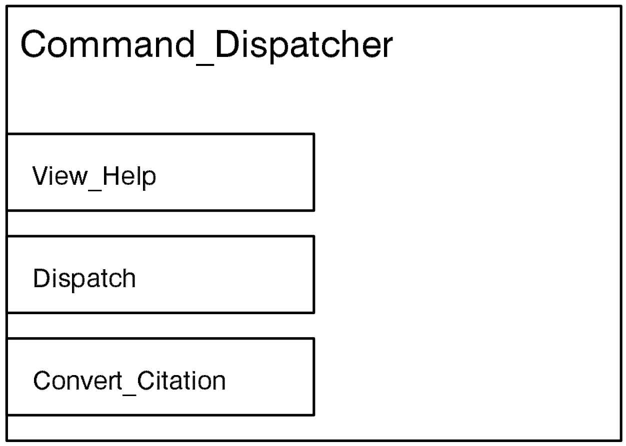
The Command_Dispatcher Package implements a passive object which provides methods to start up the first citation display, handle events generated by the user (commands issued at the keyboard), and dispatch messages to the appropriate methods associated with the objects which will handle these events.

Mapping to Requirements

This CSC implements the user command processing and dispatching required for all capabilities identified in Section 3.2 (including subsections 3.2.1 to 3.2.5) of the SRS. It also meets the design constraint in Section 3.7 of the SRS which stipulates that the Ada LRM Reader be event-driven by synchronous events.

Design

The Command_Dispatcher Package presents the following sets of methods, types, data, and exceptions in its interface:



In this OID symbol, the small rectangles represent subprograms.

CSC Specification

```
-- ***********************************************************************
-- ON-LINE Ada LANGUAGE REFERENCE MANUAL
-- by Richard Conn
with Citation_Definition;
package Command_Dispatcher is

  function Convert_Citation (CitS : in STRING)
    return Citation_Definition.CITATION_ID;
  -- Convert the indicated string ("n.n.n" or "keyword") to CITATION_ID

  procedure View_Help;
  -- View help citation and then Dispatch (Citation_Definition.USER_INPUT)

  procedure Dispatch (Current_Citation :
                      in Citation_Definition.CITATION_ID);
  -- Dispatch Current_Citation as first command and continue with
  -- USER_INPUT until command is QUIT

end Command_Dispatcher;
```

Required Program Units

The following program units are withed into the specification:

```
Citation_Definition
```

The following program units are withed into the body:

```
SYSDEP
Citation_Handler
```

```
DAF_Handler
Primitive_Citation_Handler
Print_Log_Handler
Screen_Display_Controller
Console
```

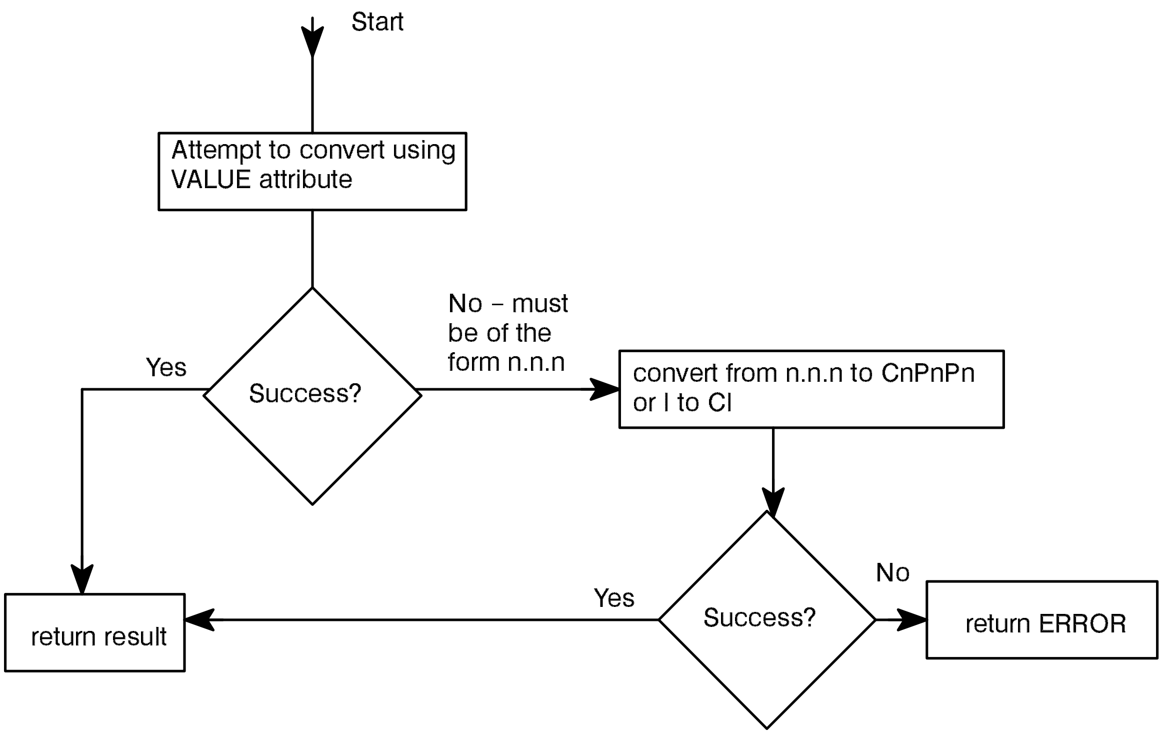
Internal Global Code, Types, and Objects

The following is the global data internal to the body of Command_Dispatcher:

```
New_Command          : STRING (1..SYSDEP.Max_String_Length);
New_Command_Length   : NATURAL;

Citation_to_Process : Citation_Definition.CITATION_ID;
```

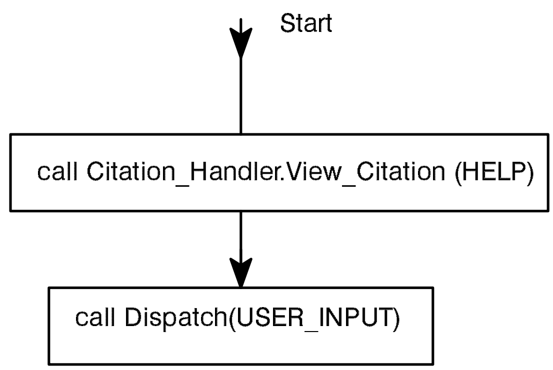
3.2.7.1. Convert_Citation Function

The algorithm for this subprogram is:



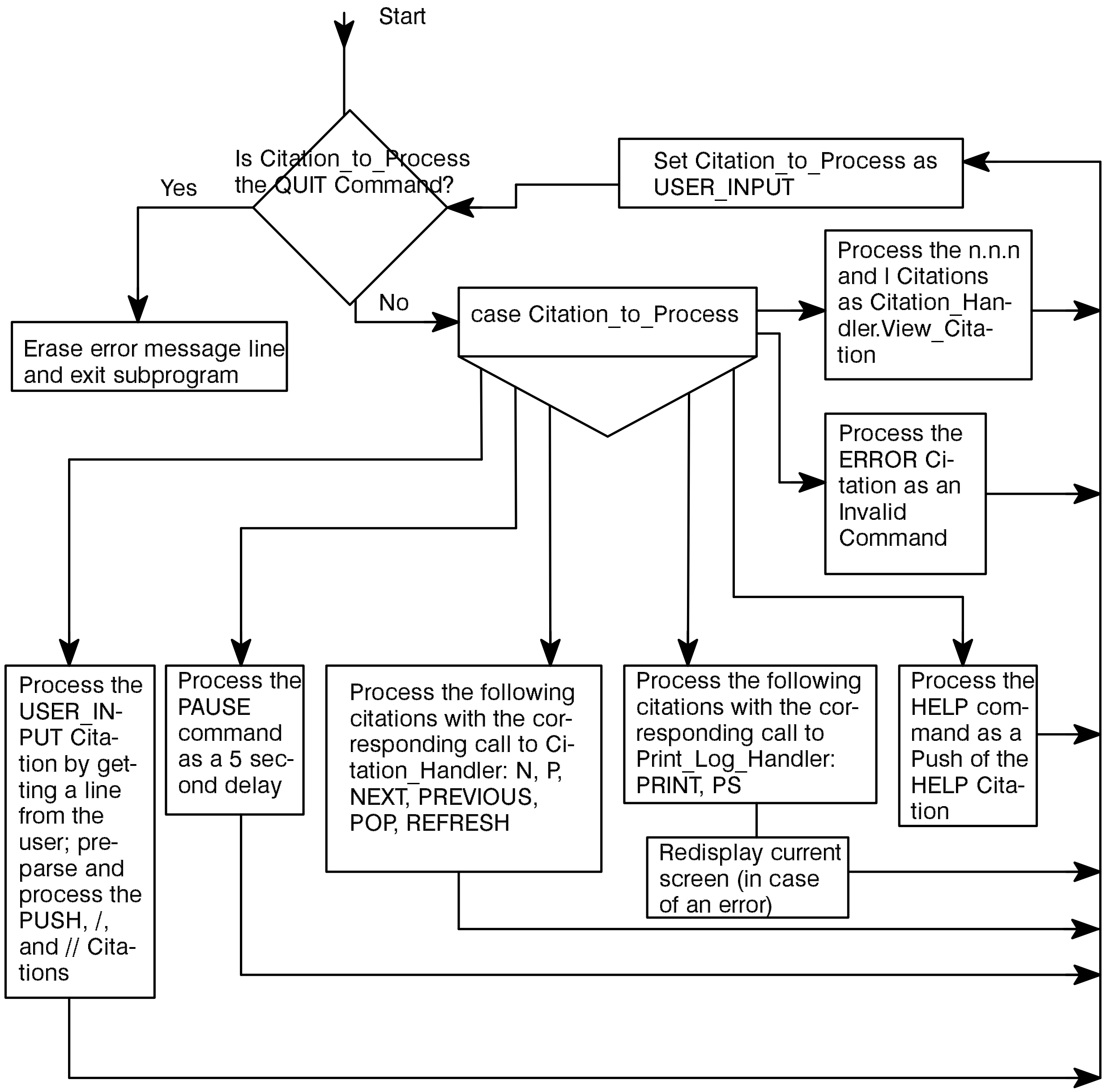
3.2.7.2. View_Help Procedure

The algorithm for this subprogram is:



3.2.7.3. Dispatch Procedure

The algorithm for this subprogram is:



3.2.8. Screen_Display_Controller Package

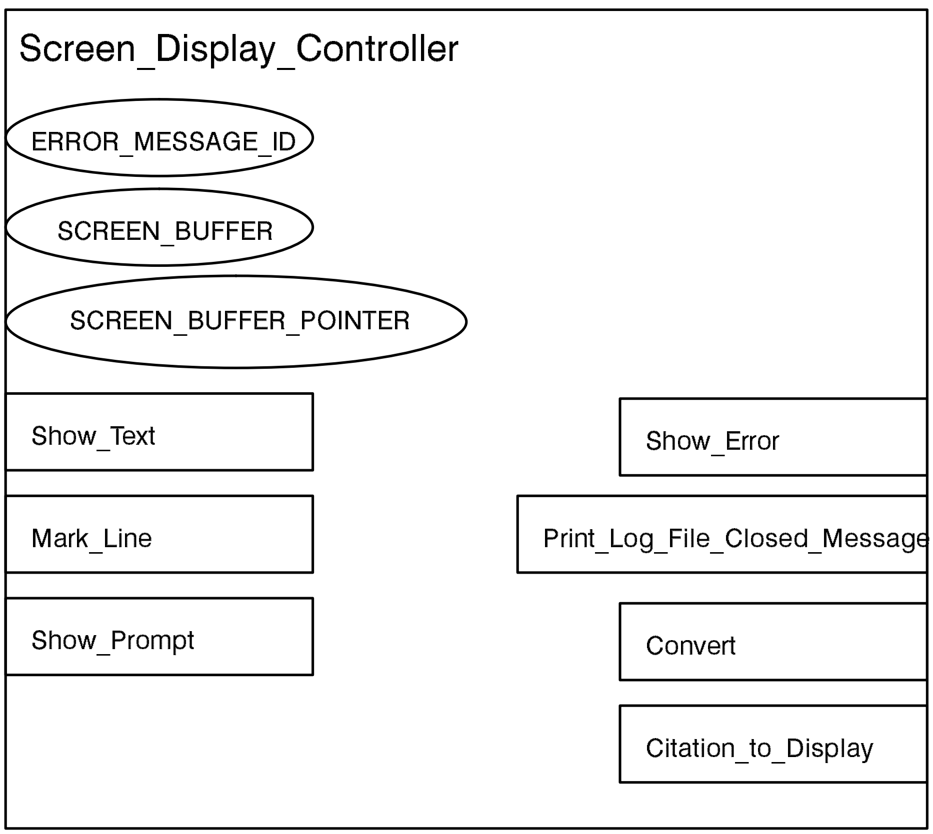
The Screen_Display_Controller package controls all output to the VT100 display. The Screen_Display_Controller serves as the single source for error messages.

Mapping to Requirements

The Screen_Display_Controller addresses the VT100 external interfaces requirement in Section 3.1 of the SRS. The VT100 Display also appears in Sections 3.3 and 3.4 of the SRS. VT100 adaptation issues are discussed in Section 3.5 of the SRS. The Screen_Display_Controller implements a large part of the user interface required in Section 3.8 of the SRS.

Design

The Screen_Display_Controller Package presents the following sets of methods, types, data, and exceptions in its interface:



In this OID symbol, the ovals represent data types and the rectangles represent subprograms.

CSC Specification

```
-- ***********************************************************************
-- ON-LINE Ada LANGUAGE REFERENCE MANUAL
-- by Richard Conn
with SYSDEP;
with Citation_Definition;
with System; -- standard Ada environment
package Screen_Display_Controller is

  type ERROR_MESSAGE_ID is (INVALID_COMMAND,
                            CANNOT_ADVANCE, CANNOT_BACK,
                            STACK_EMPTY, STACK_FULL,
                            PRINT_LOG,
                            TOO_MANY_SCREENS,
                            SEARCH_STRING,
                            DAF_NOT_FOUND,
                            INTERNAL_DAF_NDFO_ERROR,
                            INTERNAL_DAF_RE_ERROR,
                            INTERNAL_DAF_SO_ERROR,
                            INTERNAL_DAF_UE_ERROR,
                            UNEXPECTED_ERROR);
  -- Kinds of error messages which may be displayed

  type SCREEN_BUFFER is array (NATURAL´(1)..SYSDEP.Text_Line_Count) of
    DAF_Handler.LINE;
  -- Lines associated with a screen

  type SCREEN_BUFFER_POINTER is access SCREEN_BUFFER;
  -- Pointer to a screen buffer so the full buffer does not have to be
  -- passed

  procedure Show_Text;
  -- Clear screen and display the text area

  procedure Mark_Line (Number : in NATURAL);
  -- Place a mark on the indicated line

  procedure Show_Prompt;
  -- Display prompt on command line; if Search_String is null, do not
  -- display it; clear error message if one is present after one call
  -- to Show_Prompt

  procedure Show_Error (Item : in ERROR_MESSAGE_ID);
  -- Display error message

  procedure Print_Log_File_Closed_Message;
  -- Print the message that the indicated print log file is closed

  function Convert (SB_Address : in System.ADDRESS) return
    SCREEN_BUFFER_POINTER;
  -- Given the address of a screen buffer object, return a pointer to it

  function Citation_to_Display (CitX : in Citation_Definition.CITATION_ID)
      return STRING;
```

```
  -- Given a citation ID, return a string of the form "n.n.n" or "keyword"

end Screen_Display_Controller;
```

Required Program Units

The following program units are withed into the specification:

```
SYSDEP
Citation_Definition
System
```

The following program units are withed into the body:

```
DAF_Handler
Primitive_Citation_Handler
Console
Unchecked_Conversion
```

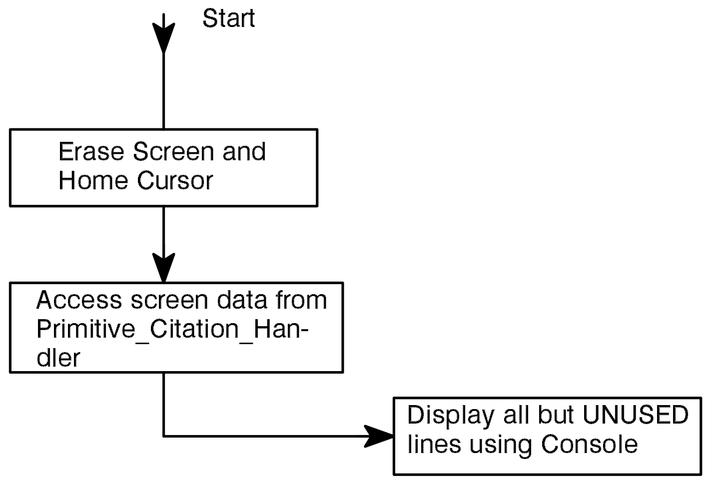
Internal Global Code, Types, and Objects

The following are in the body:

```
Error_Message_Display_Counter : NATURAL := 0;
  -- counts the number of times since the last error message was displayed;
used
  -- to clear the error message line

Search_String_Limit : constant := 12;
  -- maximum number of characters in the search string to be displayed on
the
  -- command prompt line

subtype MSTRING is STRING (1..52);
  -- type of STRING used to store the error messages, based on the length
of the
  -- longest error message

Messages : constant array (ERROR_MESSAGE_ID) of MSTRING := ( -- detail
omitted
  );
  -- the text of all error messages to be displayed
```

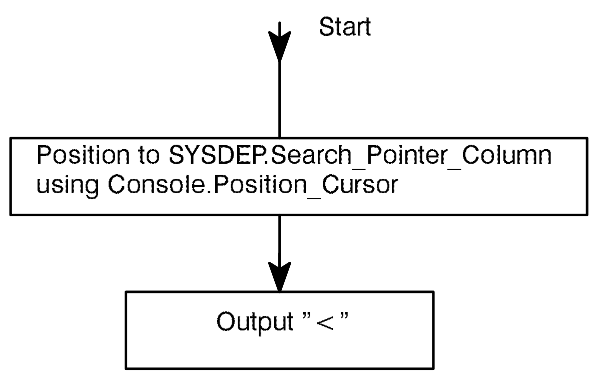
3.2.8.1. Show_Text Procedure

The algorithm for this subprogram is:



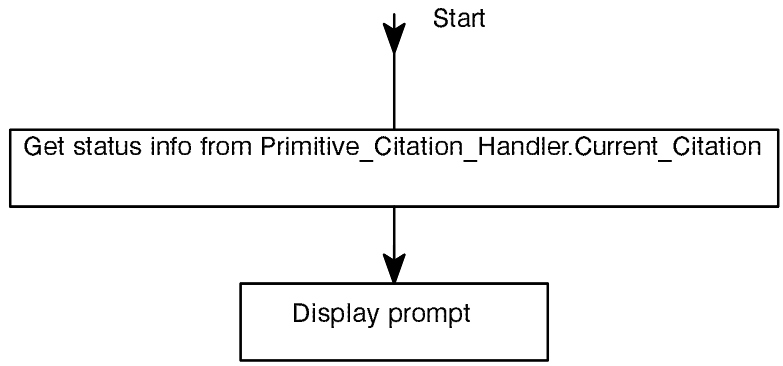
3.2.8.2. Mark_Line Procedure

The algorithm for this subprogram is:



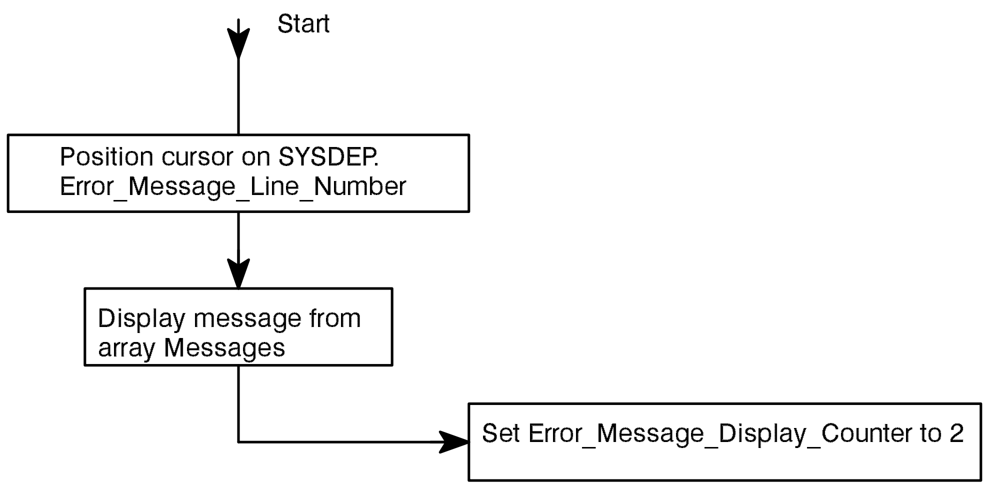
3.2.8.3. Show_Prompt Procedure

The algorithm for this subprogram is:



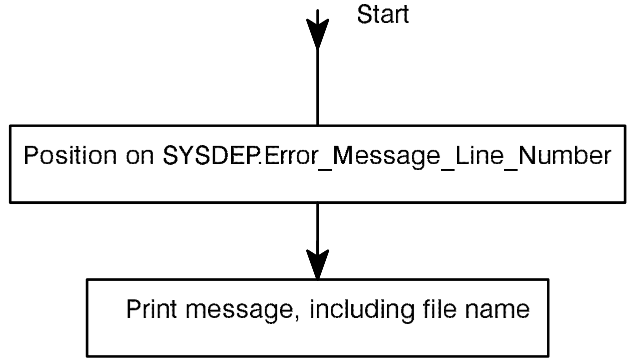
3.2.8.4. Show_Error Procedure

The algorithm for this subprogram is:



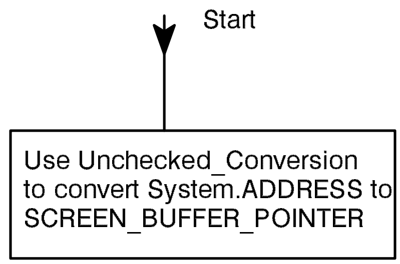
3.2.8.5. Print_Log_File_Close_Message Procedure

The algorithm for this subprogram is:



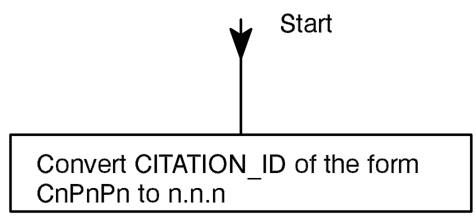
3.2.8.6. Convert Function

The algorithm for this subprogram is:



3.2.8.7. Citation_to_Display Function

The algorithm for this subprogram is:



3.2.9. Citation_Handler Package

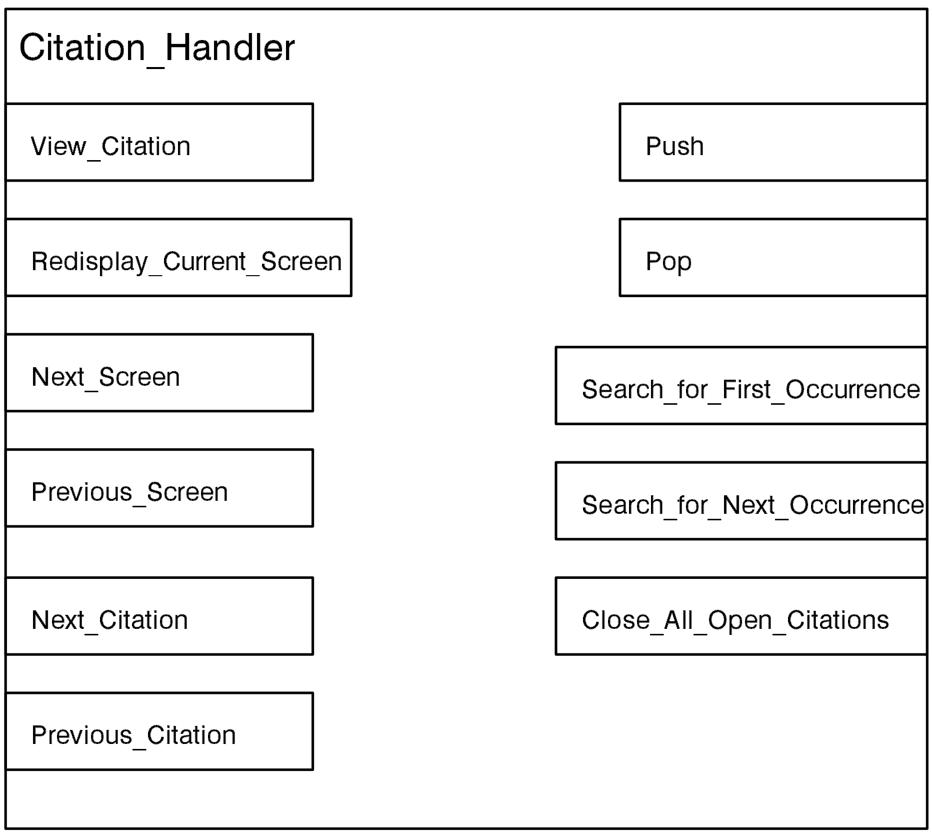
The Citation_Handler Package provides a series of methods for citation manipulation. It is used by the Command_Dispatcher to perform many of the major commands.

Mapping to Requirements

The Citation_Handler implements the display, movement, and searching capabilities in Sections 3.2.1, 3.2.3, and 3.2.4 of the SRS.

Design

The Citation_Handler Package presents the following sets of methods, types, data, and exceptions in its interface:



In this OID symbol, the rectangles represent subprograms.

CSC Specification

```
-- **********************************************************************
-- ON-LINE Ada LANGUAGE REFERENCE MANUAL
-- by Richard Conn
with Citation_Definition;
package Citation_Handler is
-- Abstract state machine for selecting and working with a given citation

  procedure View_Citation
      (New_Citation : in Citation_Definition.CITATION_ID);
  -- Start viewing a new citation, displaying the first screen

  procedure Redisplay_Current_Screen;
  -- Refresh current screen in current citation

  procedure Next_Screen;
  -- Advance to next screen in current citation and display

  procedure Previous_Screen;
  -- Back up to previous screen in current citation and display

  procedure Next_Citation;
  -- Close current citation and view first screen of next citation

  procedure Previous_Citation;
  -- Close current citation and view first screen of previous citation

  procedure Push (New_Citation : in Citation_Definition.CITATION_ID);
  -- Save position in current citation and
  -- start viewing a new citation, displaying the first screen

  procedure Pop;
  -- Return to current position in last citation before last PUSH

  procedure Search_for_First_Occurrence (Item : in STRING);
  -- Search for first occurrence of string in current citation

  procedure Search_for_Next_Occurrence (Item : in STRING);
  -- Search for next occurrence of string in current citation

  procedure Close_All_Open_Citations;
  -- Close all open citations

end Citation_Handler;
```

Required Program Units

The specification requires the following program units:

```
Citation_Definition
```

The body requires the following program units:

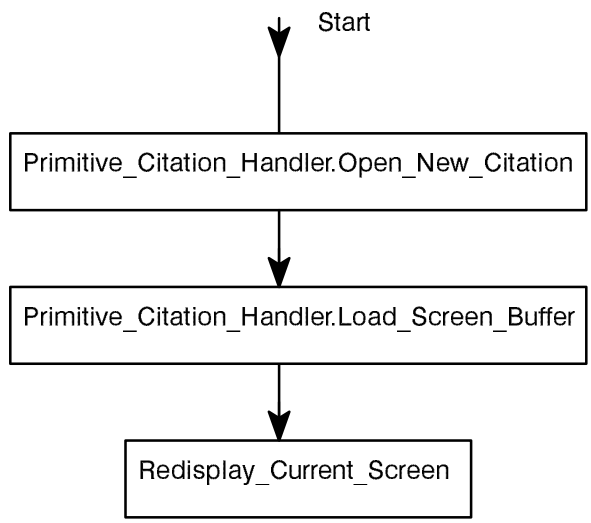
```
Primitive_Citation_Handler
Screen_Display_Controller
```

Internal Global Code, Types, and Objects

There are no items of global code, types, or objects within the body of the Citation_Handler Package.

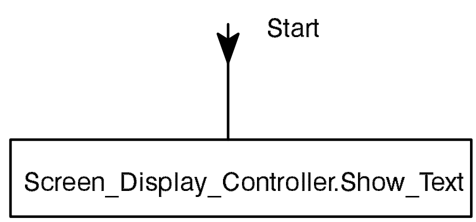
3.2.9.1. View_Citation Procedure

The algorithm for this subprogram is:



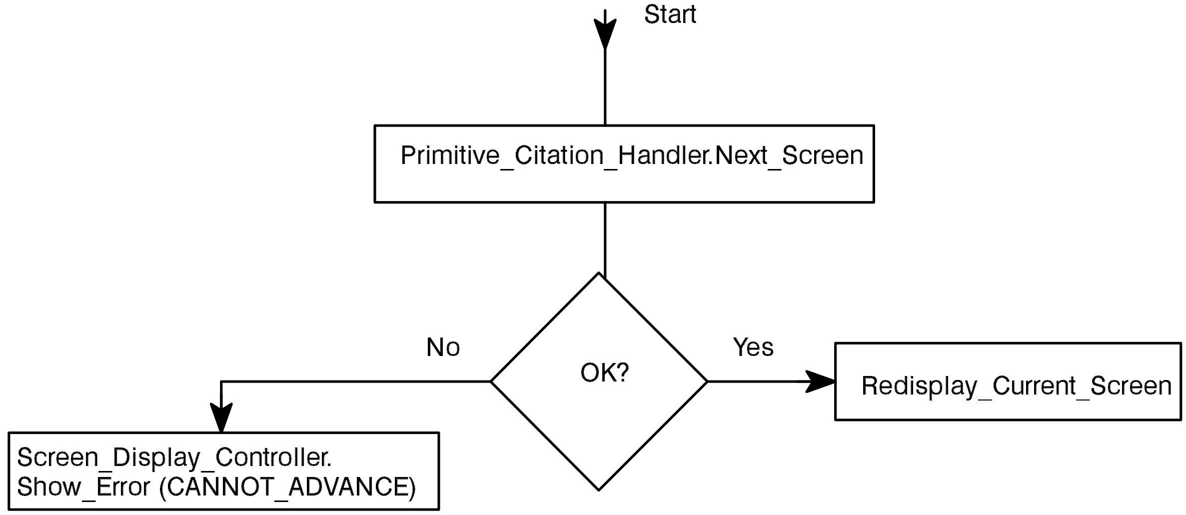
3.2.9.2. Redisplay_Current_Screen Procedure

The algorithm for this subprogram is:



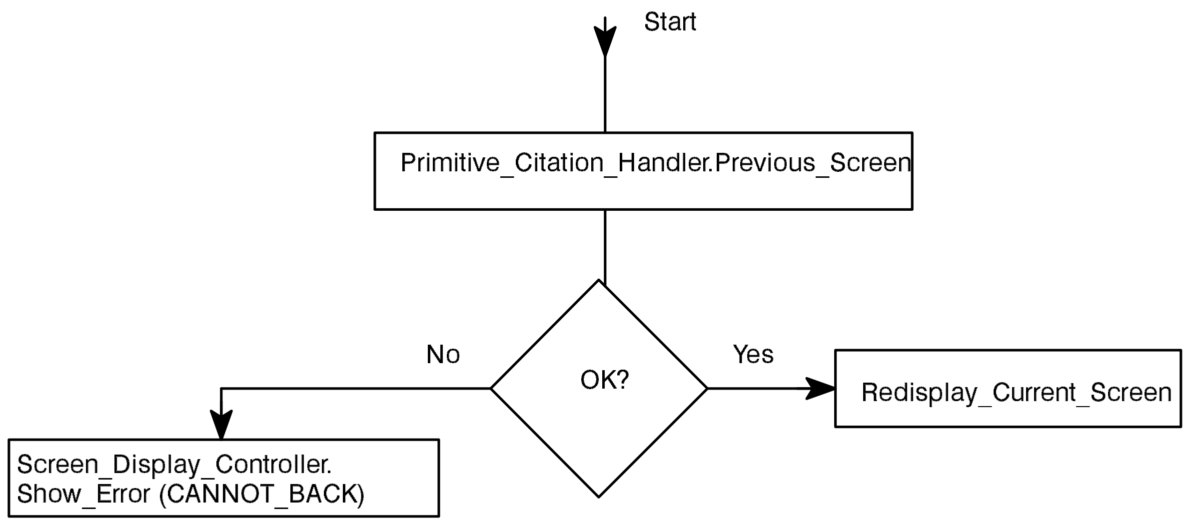
3.2.9.3. Next_Screen Procedure

The algorithm for this subprogram is:



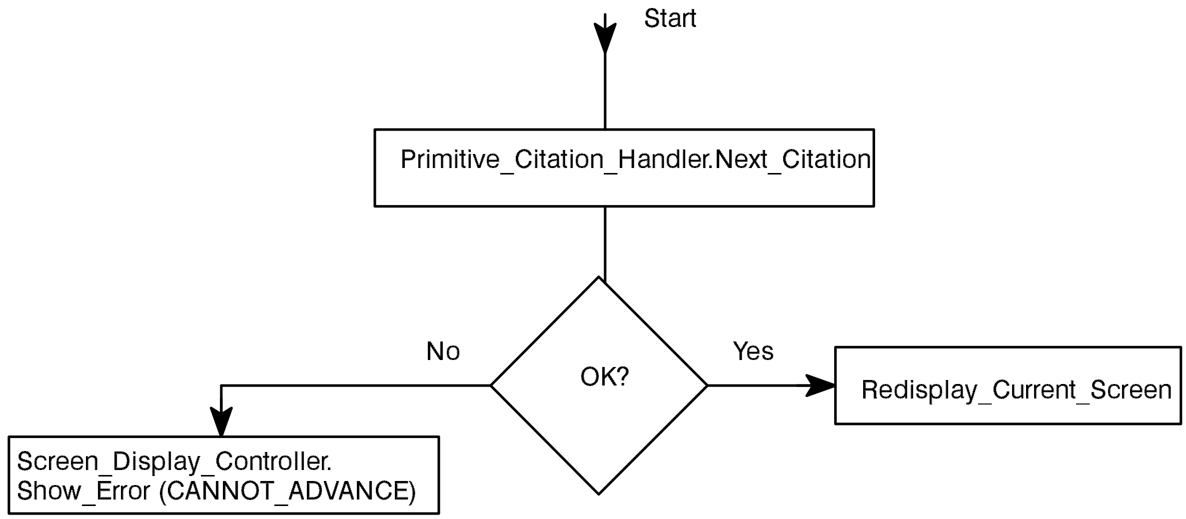
3.2.9.4. Previous_Screen Procedure

The algorithm for this subprogram is:



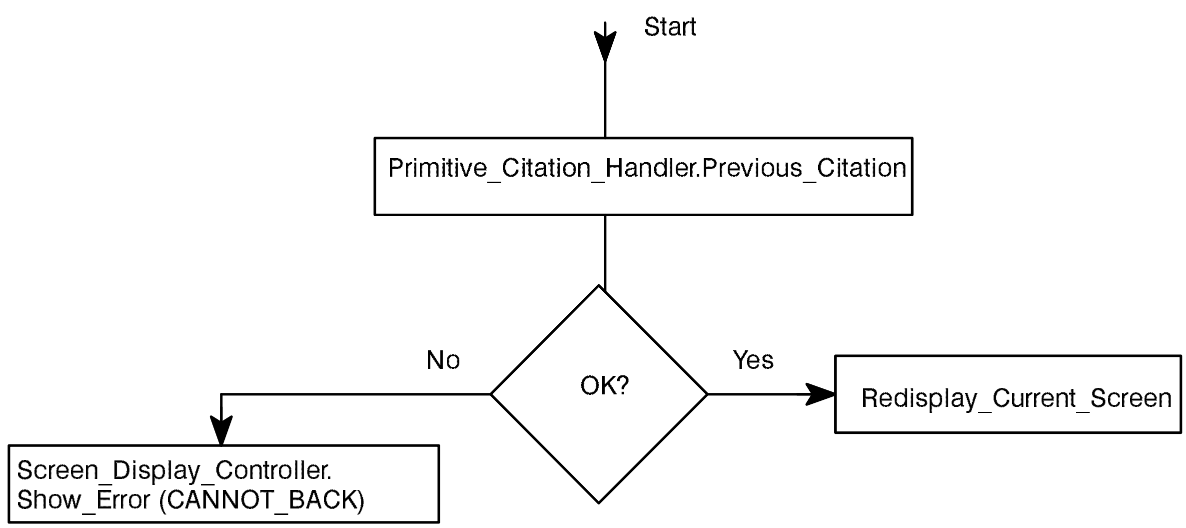
3.2.9.5. Next_Citation Procedure

The algorithm for this subprogram is:



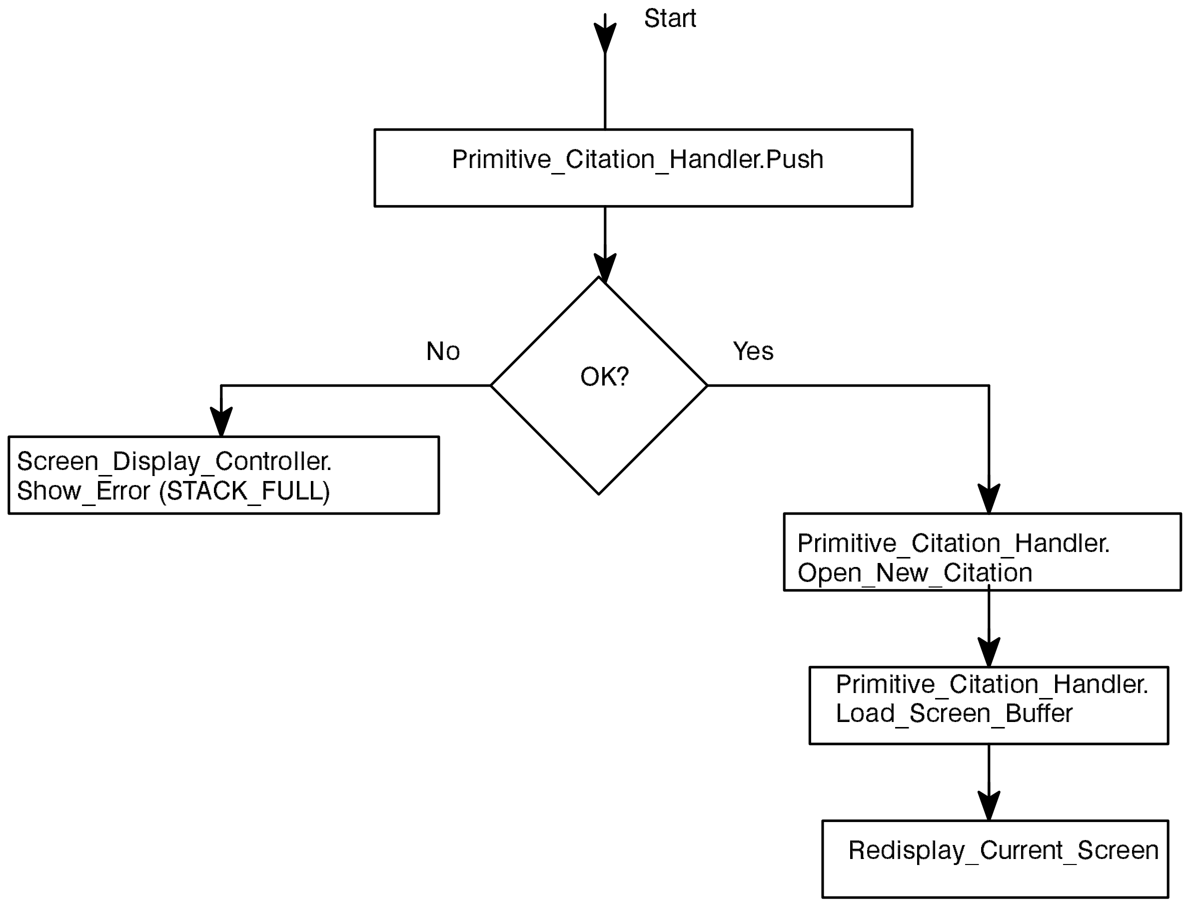
3.2.9.6. Previous_Citation Procedure

The algorithm for this subprogram is:



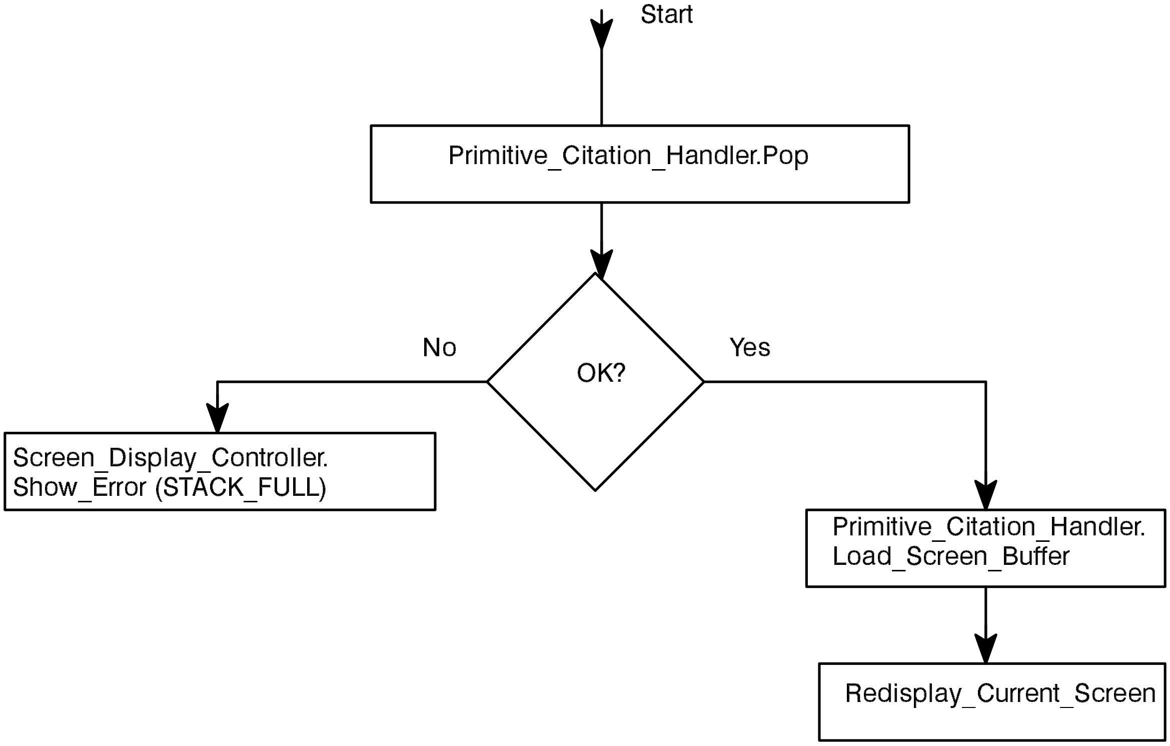
3.2.9.7. Push Procedure

The algorithm for this subprogram is:



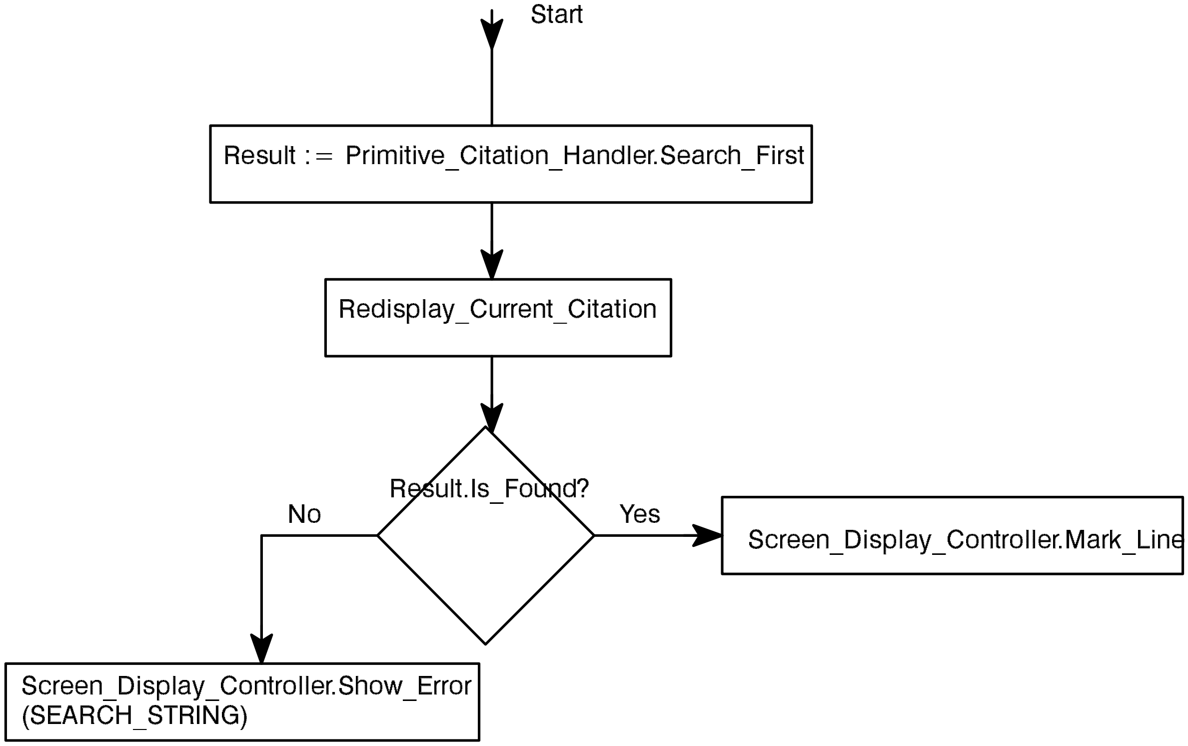
3.2.9.8. Pop Procedure

The algorithm for this subprogram is:



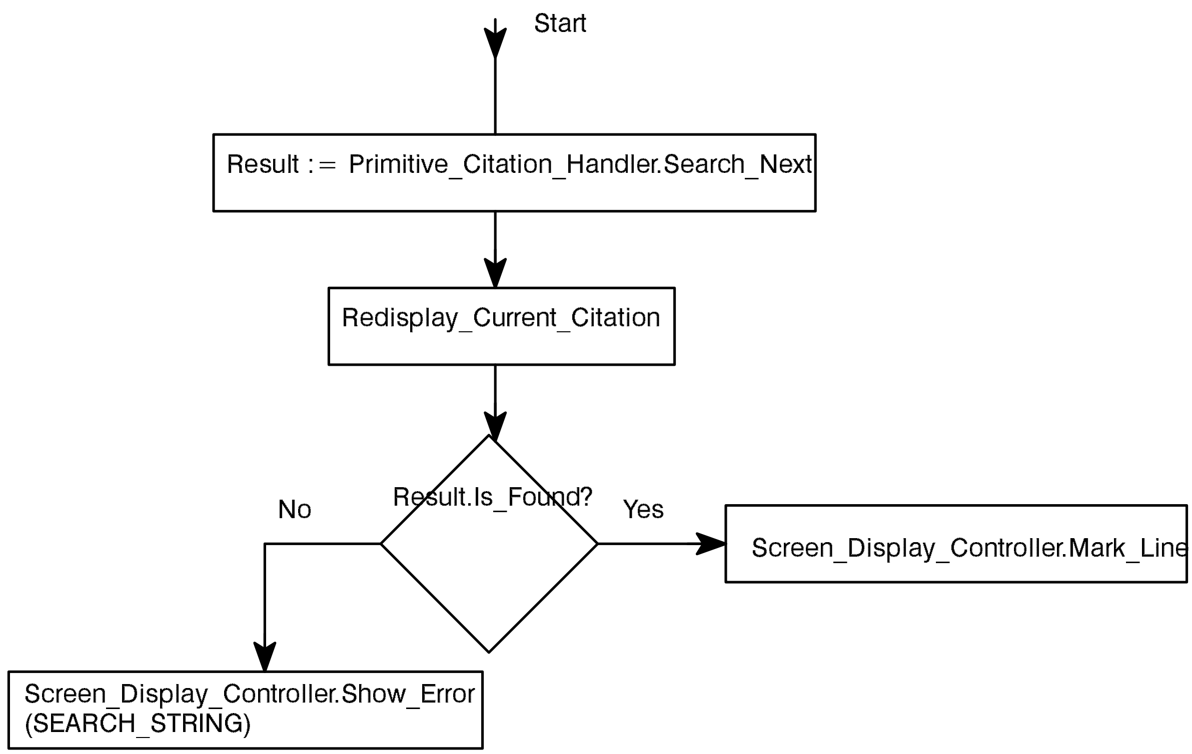
3.2.9.9. Search_for_First_Occurrence Procedure

The algorithm for this subprogram is:



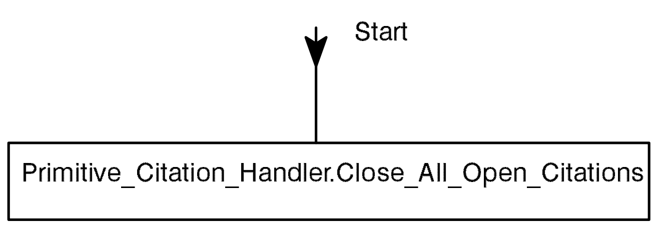
3.2.9.10. Search_for_Next_Occurrence Procedure

The algorithm for this subprogram is:



3.2.9.11. Close_All_Open_Citations Procedure

The algorithm for this subprogram is:



3.2.10. Primitive_Citation_Handler Package

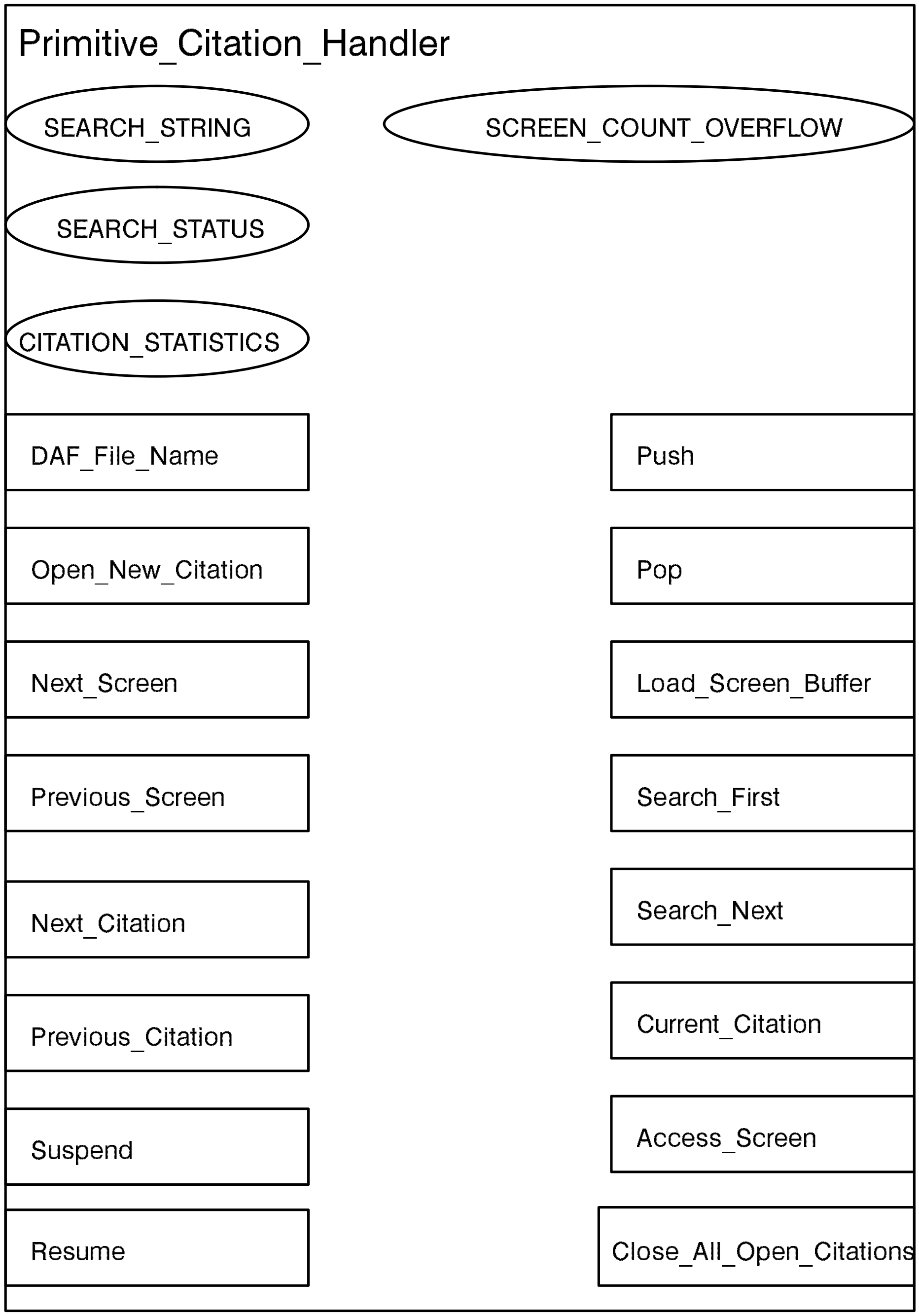
Citation manipulation is so complex an issue that a design decision was made to provide a set of primitive, independently testable set of subprograms for low-level manipulation of citations. The Primitive_Citation_Handler Package provides these subprograms. The Citation_Handler Package would then use these routines to provide the capabilities invoked through the Command_Dispatcher.

Mapping to Requirements

The Primitive_Citation_Handler Package implements the display, movement, and searching capabilities in Sections 3.2.1, 3.2.3, and 3.2.4 of the SRS.

Design

The Primitive_Citation_Handler Package presents the following sets of methods, types, data, and exceptions in its interface:



In this OID symbol, the small ovals represent data types, the large oval represents an exception, and the rectangles represent subprograms.

CSC Specification

```
-- ************************************************************************
-- ON-LINE Ada LANGUAGE REFERENCE MANUAL
-- by Richard Conn
with SYSDEP;
with Citation_Definition;
with Screen_Display_Controller;
package Primitive_Citation_Handler is

  subtype SEARCH_STRING is STRING (1..SYSDEP.Screen_String_Length);

  -- Statistics on current citation
  type CITATION_STATISTICS is record
    ID                        : Citation_Definition.CITATION_ID;
    Current_Screen_Number     : NATURAL;
    Total_Number_of_Screens   : NATURAL;
    Stack_Level               : NATURAL;
    Search_Str                : SEARCH_STRING;
    Search_Last               : NATURAL; -- index of last char in Search_Str
    Search_May_Be_Continued   : BOOLEAN;
  end record;

  -- Status of a search request
  type SEARCH_STATUS is record
    Is_Found        : BOOLEAN;  -- TRUE if string was found
    Found_on_Screen : NATURAL;  -- if found, screen string was found on
    Found_on_Line   : NATURAL;  -- if found, line string was found on
  end record;

  -- Exceptions:
  SCREEN_COUNT_OVERFLOW : exception;
    -- raised if number of screens exceeds SYSDEP.Max_Number_of_Screens
    -- raised by Open_New_Citation

  function DAF_File_Name (ITEM : in Citation_Definition.CITATION_ID)
      return STRING;
  -- Return the name of the *.daf file associated with a given CITATION_ID

  procedure Open_New_Citation (ID : in Citation_Definition.CITATION_ID);
  -- Open a new citation for processing, closing the old one if
  -- necessary; set the current screen to the first screen;
  -- build an array of information on the screens

  function Push return BOOLEAN;
  -- Push the stack, returning TRUE if OK

  function Pop return BOOLEAN;
  -- Pop the stack, returning TRUE if OK
  -- Screen Buffer is loaded appropriately

  procedure Load_Screen_Buffer;
  -- Load the screen buffer with the current screen

  function Next_Screen return BOOLEAN;
```

```
  -- Advance to the next screen, returning TRUE if done;
  -- if at last screen of current citation, advance to the first screen
  -- of the next citation
  -- Screen Buffer is loaded appropriately

  function Previous_Screen return BOOLEAN;
  -- Back up to the previous screen, returning TRUE if done;
  -- if at first screen of current citation, back up to last screen
  -- of previous citation
  -- Screen Buffer is loaded appropriately

  function Next_Citation return BOOLEAN;
  -- Advance to the first screen of the next citation, returning TRUE
  -- if done Screen Buffer is loaded appropriately

  function Previous_Citation return BOOLEAN;
  -- Back up to the first screen of the previous citation, returning TRUE
  -- if done
  -- Screen Buffer is loaded appropriately

  function Search_First (Item : in STRING) return SEARCH_STATUS;
  -- Search for the Item from the beginning of the citation;
  -- if Item is an empty string, resume search for last item requested

  function Search_Next (Item : in STRING) return SEARCH_STATUS;
  -- Resume search for Item from the next line in the citation;
  -- if Item is an empty string, resume search for last item requested

  function Current_Citation return CITATION_STATISTICS;
  -- Return the statistics on the current citation

  procedure Close_All_Open_Citations;
  -- Close all open citation files

  procedure Suspend;
  -- Suspend operation for Print_Log

  procedure Resume;
  -- Resume operation for Print_Log

  function Access_Screen
    return Screen_Display_Controller.SCREEN_BUFFER_POINTER;
  -- Return the address of the screen for printing or displaying

end Primitive_Citation_Handler;
```

Required Program Units

The specification requires the following program units:

```
SYSDEP
Citation_Definition
Screen_Display_Controller
```

The body requires the following program units:

DAF_Handler

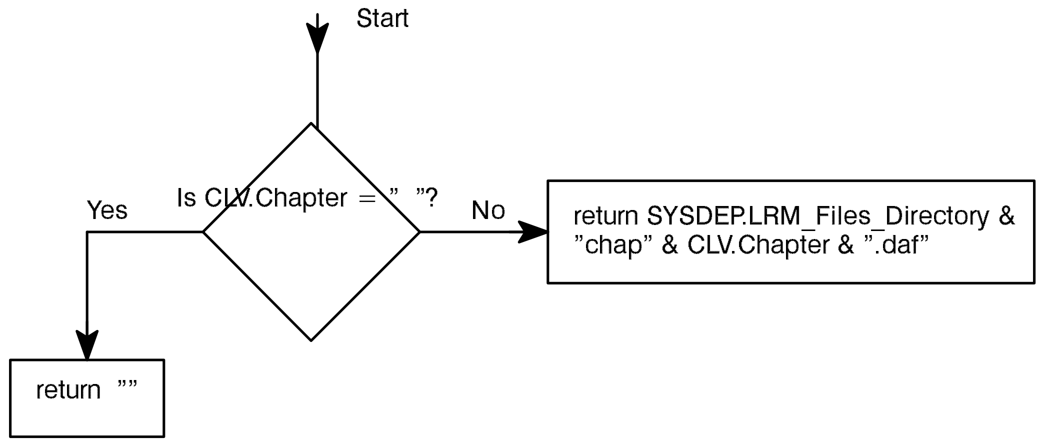
Internal Global Code, Types, and Objects

The following are global within the body:

```
-- Used to track the first and last line of each screen displayed
type SCREEN_BOUNDARIES is record
  First_Line : NATURAL := 0;
  Last_Line  : NATURAL := 0;
end record;

-- First and last lines for a maximum number of screens
type CITATION_SCREEN_LIST is array (1..SYSDEP.Max_Number_of_Screens) of
  SCREEN_BOUNDARIES;

-- Information pertaining to each citation
type CITATION_STATE_INFORMATION is record
  ID                     : Citation_Definition.CITATION_ID;
  Current_Screen         : NATURAL := 0;
  Number_Screens         : NATURAL := 0;
  Screen_List            : CITATION_SCREEN_LIST;
  Search_May_Be_Resumed  : BOOLEAN := FALSE;
  Resume_on_Line         : NATURAL;
  File_ID                : DAF_Handler.DAF_ID := 0;
end record;

-- Stack of information on all citations selected
type CITATION_VECTOR is array (1..SYSDEP.Citation_Stack_Depth) of
  CITATION_STATE_INFORMATION;

-- The actual stack of citations
Citation_Stack : CITATION_VECTOR;
Citation_Index : NATURAL := 1;

-- The current citation we are working on
Cur_Cit        : CITATION_STATE_INFORMATION;

-- The actual lines on the current screen
SBuffer      : Screen_Display_Controller.SCREEN_BUFFER;
SBuffer_Last : NATURAL;

-- Flag used by suspend/resume routines
Suspend_Flag : BOOLEAN := FALSE;

-- Variables used by search routines
Search_Str   : SEARCH_STRING;
Search_Last  : NATURAL := 0;
```

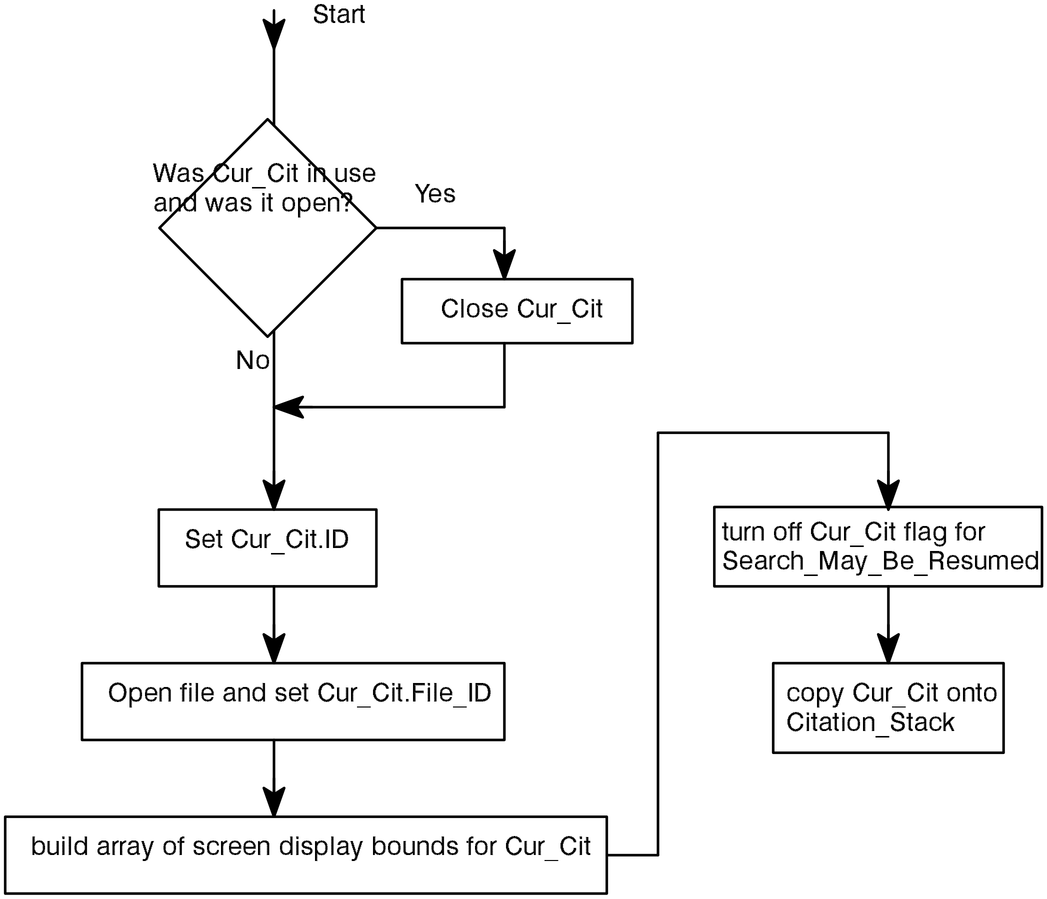
3.2.10.1. DAF_Name Function

The algorithm for this subprogram is:



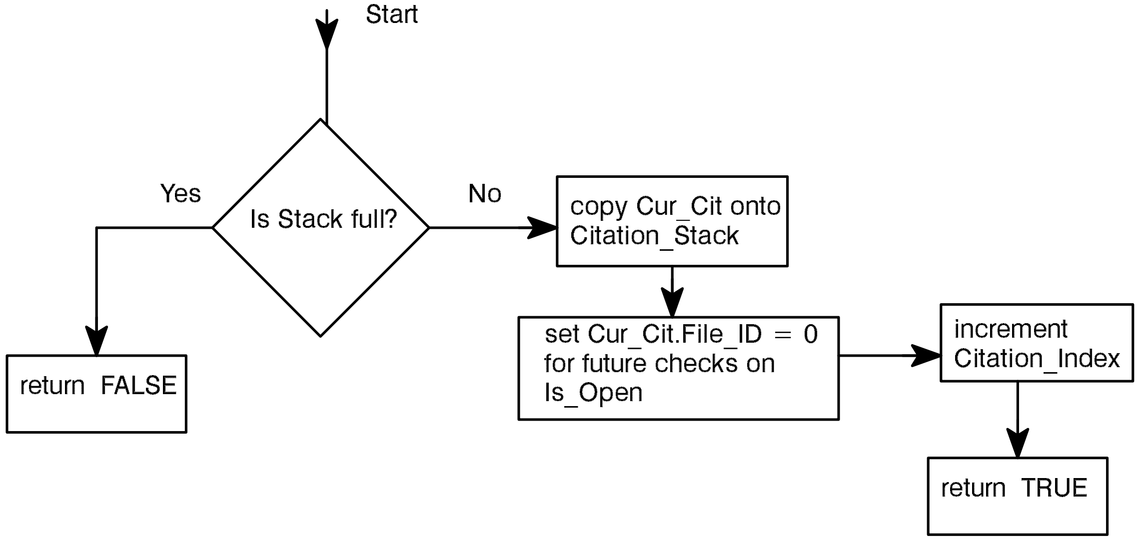
3.2.10.2. Open_New_Citation Procedure

The algorithm for this subprogram is:



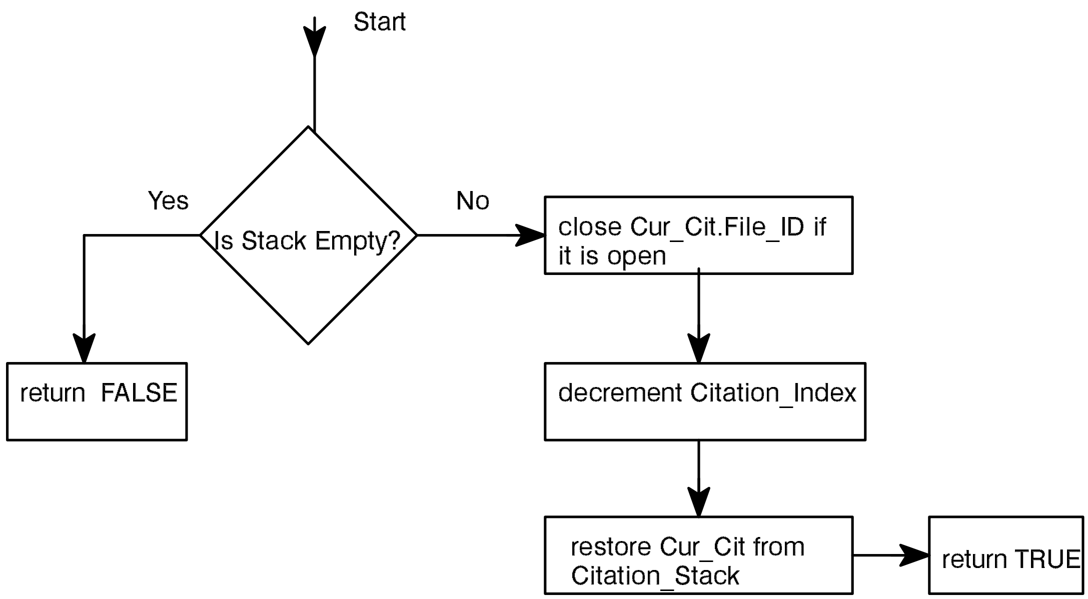
3.2.10.3. Push Function

The algorithm for this subprogram is:



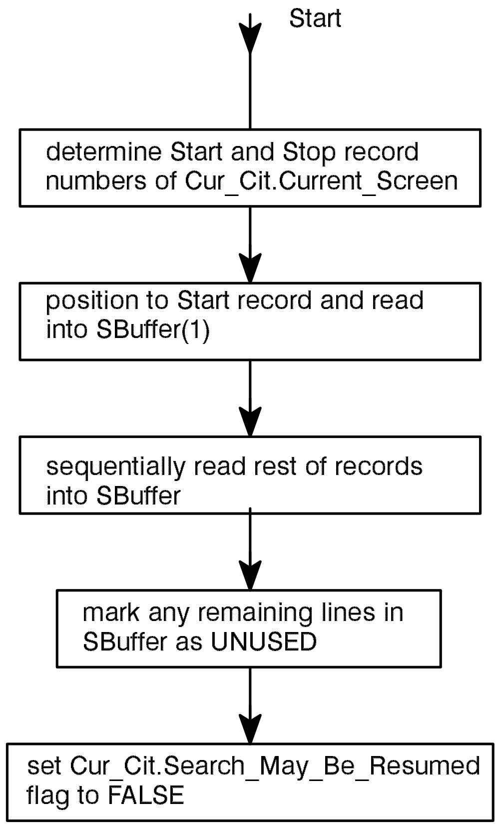
3.2.10.4. Pop Function

The algorithm for this subprogram is:



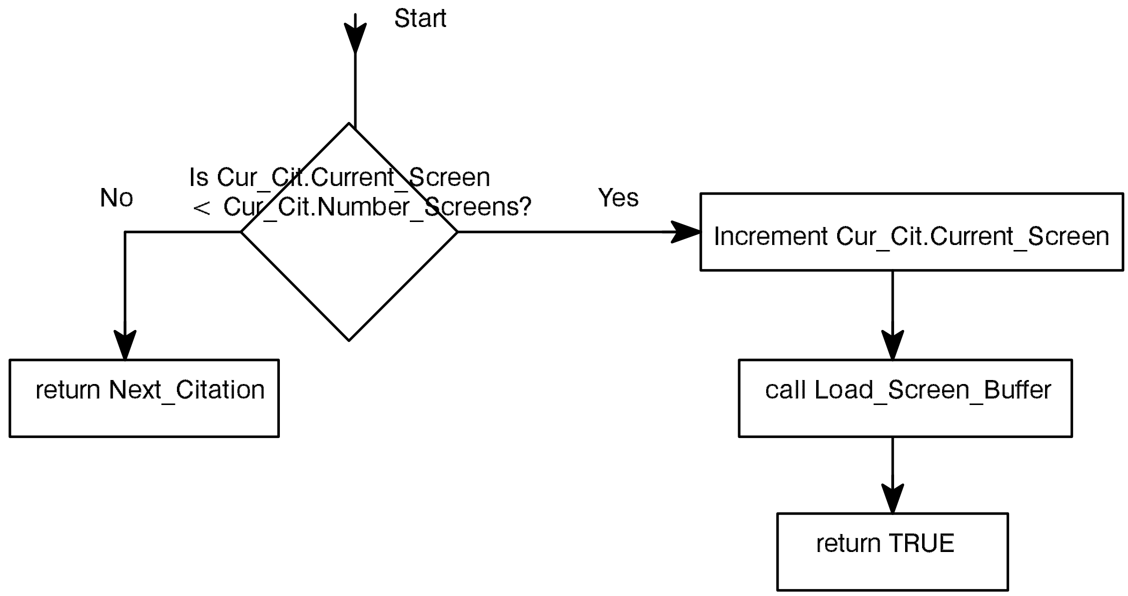
3.2.10.5. Load_Screen_Buffer Procedure

The algorithm for this subprogram is:



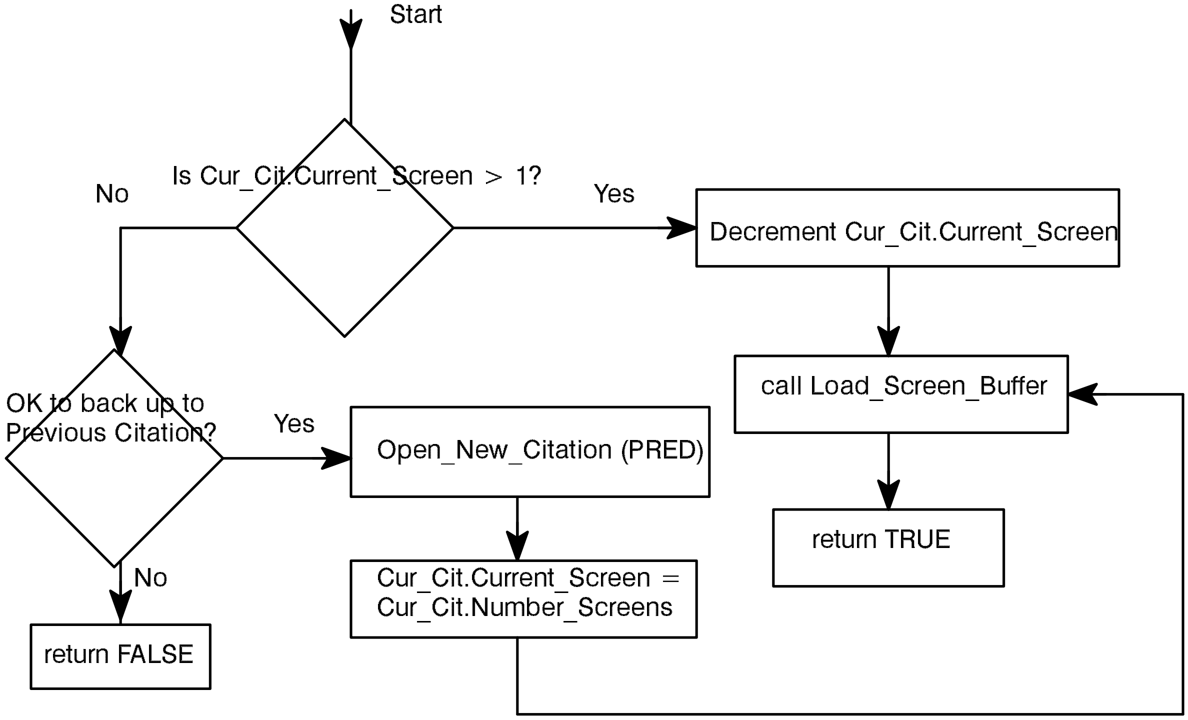
3.2.10.6. Next_Screen Function

The algorithm for this subprogram is:



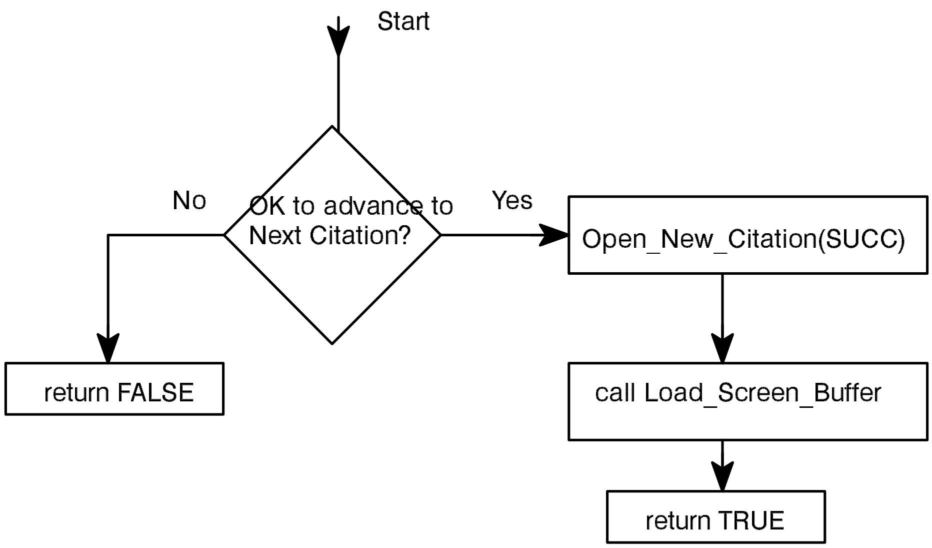
3.2.10.7. Previous_Screen Function

The algorithm for this subprogram is:



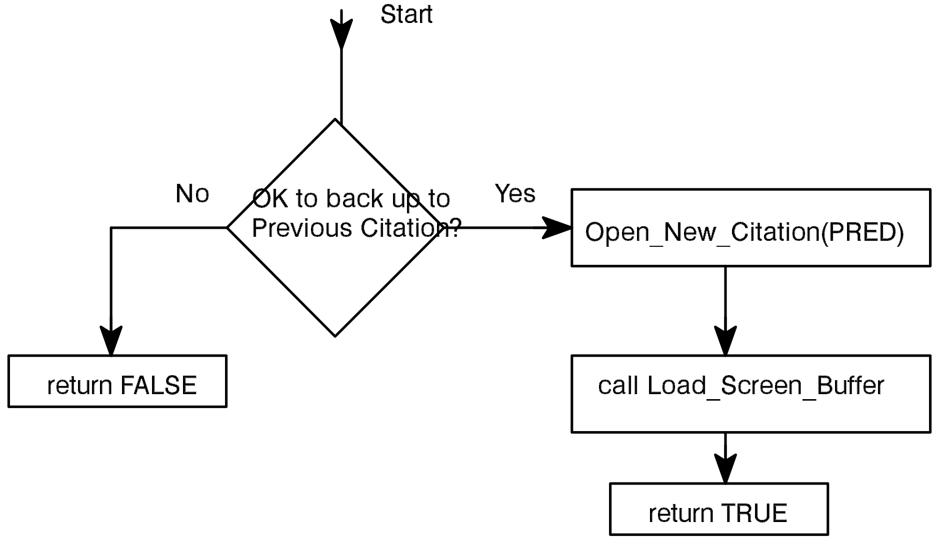
3.2.10.8. Next_Citation Function

The algorithm for this subprogram is:



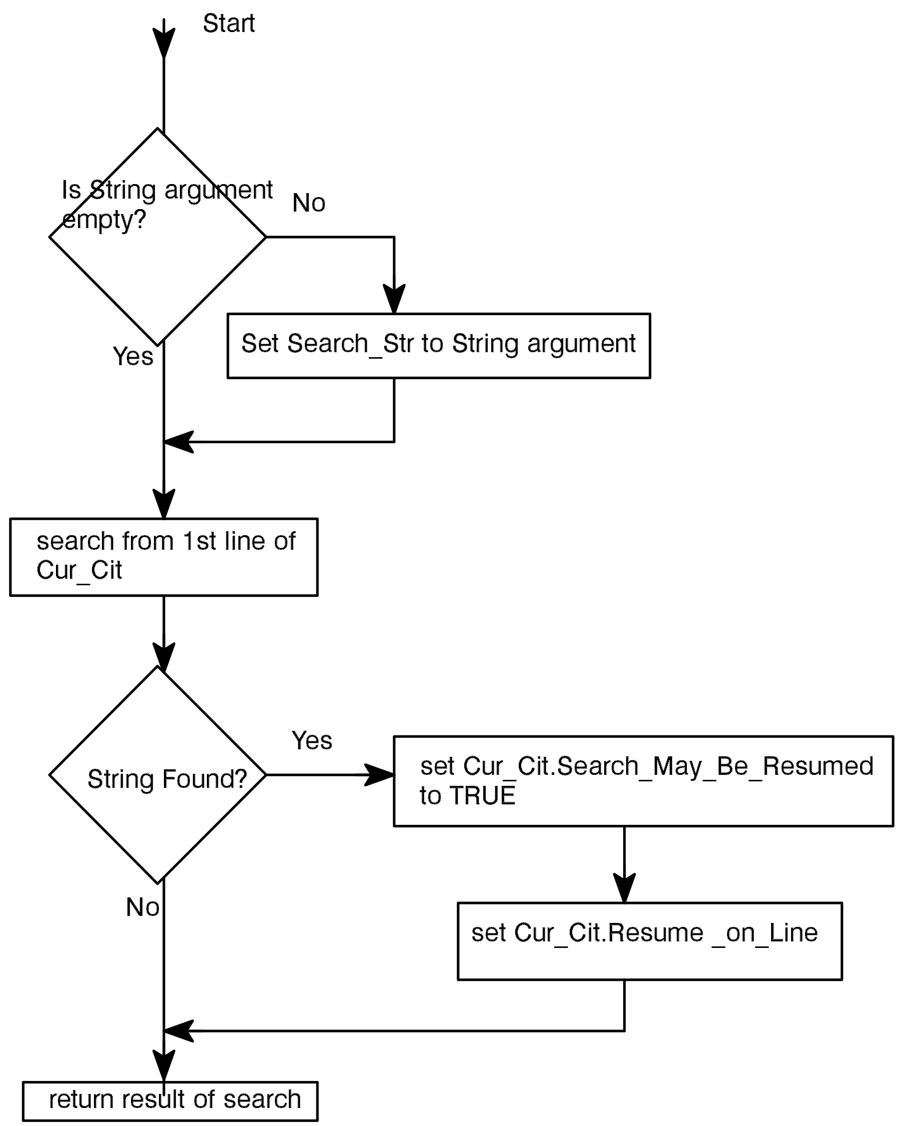
3.2.10.9. Previous_Citation Function

The algorithm for this subprogram is:



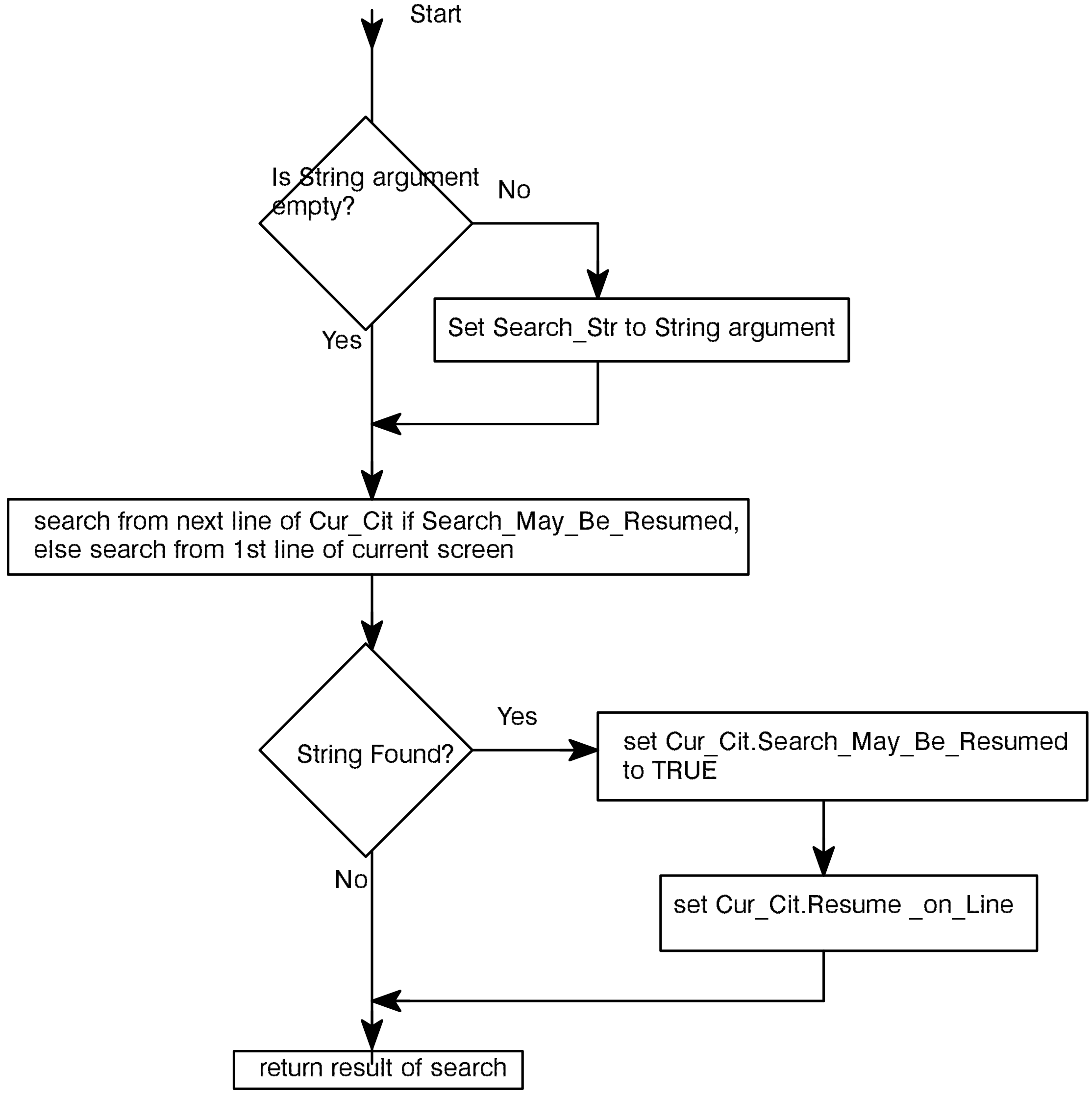
3.2.10.10. Search_First Function

The algorithm for this subprogram is:



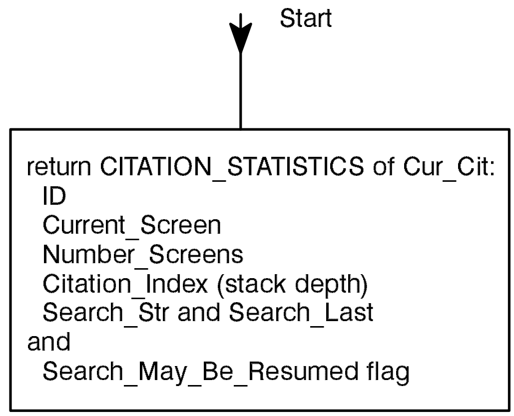
3.2.10.11. Search_Next Function

The algorithm for this subprogram is:



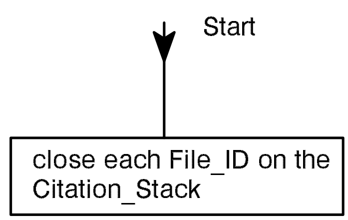
3.2.10.12. Current_Citation Function

The algorithm for this subprogram is:



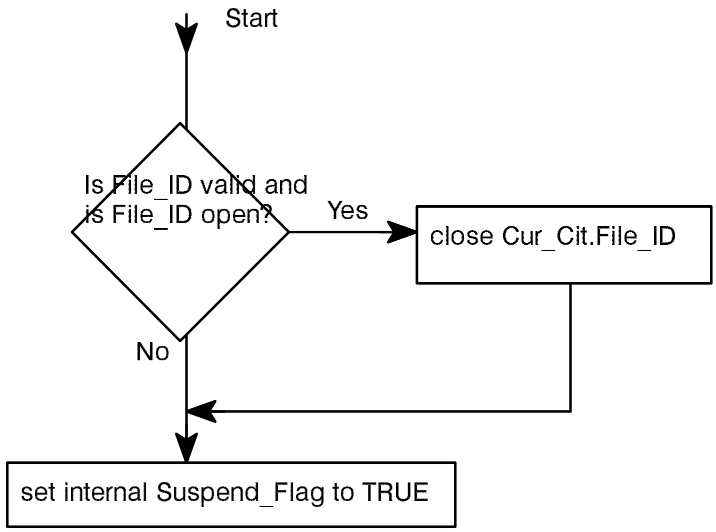
3.2.10.13. Close_All_Open_Citations Procedure

The algorithm for this subprogram is:



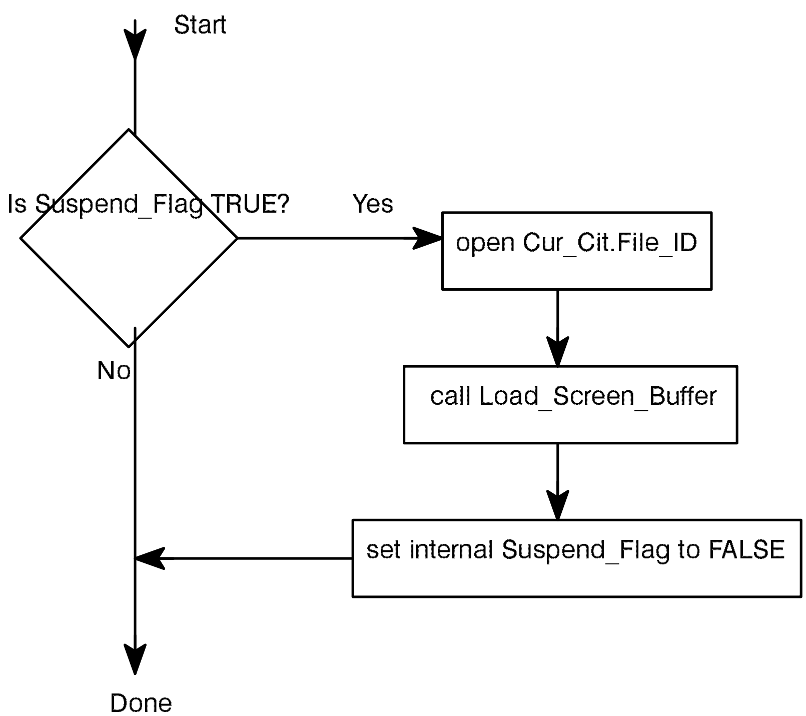
3.2.10.14. Suspend Procedure

The algorithm for this subprogram is:



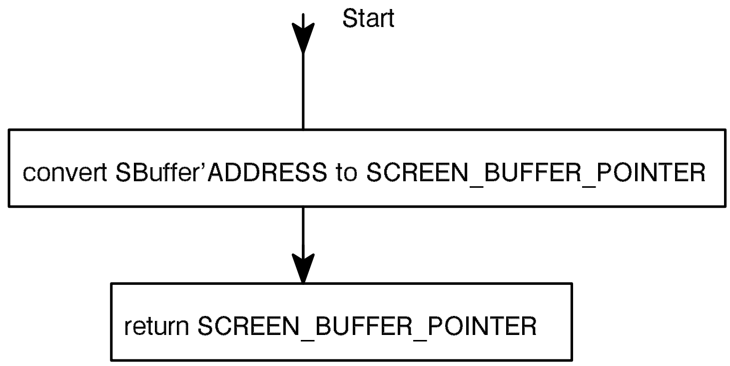
3.2.10.15. Resume Procedure

The algorithm for this subprogram is:



3.2.10.16. Access_Screen Function

The algorithm for this subprogram is:



3.2.11. Print_Log_Handler Package

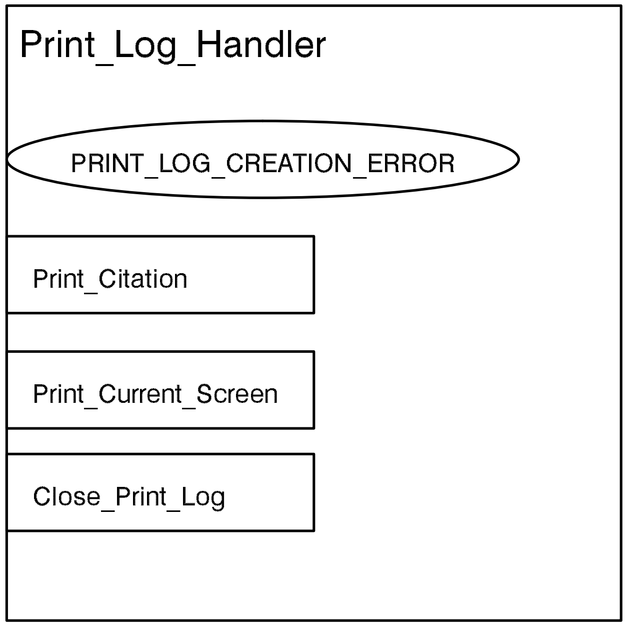
The Print_Log_Handler Package implements a passive object which creates and places entries in the Print Log File.

Mapping to Requirements

This CSC meets requirements in the following sections of the SRS: 3.2.2 (the capability to print citations), 3.3 (Print Log File), and 3.4 (Print Log File).

Design

The Print_Log_Handler Package presents the following sets of methods, types, data, and exceptions in its interface:



In this OID symbol, the long ovals represent exceptions and the small rectangles represent subprograms.

CSC Specification

```
-- **********************************************************************
-- ON-LINE Ada LANGUAGE REFERENCE MANUAL
-- by Richard Conn
package Print_Log_Handler is
-- Abstract state machine for manipulating the Print Log File

  PRINT_LOG_CREATION_ERROR : exception;

  procedure Print_Citation;
  -- Print current citation to log file

  procedure Print_Current_Screen;
  -- Print current screen to log file

  procedure Close_Print_Log;
  -- Close log file and display message to user

end Print_Log_Handler;
```

Required Program Units

No program units are withed by the specification. The following program units are withed by the body:

```
SYSDEP
Citation_Definition
DAF_Handler
Primitive_Citation_Handler
```

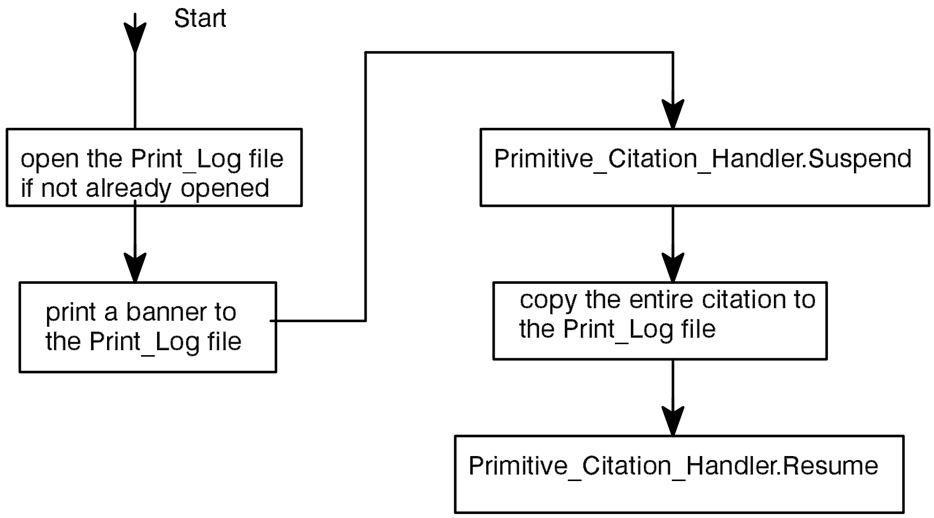
```
Screen_Display_Controller
Output_File
```

Internal Global Code, Types, and Objects

This package contains no internal global code, types, or objects.

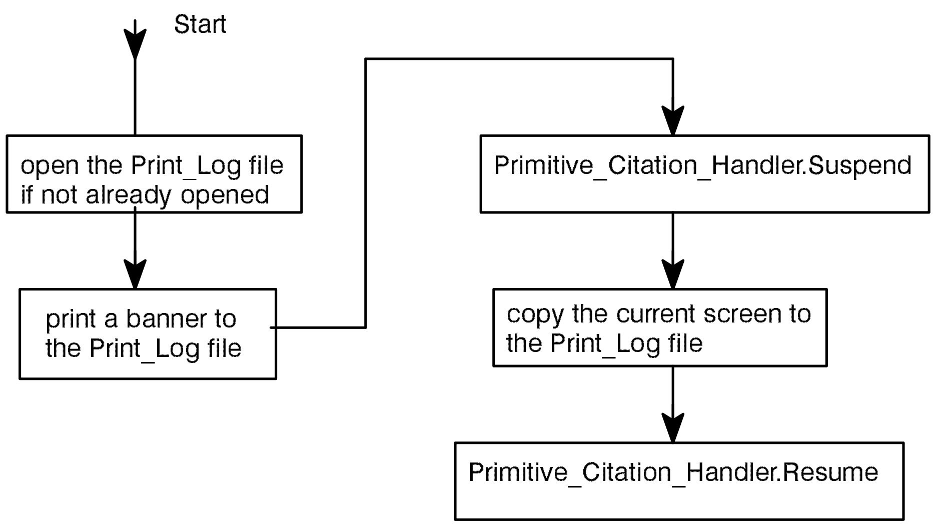
3.2.11.1. Print_Current_Citation Procedure

The algorithm for this subprogram is:



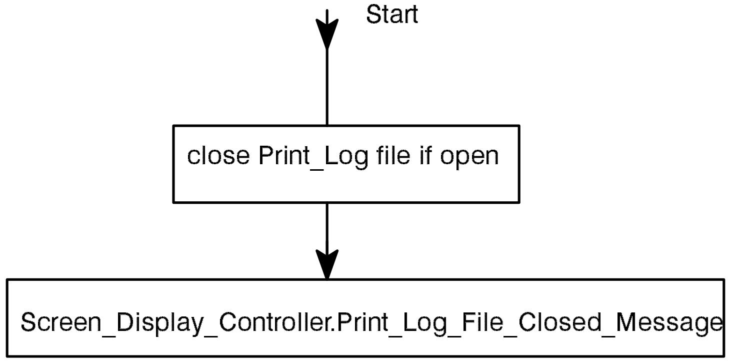
3.2.11.2. Print_Current_Screen Procedure

The algorithm for this subprogram is:



3.2.11.3. Close_Print_Log Procedure

The algorithm for this subprogram is:



3.2.12. Non-Developmental CSCs

The non-developmental CSCs are reused from the CS Parts and the standard Ada environment. They are not listed in detail here since information on them may be readily obtained from the sources identified in the references.

The CSCs from the CS Parts are:

1. CLI Package
2. Console Package
3. Input_File Package
4. Output_File Package

The CSCs from the standard Ada environment are:

1. System Package
2. Direct_IO Package
3. Unchecked_Conversion Function

4. CSCI data

4.1. SYSDEP Package

The SYSDEP (System Dependencies) Package is as follows:

```
package SYSDEP is
-- System Dependencies Package

   LRM_Files_Directory     : constant STRING :=
     "/ada2_home/local/reader/ada_lrm/";    -- GE
   --  "/usr/local/swengrg/reader/ada_lrm/";  -- UC
   -- "c:\reader\ada_lrm\";  -- PC
     -- name of the directory containing the LRM DAF files "chapxx.daf";
     -- be sure to follow this directory name with a separator so the
     -- file name may be appended to it

   Program_Name            : constant STRING :=
     "Ada LRM Reader 2.0";
   -- this will appear on the prompt line

   Print_File_Name         : constant STRING :=
     "lrm.log";
     -- this will be created in the user´s local directory

   Citation_Stack_Depth    : constant := 20; -- citations
     -- maximum number of citations which may be PUSHed;
     -- you should not have to change this

 -- These values are set for a VT100 terminal and should not have to be
 -- changed
   Screen_Width            : constant := 75; -- chars
    -- maximum number of characters that can be displayed on a line
    -- without wrap; this is set to allow right and left margins and
    -- markers you should not have to change this

   Text_Line_Count           : constant := 22; -- lines
   Command_Line_Number       : constant := 23; -- line number
   Error_Message_Line_Number : constant := 24; -- line number
   Screen_String_Length      : constant := 78; -- characters
   Search_Pointer_Column     : constant := 79; -- column number
    -- Line count, line numbers, and column numbers for VT100 display
    -- you should not have to change this

 -- These values are picked up from an examination of the text files
 -- of the Ada LRM; they represent a slight increase over the values
 -- actually determined from the examination in order to allow for
 -- a minor growth.

   Max_String_Length       : constant := 110; -- chars
     -- for command line and file line input;
     -- you should not have to change this

   Max_Number_of_Citations : constant := 40; -- citations per file
     -- based on number of citations in each file;
     -- you should not have to change this
```

```
   Total_Number_of_Citations : constant := 230; -- citations in all files
     -- based on a count of the citations in all files;
     -- you should not have to change this

   Max_Number_of_Screens   : constant := 6000/(Text_Line_Count) + 1;
     -- based on number of lines in largest file (chapin.doc);
     -- you should not have to change this

  subtype COPYRIGHT_STRING is STRING (1..60);
  type COPYRIGHT_NOTICE is array (NATURAL range <>) of COPYRIGHT_STRING;
  Full_Copyright_Notice : constant COPYRIGHT_NOTICE := ( -- detail omitted
    );
  Intro_Copyright_Notice : constant COPYRIGHT_NOTICE := ( -- detail omitted
    );

end SYSDEP;
```

4.2. Citation_Definition Package Specification

The following is an abbreviated version of the specification of package Citation_Definition (generated by the Make_Cit Procedure as the file CIT.ADA). The details of package Citation_Definition may vary somewhat in later versions of the Make_Cit CSC, but the basic structure of the Citation_Definition Package CSC can be clearly studied from the following listing:

```
with DAF_Handler;
package Citation_Definition is

  type CITATION_ID is (
    C1,
    C1P1,
    C1P1P1,
    C1P1P2,
    C1P2,
    C1P3,
    C1P4,
    C1P5,
    C1P6,
    C2,
    C2P1,
    C2P2,
    C2P3,
    C2P4,
    C2P4P1,
    C2P4P2,
    C2P5,
    C2P6,
    C2P7,
    C2P8,
    C2P9,
    C2P10,
    -- detail omitted
    C14,
    C14P1,
    C14P2,
    C14P2P1,
    C14P2P2,
```

```
  C14P2P3,
  C14P2P4,
  C14P2P5,
  C14P3,
  C14P3P1,
  C14P3P2,
  C14P3P3,
  C14P3P4,
  C14P3P5,
  C14P3P6,
  C14P3P7,
  C14P3P8,
  C14P3P9,
  C14P3P10,
  C14P4,
  C14P5,
  C14P6,
  C14P7,
  CA,
  CB,
  CC,
  CD,
  CE,
  CF,
  CONTENTS,
  FOREWARD,
  INDEX,
  POSTSCRIPT,
  HELP,
  ABOUT,
  ERROR,
-- Commands for immediate execution
  N, P,
  USER_INPUT, PRINT, PS,
  NEXT, PREVIOUS, PAUSE, PUSH, POP,
  SEARCH_FIRST, SEARCH_NEXT, REFRESH,
  QUIT
);

type CITATION_LOCATION is record
  Chapter : STRING(1..2);
  Start   : DAF_Handler.LINE_NUMBER;
  Stop    : DAF_Handler.LINE_NUMBER;
end record;

type CITATION_LOCATION_VECTOR is array (CITATION_ID) of CITATION_LOCATION;

CLV : constant CITATION_LOCATION_VECTOR := (
  C1 => ("01", 23, 46),
  C1P1 => ("01", 47, 55),
  C1P1P1 => ("01", 56, 120),
  C1P1P2 => ("01", 121, 144),
  C1P2 => ("01", 145, 177),
  C1P3 => ("01", 178, 253),
  C1P4 => ("01", 254, 430),
  C1P5 => ("01", 431, 521),
```

```
C1P6 => ("01", 522, 582),
C2 => ("02", 24, 38),
C2P1 => ("02", 39, 130),
C2P2 => ("02", 131, 209),
C2P3 => ("02", 210, 241),
C2P4 => ("02", 242, 258),
C2P4P1 => ("02", 259, 301),
C2P4P2 => ("02", 302, 349),
C2P5 => ("02", 350, 367),
C2P6 => ("02", 368, 419),
C2P7 => ("02", 420, 448),
C2P8 => ("02", 449, 529),
C2P9 => ("02", 530, 569),
C2P10 => ("02", 570, 621),
-- detail omitted
C14 => ("14", 1, 16),
C14P1 => ("14", 17, 131),
C14P2 => ("14", 132, 169),
C14P2P1 => ("14", 170, 322),
C14P2P2 => ("14", 323, 374),
C14P2P3 => ("14", 375, 441),
C14P2P4 => ("14", 442, 530),
C14P2P5 => ("14", 531, 611),
C14P3 => ("14", 612, 715),
C14P3P1 => ("14", 716, 757),
C14P3P2 => ("14", 758, 824),
C14P3P3 => ("14", 825, 892),
C14P3P4 => ("14", 893, 1162),
C14P3P5 => ("14", 1163, 1293),
C14P3P6 => ("14", 1294, 1391),
C14P3P7 => ("14", 1392, 1514),
C14P3P8 => ("14", 1515, 1686),
C14P3P9 => ("14", 1687, 1802),
C14P3P10 => ("14", 1803, 2063),
C14P4 => ("14", 2064, 2140),
C14P5 => ("14", 2141, 2160),
C14P6 => ("14", 2161, 2200),
C14P7 => ("14", 2201, 2264),
CA => ("aa", 24, 506),
CB => ("ab", 23, 175),
CC => ("ac", 24, 320),
CD => ("ad", 24, 432),
CE => ("ae", 4, 777),
CF => ("af", 21, 61),
CONTENTS => ("co", 1, 284),
FOREWARD => ("fo", 1, 81),
INDEX => ("in", 1, 5833),
POSTSCRIPT => ("po", 1, 90),
HELP => ("he", 1, 38),
ABOUT => ("xx", 1, 12),
ERROR        => ("  ", 1, 1),
N            => ("  ", 1, 1),
P            => ("  ", 1, 1),
USER_INPUT   => ("  ", 1, 1),
PRINT        => ("  ", 1, 1),
```

```
    PS           => ("  ", 1, 1),
    NEXT         => ("  ", 1, 1),
    PREVIOUS     => ("  ", 1, 1),
    PAUSE        => ("  ", 1, 1),
    PUSH         => ("  ", 1, 1),
    POP          => ("  ", 1, 1),
    SEARCH_FIRST => ("  ", 1, 1),
    SEARCH_NEXT  => ("  ", 1, 1),
    REFRESH      => ("  ", 1, 1),
    QUIT         => ("  ", 1, 1)
  );

end Citation_Definition;
```

Adaptation Information

The CITATION_ID called CONTENTS is always the successor to the last of the "sequential" citations. That is, the user may advance sequentially from the first citation to the citation before CONTENTS, but an attempt to advance beyond the citation before CONTENTS is not allowed. If this tool is adapted to work with different data files, following this convention will reduce the changes required.

4.3. DAF_Handler Exported Types, Data, and Exceptions

The following Ada code fragments show the detail of the type, data, and exceptions exported by the package DAF_Handler.

```
type LINE_TYPE is (NORMAL, SECTION, UNUSED);

type LINE is record
  Str      : STRING (1 .. SYSDEP.Screen_String_Length);
  Str_Last : NATURAL := 0; -- index of last char in Str
  Kind     : LINE_TYPE := NORMAL;
end record;

subtype LINE_NUMBER is NATURAL range 1 .. NATURAL'LAST;

subtype DAF_ID is NATURAL range 0 .. SYSDEP.Citation_Stack_Depth;

DAF_CREATION_ERROR : exception;
FILE_NOT_FOUND     : exception;
NO_DAF_OPEN        : exception;
READ_ERROR         : exception;
WRITE_ERROR        : exception;
STACK_OVERFLOW     : exception;
UNEXPECTED_ERROR   : exception;
```

See the full specification of the package DAF_Handler in Section 4 of this SDD to see further details on how these types and exceptions are used.

4.4. Screen_Display_Controller Exported Types, Data, and Exceptions

The following Ada code fragments show the detail of the type, data, and exceptions exported by the package Screen_Display_Controller.

```
type ERROR_MESSAGE_ID is (INVALID_COMMAND,
                          CANNOT_ADVANCE, CANNOT_BACK,
                          STACK_EMPTY, STACK_FULL,
                          PRINT_LOG,
                          TOO_MANY_SCREENS,
                          SEARCH_STRING,
                          DAF_NOT_FOUND,
                          INTERNAL_DAF_NDFO_ERROR,
                          INTERNAL_DAF_RE_ERROR,
                          INTERNAL_DAF_SO_ERROR,
                          INTERNAL_DAF_UE_ERROR,
                          UNEXPECTED_ERROR);
type SCREEN_BUFFER is array (NATURAL´(1)..SYSDEP.Text_Line_Count) of
  DAF_Handler.LINE;
type SCREEN_BUFFER_POINTER is access SCREEN_BUFFER;
```

The body of package Screen_Display_Controller contains an array indexed by ERROR_MESSAGE_ID that contains the text of the error messages. All error messages issued by the Ada LRM Reader CSCI are spelled out in the body of this package, so this is the one location to which all error messages may be tracked.

4.5. Primitive_Citation_Handler Exported Types, Data, and Exceptions

The following Ada code fragments show the detail of the type, data, and exceptions exported by the package Primitive_Citation_Handler.

```
subtype SEARCH_STRING is STRING (1 .. SYSDEP.Screen_String_Length);

-- Statistics on the current citation
type CITATION_STATISTICS is record
  ID                      : Citation_Definition.CITATION_ID;
  Current_Screen_Number   : NATURAL;
  Total_Number_of_Screens : NATURAL;
  Stack_Level             : NATURAL;
  Search_Str              : SEARCH_STRING;
  Search_Last             : NATURAL; -- index of last char in Search_Str
  Search_May_Be_Continued : BOOLEAN;
end record;

-- Status of a search request
type SEARCH_STATUS is record
  Is_Found        : BOOLEAN; -- TRUE if string was found
  Found_on_Screen : NATURAL; -- if found, screen string was found on
  Found_on_Line   : NATURAL; -- if found, line string was found on
end record;

SCREEN_COUNT_OVERFLOW : exception;
  -- raised if number of screens exceeds SYSDEP.Max_Number_of_Screens
```

Objects of the types CITATION_STATISTICS and SEARCH_STATUS are returned by various subprograms in this package.

4.6. Print_Log_Handler Exported Types, Data, and Exceptions

The following Ada code fragments show the detail of the type, data, and exceptions exported by the package Primitive_Citation_Handler.

```
PRINT_LOG_CREATION_ERROR : exception;
```

4.7. Other CSCs

The other CSCs in the Ada LRM Reader do not export types, data, or exceptions. These CSCs are:

1. Command_Dispatcher
2. Citation_Handler
3. LRM_Reader
4. Make_Cit
5. Make_DAF

5. CSCI data files

5.1. Data file to CSC/CSU cross reference

The following table shows the classes of data files used in the Ada LRM Reader and the CSCs associated with them. Note that DAF stands for Direct Access File.

Data File	*Associated CSCs*	*Relationship*
*.doc	Input_File	Package Input_File reads the *.doc files
*.daf	DAF_Handler	Package DAF_Handler creates and reads the *.daf files
lrm.log	Print_Log_Handler	Package Print_Log_Handler creates the lrm.log file

5.2. *.doc Files

The *.doc files are ASCII text files. The lines of these files vary in length from 0 to 102 characters (determined by examination). If they are in the UNIX file format, each line is terminated by a line feed character. If they are in the MSDOS file format, each line is terminated by a carriage return/line feed pair. If they are in the VAX/VMS file format, each line is not terminated.

The *.doc files comprise the source text to the Ada LRM and to the HELP and ABOUT citations. The following table lists these files by name. Note that under UNIX, all file names must be lower-case.

Chapters	*Appendices*	*Special*	*HELP and ABOUT*
chap01.doc	chapaa.doc	cahpco.doc	chaphe.doc
chap02.doc	chapab.doc	chapfo.doc	chapxx.doc
chap03.doc	chapac.doc	chapin.doc	
chap04.doc	chapad.doc	chappo.doc	
chap05.doc	chapae.doc		
chap06.doc	chapaf.doc		
chap07.doc			
chap08.doc			
chap09.doc			
chap10.doc			
chap11.doc			
chap12.doc			
chap13.doc			
chap14.doc			

As delivered from the Ada Software Repository, these *.doc files contain page numbers and an excessive number of blank lines. The *.doc files have been modified to remove the page numbers.

5.3. *.daf Files

The *.daf files correspond to the *.doc files, but the information the *.daf files contains is in a DAF record structure to facilitate random access and reduced overhead in the screen display functions. Each record in the *.daf files is of type LINE, defined as:

```
type LINE_TYPE is (NORMAL, SECTION, UNUSED);

type LINE is record
  Str       : STRING (1 .. SYSDEP.Screen_String_Length);
  Str_Last  : NATURAL := 0;  -- index of last char in Str
  Kind      : LINE_TYPE := NORMAL;
end record;
```

LINE records in the *.daf files have only two values in their Kind fields: (1) SECTION, which indicates that this line starts a new section, or citation, of the Ada LRM, and (2) NORMAL, which indicates that this line is simply a part of the current section, or citation.

Since the *.doc files contained an excessive number of blank lines, the Make_DAF Procedure looks for duplication in sequence of blank lines and removes groups larger than 3, leaving the *.daf files with much fewer blank records.

5.4. lrm.log File

The lrm.log file is composed of ASCII text lines, ranging in length from 0 to 80 characters. It contains groups of text: (1) the text of one or more entire citations and (2) the text of one or more single screens from one or more citations. Prologues are placed into the lrm.log file before each citation or screen. The prologue for an entire citation looks like this:

```
----------------------------------------------------
-- Citation: 4.1
```

The prologue for a single screen of a citation looks like this:

```
----------------------------------------------------
-- Citation: 4.1
-- Screen Number: 2
```

The entries in the lrm.log file are derived from the citations in the *.daf files in response to the user issuing PRINT and PS commands.

6. Requirements traceability

The following Requirements Traceability Matrix provides a mapping between the SRS and this SDD:

SRS Section	*SDD Sections*			
3.1	3.2.8			
3.2	3.2.6			
3.2.1	3.2.6	3.2.9	3.2.10	
3.2.2	3.2.6	3.2.11		
3.2.3	3.2.6	3.2.9	3.2.10	
3.2.4	3.2.6	3.2.9	3.2.10	
3.2.5	3.2.6			
3.3	3.2.1	3.2.2	3.2.8	3.2.11
3.4	3.2.2	3.2.7	3.2.8	3.2.11
3.5	3.2.4	3.2.8		
3.5.1	3.2.4			
3.5.2	Implementation not required			
3.6	3.2.2	3.2.3	3.2.5	3.2.7
3.7	3.2.6			
3.8	3.2.8			
3.9	Implementation not required			

7. Notes

Acronyms

Acronym	Meaning
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CSU	Computer Software Unit
DAF	Direct Access File
I/O	Input/Output
OID	Object Interaction Diagram
SDD	Software Design Document
SDP	Software Development Plan
SRS	Software Requirements Specification
STD	State Transition Diagram
SUM	Software User's Manual