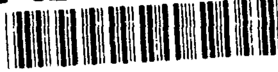
AD-A240 602



(2)

Carnegie Mellon University
Software Engineering Institute

DTIC
ELECTE
SEP 23 1991
S D
D

Saddle User's Guide

This document has been approved for public release and sale; its distribution is unlimited.

91-11239

System for User Interface Development

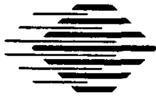
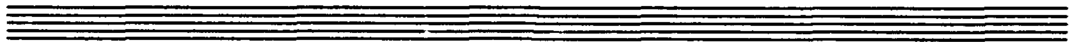
Version 1

Date April 1991

User's Guide

CMU/SEI-91-UG-3
April 1991

Serpent: Saddle User's Guide

NTIS
DTIC
By
Dist
A-1

User Interface Project

Approved for public release.
Distribution unlimited.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

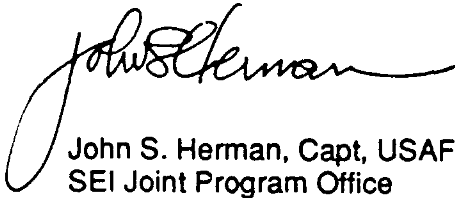
SEI Joint Program Office
ESD/AVS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER



John S. Herman, Capt, USAF
SEI Joint Program Office

The Software Engineering Institute is sponsored by the U.S. Department of Defense.
This report was funded by the U.S. Department of Defense.

Copyright © 1991 by Carnegie Mellon University.

This document is available through the Defense Technical Information Center. DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center, Attn: FDRA, Cameron Station, Alexandria, VA 22304-6145.

Copies of this document are also available through the National Technical Information Service. For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Table of Contents

1 Introduction **1**
1.1 This Manual 1
1.1.1 Organization 1
1.1.2. Typographical Conventions 1
1.2 Other Serpent Documents 2

2 Overview **4**
2.1 How Saddle Relates to Serpent 4
2.2 Serpent Architecture 6
2.3 Shared Data Concepts 7

3 Specifying Shared Data Types in Saddle **9**
3.1 Language Specifics 9
3.2 How Constructs Interrelate 9
3.3 Using Shared Data Types 11

4 Using the Saddle Processor **12**
4.1 Task 1: Setting Up 12
4.2 Task 2: Executing Saddle 13
4.3 Results 14
4.3.1 Nominal Messages 14
4.3.2 Output Files 15
4.3.3 Error Messages 15
4.3.4 Other Errors 16
4.3.5 File Naming Errors 17

Appendix A A Formal Saddle Specification **18**

List of Figures

Figure 1-1 Serpent Documents 3
Figure 2-1 Dialogue Specification in Serpent 4
Figure 2-2 Serpent Architecture 6

List of Examples

Example 2-1	Shared Data Definition, Elements, and Components	8
Example 3-1	Use of ID type to define a list	11

List of Tables

Table 3-1 Saddle Construct Equivalents for C 10

1 Introduction

Serpent is a user interface management system (UIMS) that supports the development and execution of the user interface of a software system from the prototyping phase through production and maintenance. Serpent encourages a separation of functionality between the user interface and application portion of a system. Serpent is also easily extended to support additional input/output (I/O) toolkits.

1.1 This Manual

This manual describes Saddle—the language used to specify interfaces between an application and Serpent. Included are how to create Saddle files (files that define the data shared by the application and the user interface) and how to execute Saddle. Readers should be familiar with general UIMS concepts, have a working knowledge of programming languages, and understand the concepts described in the *Serpent Overview* (CMU/SEI-91-UG-1) and *Serpent: System Guide* (CMU/SEI-91-UG-2).

1.1.1 Organization

This guide contains the following chapters and appendix:

- **Introduction:** An overview of Saddle, including its relation to Serpent and the basic concepts of shared data necessary to use Saddle.
- **Specifying shared data types in Saddle**: A description of the activities involved in creating Saddle files, including the data types Saddle supports.
- **Using the Saddle Processor**: Procedures for using the Saddle processor.
- **Appendix A: A Formal Specification of Saddle:** in Backus-Naur Form (BNF)

1.1.2. Typographical Conventions

The following conventions are observed in this manual.

Code examples	`Courier typeface`
Variables, attributes, etc.	`Courier typeface`
Syntax	`Courier typeface`
Warnings and Cautions	***Bold, italic statements***

1.2 Other Serpent Documents

The following documents provide information about the Serpent system

Serpent Overview
Introduces the Serpent system.

Serpent: System Guide
Describes installation procedures, specific input/output file descriptions for intermediate sites and other information necessary to set up a Serpent application.

Serpent: Dialogue Editor User's Guide
Describes how to use the editor to develop and maintain a dialogue.

Serpent: Slang Reference Manual
Provides a complete reference to Slang, the language used to specify a dialogue.

Serpent: C Application Developer's Guide
Serpent: Ada Application Developer's Guide
Describe how the application interacts with Serpent. These guides describe the runtime interface library, which includes routines that manage such functions as timing, notification of actions, and identification of specific instances of the data.

Serpent: Guide to Adding Toolkits
Describes how to add user interface toolkits such as various Xt-based widget sets to Serpent or to an existing Serpent application. Currently, Serpent includes bindings to the Athena Widget Set and the Motif Widget Set.

The following figure shows Serpent documentation in relation to the Serpent system:

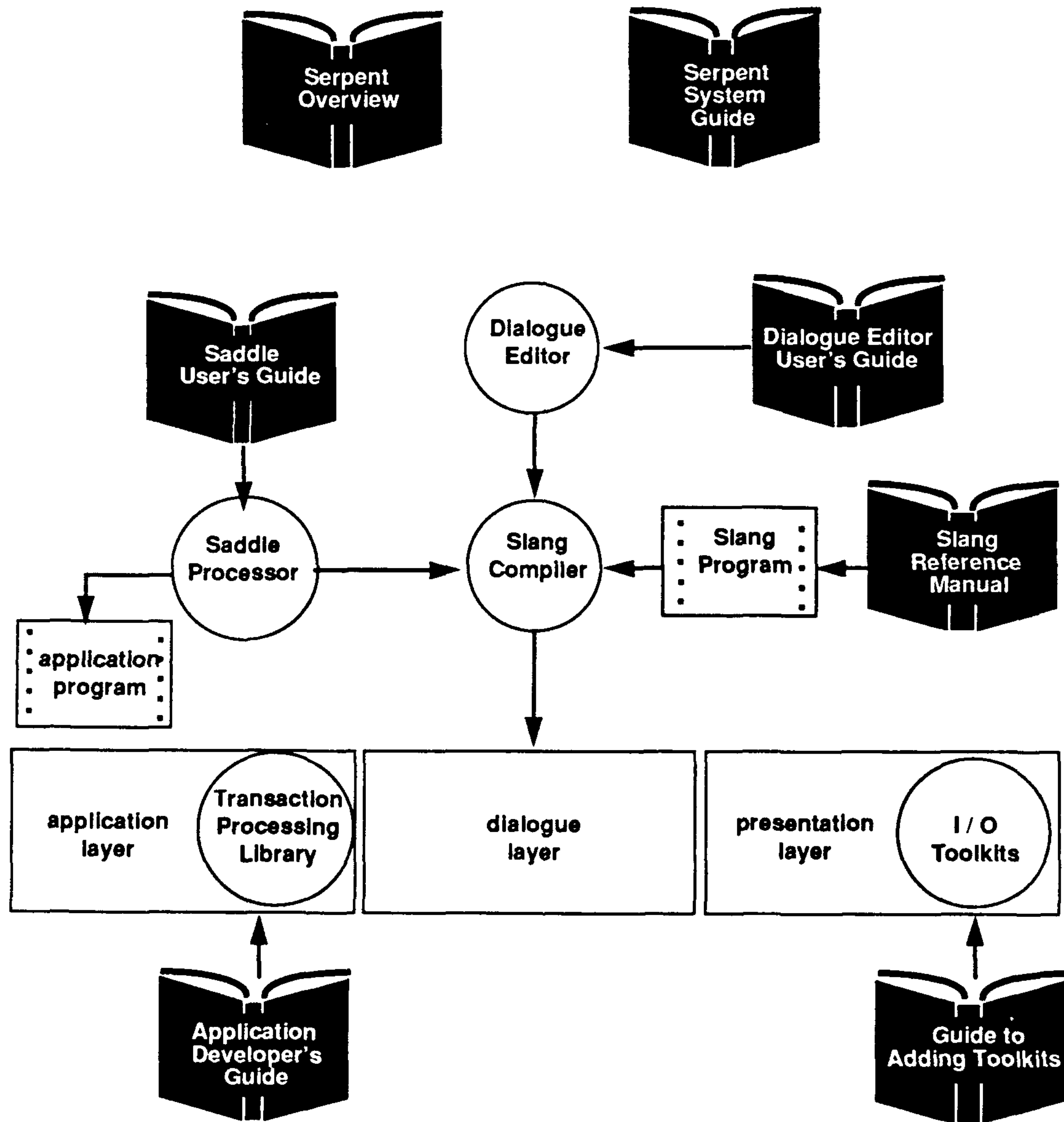


Figure 1-1 Serpent Documents

2 Overview

This overview provides general information about Saddle and shared data. Subsequent chapters in this guide present more detailed information about using Saddle.

2.1 How Saddle Relates to Serpent

Saddle is the part of Serpent that provides a clean separation between the user interface and application portion of a system. Figure 2-1 illustrates how Serpent is used in the specification process.

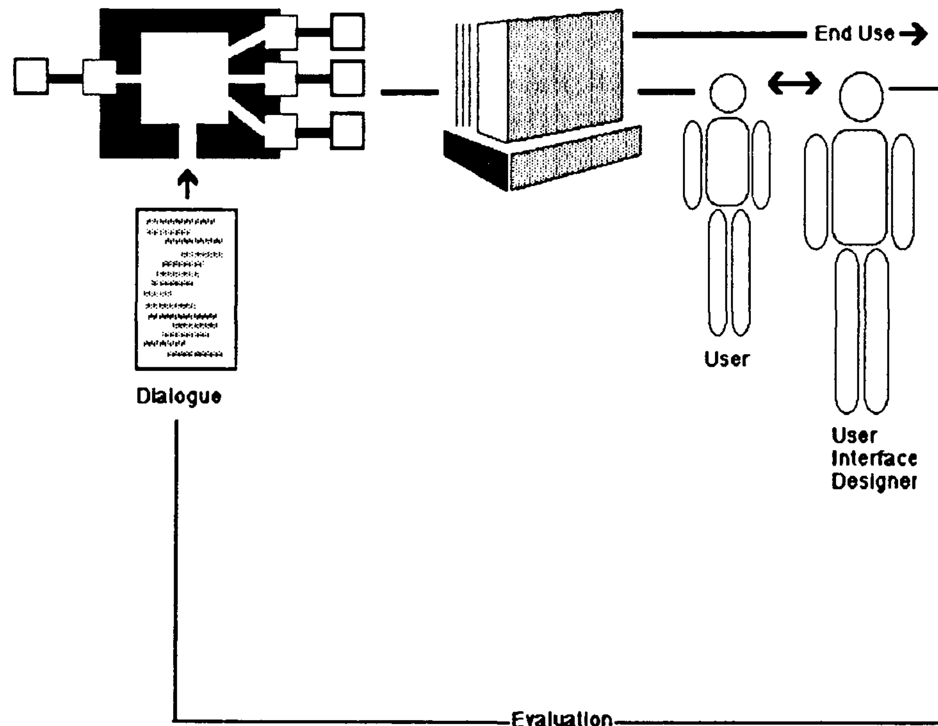


Figure 2-1 Dialogue Specification in Serpent

The designer of a system makes fundamental decisions about the content of the information communicated to and from the end user. Decisions must also be made about which tasks belong to the functional portion of the application and which tasks belong to the user interface. As a general rule, those tasks that generate information (regardless of the form) to be communicated to the end user belong in the functional portion, and those tasks concerned with the form of the information belong on the user interface side. For example, the functional portion of the application should not be concerned with the language understood by the end user.

The interface between the functional portion of the application system and Serpent is called *application shared data*. The interface between the presentation layer and the dialogue layer is called *toolkit shared data*. Saddle is the language through which this shared data is specified (prior to using other portions of Serpent). The Saddle specification is processed by Saddle, and the output of the specification is then used by the other portions of Serpent. The dialogue layer also uses Saddle to describe portions of its local data. This data is called *dialogue shared data.*

Using the Serpent dialogue editor, the user interface developer specifies in a *dialogue* the user interface for the application. Using any methods, the developer of the functional portion of the application system completes the design and the implementation of the portion of the application that does not involve the user interface. The system is now ready for execution; the end user will see a complete system, the user interface and the functionality of the application.

2.2 Serpent Architecture

The Serpent architecture has three layers, as shown in Figure 2-2.

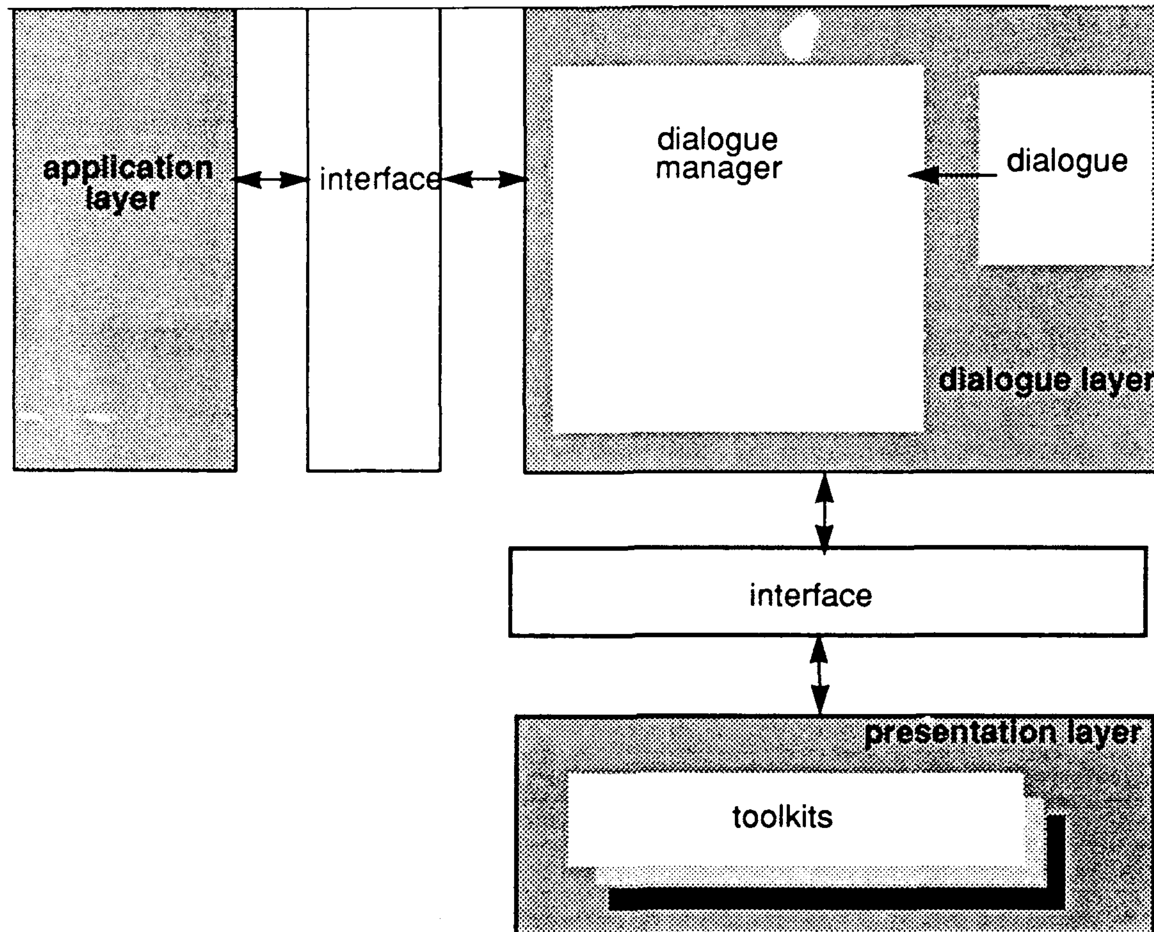


Figure 2-2 Serpent Architecture

1. **Presentation layer.** This layer controls the end user interactions, providing low level feedback. A collection of interactive primitives, or *objects*, reside at the presentation level; the objects available depend upon the input/output toolkit used to present the user interface. For a menu object, for example, the presentation level displays the menu, highlights the current entry when the cursor is within the boundaries of the entry, and detects the particular entry selected by the user.

2. **Dialogue layer.** The dialogue layer determines what information is currently available to the user and the form of that information. While the presentation level manages the presentation, the presentation is specified at the dialogue level by the user interface developer in a *dialogue*. Upon compilation, the *dialogue* is transformed into the dialogue manager. In the menu example, the presentation level manages the menu object, while the dialogue manager tells the presentation level the position and the contents of the menu and acts upon the menu selection that the user makes.

3. **Application layer.** This layer performs those functions that are specific to the application. The application should be written to be independent of not only the presentation but also of the media by which the presentation is delivered. That is, because the presentation and dialogue layers are designed to handle all of the user interface details, the application need only provide information (in its own terms) for the user. The form or media in which that information is presented should be independent of this layer. For example, the application information could be presented visually or aurally; input could come from a keyboard or voice.

2.3 Shared Data Concepts

Shared data is the means by which different parts of Serpent communicate. The information to be communicated is specified using Saddle, which is presented in detail in the chapters following this section.

The basic steps in processing Saddle are:

1. Set up Saddle.
2. Execute Saddle.
3. Generate files.

Setting up Saddle consists of specifying shared data types in a shared data definition (*sdd*) file. The files containing shared data are the interface through which an application communicates with Serpent (*application shared data*). It is also the means by which input/output toolkits and Serpent communicate *(technology shared data*). (Both the toolkit application and the dialogue manager use the same specification of shared data in a given Serpent execution.) It also specifies some dialogue local data (*dialogue shared data*).

Shared data consist of shared data *elements*, which are objects in the interface. Shared data elements can be *simple* or *complex*, and have attributes called *components*. Instances of elements are called *ids*, occur at runtime, and are managed by the runtime interface library. Example 2-1 illustrates a shared data definition, including elements, and components.

```
example: shared data
 a_record_name:   record

   a: integer;
   d: string(80);

 end record;
end shared data;
```

Example 2-1 Shared Data Definition, Elements, and Components

The shared data elements defined in Example 2-1 is `a_record_name`. Within the record specification, `a` and `d` are components: `a` is defined as an integer component, whereas `d` is an 80 character string.

3 Specifying Shared Data Types in Saddle

For the application to communicate with Serpent, the application's shared data interface must be specified. Saddle provides constructs for the specification of several shared data types. This chapter describes the characteristics of Saddle construct.

3.1 Language Specifics

Saddle types can be *simple* or *complex.*

The characteristics of simple types are:

- INTEGER: four bytes, e.g., 6, -1, 4273; the range of values available in an integer is from -2^{31} to $2^{31}-1$.
- REAL: floating point, not fixed point; eight bytes, e.g., -3.0, 75.385; the precision available in a real number is approximately 15 significant figures.
- STRING: multiple characters treated as a single entity; the length specified in the Saddle declaration is the actual number of characters in the string; if not specified, the length is assumed to be 1.
- BUFFER: a set of (conceptually) continuous bytes of arbitrary length; buffers consist of a size (in bytes), the type of the data, and the address of the actual data.
- ID: an identification of an instance of a shared data element; this is the mechanism used within Saddle to construct more complicated structures.
- BOOLEAN: may hold the value TRUE or FALSE.

A *RECORD* is a collection of simple constructs, conceptually organized as a single row in a table.

3.2 How Constructs Interrelate

A shared data *element* is equivalent to a shared data type. The element is made up of *components.*

Components are not Saddle types. This means that simple constructs within a RECORD represent fields within the RECORD, not types. For each Saddle construct, Table 3-1 shows the equivalent construct in the C language under normal circumstance. Further formal specifications for Saddle are in Appendix A.

Saddle	C
a: RECORD;	typedef struct {
END RECORD;	} a;
b: INTEGER;	int b;
c: REAL;	double c;
d: STRING (#)[1]	char d (#+1)[4];
e: BUFFER;	buffer e[5];
f: ID (OF) name[2];	id_type f[5];
g: BOOLEAN;	boolean g;
h: SHARED DATA	-
END SHARED DATA;	-
! <text><eol>[3]	/* <text> */

Table 3-1 Saddle Construct Equivalents for C

Notes

1. Square brackets ([]) or curly brackets ({}) may be substituted for parentheses in this construct, but they must match; for example, using an open parenthesis with a corresponding close square bracket is illegal; if no length is specified, it is assumed to be 1.

2. In the ID construct, the *name* is optional; if *name* is used, the word OF is optional.

3. The exclamation point starts a comment; the term *eol* indicates the end of line mark, or carriage return.

4. Note that the size of the array in C is 1 byte greater than it is in Saddle; this is because the Saddle string is treated like the C *string*, which requires an extra byte for the string termination character ('\0').

5. The buffer and id_type are specific Serpent types that are already defined.

3.3 Using Shared Data Types

The shared data types defined in Saddle are intended for specific uses: Saddle is a method of declaring the structure of values to be passed across shared data. The actual values are assigned at runtime through the use of a library described in *Serpent: C Application Developer's Guide* and in *Serpent: Ada Application Developer's Guide*. The intended use of the shared data types is as follows:

- INTEGER carries a value that is a whole number.
- REAL carries a value containing a decimal point.
- STRING carries a number of characters.
- BUFFER carries undifferentiated byte strings. These can be strings of any length, or they can be encoded byte strings for use either by the dialogue or by one of the input/output technologies integrated into Serpent. Built-in functions within the dialogue language enable a buffer to be decoded by a dialogue (see *Serpent: Dialogue Editor User's Guide*).
- ID allows the construction of complex structures. Example 3-1 shows a list construct in Saddle. Each node in the list is represented by an instance of the record type `list_example`. The `value` component will have the value of the element in the list and the `next` component will have the value of the identification assigned at runtime to the next instance of `list_example`. It is the responsibility of the application program to retrieve the identification and place it into the correct instance of `list_example`.
- BOOLEAN carries a value that is either 1 (TRUE) or 0 (FALSE).

```
list: shared data
 list_example:   record
       value: integer;
         next: ID OF list_example;
 end record;
end shared data;
```

Example 3-1 Use of ID type to define a list

4 Using the Saddle Processor

Using Saddle entails two major tasks: setting up and executing.

4.1 Task 1: Setting Up

Setting up Saddle for use requires the creation of a shared data definition (sdd) file. Any text editor can be used to create this file.

Example 4-1 illustrates a sample sdd file.

```
<< execution string >>

test_name: shared data

! there is only one record in this sdd

    the_record: record
                  a: integer;!this is an
integer specification
         b: real;
         c: string;   !a string of length 1
         d: string (212);!a string of length
212
         e: buffer;
         f: id of the_record;
         g: boolean;
 end record;

 end shared data;
```

Example 4-1 Sample Sdd File

In Example 4-1, notice that shared data is defined to the system by a `shared data` statement (the first line), along with a corresnonding `end shared data` statement (the last line). All the statements between those two make up the *shared* data *definition.* All `record` declarations define shared data elements; all names declared within elements define components. If the shared data defines dialogue shared data, the first line must be dialogue shared data.

Example 4-1 also illustrates the different types of *declarations*, along with their syntax. Notice the use of comments, which open with an exclamation point and close with an end-of-line: `!there is only one record in this sdd` is a comment.

The comment line, `!this is an integer specification`, is an example of an *inline* comment.

Finally, Example 4-1 contains Saddle *reserved* words such as `integer`, `real` and `string`.

For practice, enter Example 4-1 into a file named **test.sdd** in your current directory. Note that Saddle allows the following:

- Combining upper and lower case, since Saddle is not case-sensitive.
- Entering any number of tabs, spaces, or carriage returns between any two items on a line, since Saddle is not position-sensitive.
- Placing comments after any semicolon or at the beginning of a line.

(**Note**: The *execution string* is the actual string of characters that must be keyed in to execute an application. For now, anything between the double angle brackets (<< and >>) is legal; to create a real sdd with its corresponding application, *execution string* must be replaced with the actual execution command for that application.)

4.2 Task 2: Executing Saddle

Once the .sdd file exists, it can be executed with the following command:

sdd file_name <CR>

where file_name is the first part of the complete name of the sdd file. Thus, in Example 4-1, file_name is test, and entering

sdd test <CR>

executes Saddle, using the file test.sdd as input. Note that Saddle assumes that the file is an sdd file. The full name of the file is also legal:

sdd test.sdd <CR>

as are absolute or relative pathnames. For example,

sdd../afile <CR>

executes Saddle, using the file afile.sdd in the current working directory's parent as input.

If the application program is written in C,

add -c test <CR>

results in the creation of a file (test.h) for inclusion into the application program. (This is the default.)

If the application program is written in Ada,

sdd -a test <CR>

results in the creation of a package specification (test.a) for inclusion into the application program.

4.3 Results

Results from Saddle depend on whether or not execution is successful. Results can be in the form of nominal messages, output files, and error messages.

4.3.1 Nominal Messages

Upon starting Saddle, a stream of messages are generated and sent to the UNIX logical name **stdout**. Normally **stdout** appears on the user's terminal; however, it can be redirected to another file. For messages described in this subsection, assume that the following command has been entered:

sdd test <CR>

The first message will be:

Welcome to Saddle version x.xx

There will also be a message indicating which file is being read as input, in this case:

Reading from test.sdd

Any of several possible error messages could be displayed now (see Section 4.3.3 following this section), some of which are fatal. If no error messages are displayed, the message

Saddle completed successfully

will be displayed, followed by the system prompt.

4.3.2 Output Files

Of the files generated upon successful execution of Saddle, some are persistent and some are not. Persistent files are placed in the directory containing the .sdd file. The three persistent files Saddle generates automatically are:

1. **test.h**, a C include file (if you use the -c option or no option)
2. **test.a**, an Ada package specification file (if you use the -a option)
3. **test.ill**, a special file for internal Serpent use

The three files contain declarations specific to their respective languages, called *Saddle target languages*, equivalent to their respective declarations. All of these files are created in the same directory as the original .sdd file, regardless of whether absolute or relative pathnames are used. The last file, the .ill file, contains declarations made in a Serpent specific language, ILL (for Interface Layer Language).

Saddle generates other files that are normally temporary: if Saddle runs normally, those files are deleted automatically. Saddle uses these files to generate a program that, in turn, automatically generates the ill file. Once the ill file has been generated, the files are removed from the system.

4.3.3 Error Messages

The two general types of errors in Saddle relate to either naming or syntactic aspects.

The first class deals with naming previously declared shared data objects. A fatal error occurs if any two element names in a single shared data definition match, since element names must be unique. When this condition occurs, Saddle prints out:

Saddle detected a FATAL ERROR...

When this naming error occurs, simply rename the offending name or names, and rerun Saddle.

Note: Testing or matching the names is not case-sensitive in Saddle. For example, to determine if two names match, Saddle first converts them to lower case and then compares them. If the names match, a fatal error occurs. Saddle handles the comparison to accommodate target languages (such as Ada) that are not case-sensitive.

This rule also applies to components within a single record: an error occurs if any two components within a single record have the same name.

The second class of fatal errors concerns the syntax of the Saddle language. Saddle is made up of a front end, which reads the .sdd file, parses it, and stores the declarations in an internal form and several back ends that generate its output files from the internal form. While the back ends are written in the C language, the front end is structured around two UNIX programs known as *lex* and *yacc*. Between the two programs, all Saddle input is read, analyzed, and checked against the formal grammatical description for Saddle.

If at any time the resultant grammatical analyzer detects items that are out of place, a yacc error-handling routine is invoked. This routine is relatively limited in scope and in the quality of information it provides; furthermore, it stops all parsing and subsequent analysis. Therefore, the Saddle error handling routine intercepts yacc and lex errors. This serves the dual purpose of making all Saddle error messages look similar and allowing sdd file analysis to continue. Hence, if the message

YACC detected a FATAL ERROR: syntax error

appears, most likely it is due to a mistake detected in the syntax of a line. Yacc also generates a syntax error if you use a name that matches the characters of a reserved word (see Appendix A for a list of reserved words). Occasionally, errors may arise from using streams of characters that the Saddle grammar cannot recognize; these rare errors are usually caused by one or more illegal characters.

Saddle may not be able to determine how to realign the grammar rules with the input stream from the sdd file; this depends on the point where the yacc error is detected. For this reason, sometimes Saddle stops parsing and analyzing of the input stream. When this happens, Saddle may not exit cleanly, which may cause a core dump If this happens, the dump can be deleted by typing

rm core <CR>

Although removing the core dump is not necessary, it is recommended because the core dump usually takes up a lot of space.

For details on the formal Saddle grammatical description, refer to Appendix A.

4.3.4 Other Errors

After the welcome message, if the following message appears:

could not open file_name.sdd

either the named .sdd file does not exist in the specified directory, or you do not have read access to the named .sdd file. Check the name of the .sdd file, check its access rights, and rerun Saddle.

4.3.5 File Naming Errors

The file name dm.sdd is reserved for use with dialogue manager shared data only. If it is used and the dialogue key word is not specified, a warning message results.

If any other messages appear, check with your system administrator.

Appendix A A Formal Saddle Specification

The following is the formal specification of Saddle in Backus-Naur Form (BNF).

```
saddle_spec =
  "<<" exec_string ">>" comment data_definition

data_definition =
  name ":" [dialog /dialogue] SHARED DATA
  type_type {record_type}
  END SHARED DATA eos

record_type =
  name ":" RECORD
  simple_type {simple_type}
  END RECORD eos

type_choice =
   BOOLEAN
  | INTEGER
  | REAL
  | STRING [string_size]
  | BUFFER
  | ID [[OF] name]

string_size =
  "(" positive_integer ")"
  |"{" positive_integer "}"
  |"[" positive_integer "]"

positive_integer =
  decimal_character {decimal_character}

name =
  alphabetic_character {any_character}

any_character =
  alphabetic_character
  |decimal_character
  |  "_"

eos =
  ";" {comment}

comment =
  "!" line_of_text eol
```

Notes

1. Upper case denotes a literal (reserved word), although Saddle is not case-sensitive.

2. Lower case denotes a BNF rule.

3. A space between two items in a rule indicates separation, that is, lexically different items. A carriage return or a tab between two items in a rule has the same meaning as a space.

4. A vertical bar (|) indicates choice, that is, one of the choices separated by vertical bars must be used.

5. Items enclosed in square brackets ([]) are optional.

6. Items enclosed in curly brackets ({}) may be executed any number of times, including zero.

7. Items enclosed in double quotes (" ") are punctuation literals.

8. The maximum length for a name is 32 characters. If a name exceeds 32 characters, Saddle truncates it to 32. Reserved words are not allowed as names. The following are reserved words:

BOOLEAN	INTEGER
BUFFER	OF
DATA	REAL
END	RECORD
ID	SHARED
INCLUDE	STRING

9. Saddle allows multiple spaces between items. A tab or carriage return between two Saddle items has the same meaning as a space.

10. The `exec_string` is the actual string of characters that must be keyed in to execute the application for which this sdd was created.

11. If no length is specified for a `string_type`, it is assumed to be 1.

12. An `id_type` may be used only to identify an element.

13. Note that the end of the line ends comments.

A
Application
  layer 7
  shared data 5, 7

B
BOOLEAN 9, 11
BUFFER 9, 11

C
Components 7, 9

D
Declarations 12
Dialogue 7
Dialogue editor 5
Dialogue layer 7
Dialogue shared data 5, 7
Documentation 2, 3

E
Execution string 13

I
ID 7, 9, 11
INTEGER 9, 11
Interface Layer Language file (.ill file) 15

P
Presentation layer 6

R
REAL 9, 11
RECORD 9
Reserved words 13

S
Saddle 1
Saddle target languages 15
Serpent 6
  documents 2
Shared data 4, 7, 12
  concepts 7
  definition 12
  element 7, 9
  types 11
Shared data element 9
STRING 9, 11

T
Technology shared data 7
Toolkit shared data 5

UNLIMITED, UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY N/A 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited
4. PERFORMING ORGANIZATION REPORT NUMBER(S CMU/SEI-91-UG-3	5. MONITORING ORGANIZATION REPORT NUMBER(S) CMU/SEI-91-UG-3
6a. NAME OF PERFORMING ORGANIZATION Software Engineering Institute 6b. OFFICE SYMBOL (if applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI Joint Program Office
6c. ADDRESS (City, State and ZIP Code) Carnegie Mellon University Pittsburgh PA 15213	7b. ADDRESS (City, State and ZIP Code) ESD/AVS Hanscom Air Force Base, MA 01731
8a. NAME OFFUNDING/SPONSORING ORGANIZATION SEI Joint Program Office 8b. OFFICE SYMBOL (if applicable) ESD/AVS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962890C0003
8c. ADDRESS (City, State and ZIP Code) Carnegie Mellon University Pittsburgh PA 15213	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO: 63752F PROJECT NO.: N/A TASK NO: N/A WORK UNIT NO.: N/A

11. TITLE (Include Security Classification)
Serpent· Saddle User's Guide

12. PERSONAL AUTHOR(S)
SEI User Interface Project

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Final	FROM TO	April 1991	

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (Continue on reverse of necessary and identify by block number)
Serpent, UIMS, user interface management system, user interface generators, Saddle

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

Serpent is a user interface management system (UIMS) that supports the development and implementation of user interfaces, providing an editor to specify the user interface and a runtime system that enables communication between the application and the end user. This document describes Saddle, the language used to specify interfaces between an application and Serpent. Included is informati on on creating Saddle files (files that define the data shared by the application and the user interface) and executing Saddle.

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ■ SAME AS RPT DTIC USERS ☐ ■	21. ABSTRACT SECURITY CLASSIFICATION Unclassified, Unlimited Distribution
22a. NAME OF RESPONSIBLE INDIVIDUAL John S. Herman, Capt, USAF	22b. TELEPHONE NUMBER (Include Area Code) (412) 268-7630 22c. OFFICE SYMBOL ESD/AVS (SEI

DD FORM 1473, 83 APR EDITION of 1 JAN 73 IS OBSOLETE UNLIMITED, UNCLASSIFIED SECURITY CLASSIFICATION OF THIS

ABSTRACT —continued from page one, block 19