

AD-A083 267

VIRGINIA POLYTECHNIC INST AND STATE UNIV BLACKSBURG --ETC F/8 9/2
DIALOG. A SIMULA CLASS FOR WRITING INTERACTIVE PROGRAMS. (U)
MAY 79 R J OR6ASS

UNCLASSIFIED

VPI-TM-79-3

AFOSR-TR-80-0299

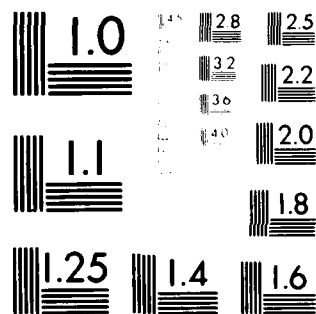
AFOSR-79-0021

NL

1 1 1
2016
10/1/86



END
DATE
FILMED
5-80
DTIC



MICROCOPY RESOLUTION TEST CHART
 NATIONAL BUREAU OF STANDARDS-1963-A



AFOSR-TR. 80-0299

LEVEL

EXTENSION DIVISION

VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE
GRADUATE PROGRAM IN NORTHERN VIRGINIA

(10)
P.S.

P. O. Box 17186
Washington, D. C. 20041
(703) 471-4600

DIALOG

A SIMULA* CLASS FOR WRITING INTERACTIVE PROGRAMS†‡

Richard J. Orgass

Technical Memorandum No. 79-3

May 4, 1979

ABSTRACT

A SIMULA class containing procedures for easily writing programs that interact with a user by asking questions at run time and which dynamically name and open files at run time is described. The class uses properties of IBM SIMULA that are not available in other implementations. It also depends on the EBCDIC character codes rather than ASCII but it is assumed that a user's terminal is an ASCII terminal.

Keywords and Phrases: SIMULA, interactive programming

CR Categories: 4.22, 4.49, 4.39

-
- * SIMULA is a registered trademark of the Norwegian Computing Center, Oslo, Norway.
 - † Research sponsored by the Air Force Office of Scientific Research, Air Force Systems Command, under Grant No. AFOSR-79-0021. The United States Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation hereon.
 - ‡ The information in this document is subject to change without notice. The author, Virginia Polytechnic Institute and State University, the Commonwealth of Virginia and the United States Government assume no responsibility for errors that may be present in this document or in the program described here.

DDC FILE COPY

DTIC
ELECTE
S AFR 1 1980 D

80 4 21 012

Approved for public release;
distribution unlimited.

Located at Dulles International Airport—400 West Service Road

Copyright, 1979

by

Richard J. Orgass

General permission to republish, but not for profit, all or part of this report is granted, provided that the copyright notice is given and that reference is made to the publication (Technical Memorandum No. 79-3, Department of Computer Science, Graduate Program in Northern Virginia, Virginia Polytechnic Institute and State University), to its date of issue and to the fact that reprinting privileges were granted by the author.

INDEX

```

Appendix . . . . . 14
boolean_request . . . . . 4
Breakoutimage . . . . . 12
conc2 . . . . . 12
dd_name . . . . . 9
first_token . . . . . 11
find_infile . . . . . 7
find_outfile . . . . . 8
frontstrip . . . . . 11
integer_request . . . . . 5
initialize_terminal . . . . . 9
is_integer . . . . . 11
Letter . . . . . 11
nextfile . . . . . 12
no_blanks . . . . . 11
rest . . . . . 11
restore_terminal . . . . . 9
text_request . . . . . 2
upcase . . . . . 11

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
84

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM | | | | | | | | | | |
|--|--|---|-----------------------------|--|---------------|--|---------|--|-------------|--|---------------|--|
| 1. REPORT NUMBER (18) AFOSR-TR-80-0299 | 2. GOVT ACCESSION NO. (19) AD A083 267 | 3. RECIPIENT'S CATALOG NUMBER | | | | | | | | | | |
| 4. TITLE (and Subtitle) (6) A SIMULA CLASS FOR WRITING INTERACTIVE PROGRAMS (7) DIALOG | | 5. TYPE OF REPORT & PERIOD COVERED Interim | | | | | | | | | | |
| 7. AUTHOR(s) (10) Richard J. Orgass | | 6. PERFORMING ORG. REPORT NUMBER | | | | | | | | | | |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS Virginia Polytechnic Inst. & State University Department of Computer Science/ Washington, DC 20041 | | 8. CONTRACT OR GRANT NUMBER(s) (15) AFOSR-79-0021 | | | | | | | | | | |
| 11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research/NM Bolling AFB, Washington, DC 20332 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (16) 61102F (17) 2304/A2 | | | | | | | | | | |
| 14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (13) 242 (14) VPI-TM-79-3 | | 12. REPORT DATE (11) 4 May 79 | | | | | | | | | | |
| | | 13. NUMBER OF PAGES 23 | | | | | | | | | | |
| | | 15. SECURITY CLASS. (of this report) UNCLASSIFIED | | | | | | | | | | |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE | | | | | | | | | | |
| 16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited. (9) Technical memo. | | | | | | | | | | | | |
| 17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) <table border="1"> <tr><td>Approved for public release</td><td></td></tr> <tr><td>NTIS GPO</td><td></td></tr> <tr><td>DOC TAB</td><td></td></tr> <tr><td>Unannounced</td><td></td></tr> <tr><td>Justification</td><td></td></tr> </table> | | | Approved for public release | | NTIS GPO | | DOC TAB | | Unannounced | | Justification | |
| Approved for public release | | | | | | | | | | | | |
| NTIS GPO | | | | | | | | | | | | |
| DOC TAB | | | | | | | | | | | | |
| Unannounced | | | | | | | | | | | | |
| Justification | | | | | | | | | | | | |
| 18. SUPPLEMENTARY NOTES <table border="1"> <tr><td>By</td><td></td></tr> <tr><td>Distribution/</td><td></td></tr> </table> | | | By | | Distribution/ | | | | | | | |
| By | | | | | | | | | | | | |
| Distribution/ | | | | | | | | | | | | |
| 19. KEY WORDS (Continue on reverse side if necessary and identify by block number) SIMULA, interactive programming <table border="1"> <tr><td>Subject</td><td></td></tr> <tr><td>Index</td><td></td></tr> </table> | | | Subject | | Index | | | | | | | |
| Subject | | | | | | | | | | | | |
| Index | | | | | | | | | | | | |
| 20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A SIMULA class containing procedures for easily writing programs that interact with a user by asking questions at run time and which dynamically name and open files at run time is described. The class uses properties of IBM SIMULA that are not available in other implementations. It also depends on the EBCDIC character codes rather than ASCII but it is assumed that a user's terminal is an ASCII terminal. 421098 | | | | | | | | | | | | |

1. Problem Statement

A continuing problem in the design of genuinely interactive programs is that it is quite painful to write code for reliably and meaningfully interacting with the user sitting at his terminal. This is true in all time sharing systems and is a particularly severe problem in the CMS system because the terminal interface provided by the operating system is quite crude.

The minimal sequence of events that must occur when a user is asked to provide input may be described as follows:

- (1) Prompt the user with a question.
- (2) Read the answer.
- (3) Check the answer to make sure that it is an acceptable answer.
- (4) If the answer is unacceptable, provide a corrective message and ask the question again.

There are two other properties of a question asking program that are particularly helpful. First, it is a good idea to provide a mechanism for a user to request an explanation of the question. If the user doesn't understand the question, he should be able to type "help" or "?" to request an explanation of the question in place of an answer. Second, it should be possible to provide a default answer that is the most typical answer. A user should be able to select this answer simply by hitting the "return" key.

Many programs perform useful work by processing files and writing other files. A very useful way of identifying these files is to read the file specification at run time. One would like to be able to prompt for a file name and then open the file to read or write and then request additional file names. This is, at best, difficult in CMS.

When processing text input, it is often useful to have a collection of procedures for testing strings to find out if they have specific properties or to extract substrings. A few simple procedures provide the capabilities that are needed in many applications but more elaborate applications are best served by using an implementation of the SNOBOL scanner.

This memorandum describes a SIMULA class that contains procedures for performing these operations. A SIMULA coded implementation of a coroutine SNOBOL scanner is available for applications that require extensive text processing capabilities.

Section 2 describes procedures for asking questions at a terminal and for examining a user's response and Section 4 contains

2.

a description of file management procedures. Some useful utility procedures are described in Section 5 and Section 6 gives detailed information about gaining access to these procedures and incorporating them into SIMULA programs.

The terminal transcripts exhibited in this memorandum adhere to the following conventions. A running program prompts with an asterisk (*) and CMS at monitor level prompts with a period (.). All of the special characters (CHARDEL, LINEDEL, etc.) are non-printable ASCII characters.

2. Query Procedures

All of the procedures described here have the following properties. When the procedure is called the parameters include the question to be asked, a default answer, and a procedure to print a further explanation of the question. When they are executed, the question is printed on the terminal and the user's response is the return value. If the user responds with a carriage return, the return value is the default value and if the user responds with a question mark (?) the procedure to explain the question is invoked and then the question is asked again.

For example, suppose a program wishes to read an input file name from the terminal into a text variable `file_spec`. This can be accomplished by executing the following statement:

```
file_spec :- text_request("Input file:", NOTEXT, TRUE);
```

The first parameter is the question that is to be printed on the terminal. The second parameter is the default answer and since this parameter is NOTEXT, there is no default answer. The third parameter is the constant TRUE to indicate that no help is available. When this statement is executed, the terminal transcript will look like this:

```
Input file:
*?
No help available.
Input file:
*
? Default value may not be selected. Please try again.
Input file:
* letter simula
```

After this, the return value of `text_request` is the text " letter simula". In the first response, the user asked for an explanation of the query by responding with a "?". Since the call to `text_request` did not include a help procedure, the appropriate message was printed. Next, the response was an empty line indicating that the default value was desired. Since no default value was specified in the procedure call, a corrective response was printed and

and then the question was asked again. Finally, the third response was a character string and this string became the return value of the procedure.

As a second example, suppose that the statement:

```
f :- text_request("Output file:", "letter data", TRUE);
```

is executed. In this case the terminal transcript would look like this:

```
Output file:/letter data/:
*?
No help available.
Output file:/letter data/:
*
```

If the second response was simply a carriage return, then the return value of `text_request` would be the string "letter data". On the other hand, if the response were some character string, then this string would be the return value.

Suppose it is desirable to provide a help message in response to the input "?". This might be accomplished by writing the following procedure:

```
PROCEDURE help;
BEGIN
    Outtext("This file will contain a list of addresses");
    Outimage;
    Outtext("after the program is executed.");
    Outimage;
END of help;
```

The call

```
f :- text_request("Output file:", "letter data", help);
```

might generate the following transcript:

```
Output file:/letter data/:
*?
This file will contain a list of addresses
after the program is executed.
Output file:/letter data/:
* mylib address
```

After this transcript, the return value of `text_request` is the string " mylib address".

4.

The heading of the declaration of `text_request` is:

```
TEXT PROCEDURE text_request(prompt, default, no_help);
    NAME prompt, default, no_help;
    TEXT prompt, default;
    BOOLEAN, no_help;
```

The formal parameter `prompt` is the question to be printed on the terminal. The parameter `default` is the value to be returned if the user's response is a carriage return. The parameter `no_help` is to be TRUE if there is no help available for this query. If there is help available, it is printed by a boolean procedure that returns the value FALSE.

The procedure `boolean_request` is used to ask yes or no questions in very much the same way. The heading of its declaration is

```
BOOLEAN PROCEDURE boolean_request(prompt, default,
                                   no_help);
    NAME prompt, no_help;
    VALUE default;
    TEXT prompt;
    BOOLEAN default, no_help;
```

The parameter `prompt` is the question that is to be printed on the terminal and the parameter `default` (which must be TRUE or FALSE) is the return value if the user responds by entering "return". If the user enter responds with a "?" and if `no_help` is TRUE then the message "No help available." is printed. On the other hand, if `no_help` is a boolean procedure that returns FALSE and prints an explanation, this text is printed instead of "No help available."

For example, if the procedure `help6` is declared as:

```
PROCEDURE help6;
BEGIN
    Outtext("If tabs may be used at indentation");
    Outimage;
    Outtext("answer ""yes"", otherwise answer ""no"".");
    Outimage;
END of help6;
```

Then the execution of the statement

```
tabs := boolean_request("Tabs in indentation?:", FALSE,
                        help6);
```

might result in the following transcript:

```

Tabs in indentation?:/n/:
*?
If tabs may be used at indentation answer "yes",
otherwise answer "no".
Tabs in indentation?:/n/:
*why
? Please answer y or n.
Tabs in indentation?:/n/:
*y

```

After this transcript, the return value of `boolean_request` is TRUE. If the last response was an empty line or "n", then the return value would be false.

Note that the responses "y", "yes", "Y", "YES" are all equivalent as are "n", "no", "N", "NO". More precisely, lower case letters are translated into upper case letters before the response is examined.

The procedure `integer_request` is used to prompt for an integer response and to check if the response is an integer. Further, if the response is an integer, it is checked for being in an acceptable range. The heading of this procedure is:

```

INTEGER PROCEDURE integer_request(prompt, default,
                                min, max, no_help);
    NAME prompt, no_help;
    VALUE default, min, max;
    TEXT prompt;
    INTEGER default, max, min;
    BOOLEAN no_help;

```

The formal parameter `prompt` is the question to be printed on the terminal and the formal parameter `default` is the value that is returned if the user responds with a "return". After an integer is read from the terminal, it is checked to confirm that it is between `min` and `max`. If it fails this test, the user is asked for a correct answer. (The default value is also checked against the range if the user responds with "return"; this helps catch programming errors.) The formal parameter `no_help` is used to deal with the user response "?" as described above.

If any integer is an acceptable response, the SIMULA defined constant `Maxint` may be used in a call. For example, if the default answer to the prompt "Enter any integer:" is 0, one would execute the statement:

```

result := integer_request("Enter any integer:",0,
                          -Maxint, Maxint, TRUE);

```

As a more detailed example, consider the execution of the statement

6.

```
result := integer_request("reserved words:", 1, 0,  
                           3, TRUE);
```

The terminal transcript might look like this:

```
Reserved words:/0/:  
*?  
No help available.  
Reserved words:/0/:  
*bye  
? The input was not an integer. Please try again.  
Reserved words:/0/:  
*12  
? The input integer was out of the acceptable range [0,3].  
Please try again.  
Reserved words:/0/:  
*2
```

After this sequence of events, the return value of `integer_request` is 2.

These three procedures provide most of the terminal prompting activities that are needed. While it might be desirable to prompt for floating point numbers, this writer has not felt the need and, therefore, did not include it in this program.

An excerpt from a program that uses these procedures as well as a sample terminal transcript from the execution of the program appears in the Appendix.

3. File Management

Connecting files to a running program in CMS is, at best, a rather tedious process. Three procedures to make this task easier have been written and are described in this section.

In SIMULA, a new Infile is created by executing the statement

```
f :- NEW Infile("DDN");
```

The parameter "DDN" must be the DD name of a file. That is, before this statement is executed, a CMS FILEDEF statement such as

```
FILEDEF DDN DISK MUMBLE FOO
```

must be executed. If this is not the case, then a SIMULA run time error will occur. The procedure `find_infile` also returns a `REF(Infile)` object after creating it. It is much easier to use for two reasons: First, its parameter is a file specification, not a DD name. Second, the FILEDEF is executed inside `find_infile` and if the parameter of the procedure is not the specification of an existing file, the user is prompted for a correct file specification.

The heading of the declaration of find_infile is:

```
REF(Infile) PROCEDURE find_infile(t);
```

```
NAME t; TEXT t;
```

The formal parameter t of find_infile is a text object whose value is a file specification of the form

```
<fn> <ft> [<fm>]
```

When the procedure is called, the following sequence of events occurs:

- (1) The actual parameter is checked to make sure that it is the file specification of an existing file such that the running program has read access to the file.
- (2) A unique DD name is composed; this name is of the form DIALOGxx where xx is a two digit integer.
- (3) A CMS FILEDEF command for this DD name and file specification (on disk) is executed.
- (4) A new object of type REF(Infile) is created using the DD name and this object is the return value of find_infile.

If the actual parameter of find_infile is not the file specification of an appropriate file, an error message indicating the problem is printed on the terminal and the user is asked to provide another file specification. The procedure text request described above is used for this dialog. The help response given in response to "?" describes the syntax of file specifications and gives a reference to the CMS manual.

Using find_infile, a file named MUMBLE DATA might be opened to read by the following statements:

```
f :- find_infile("MUMBLE DATA");
f.Open(Blanks(80));
```

In the best of all possible worlds, the procedure find_infile would execute the Open before returning. Unfortunately, this writer does not know how to determine the LRECL of a CMS file from a running SIMULA program.

Here is an example of a terminal transcript that might occur if find_infile is executed with an incorrect actual parameter. Suppose the program contains the statement

```
f :- find_infile("fumble simula");
```

8.

and that this file does not exist or that the program does not have read access to the file. The terminal transcript might be:

```
find_infile: File FUMBLE SIMULA does not exist.  
Enter file specification:  
*dialog2 simula  
find_infile: File DIALOG2 SIMULA does not exist.  
Enter file specification:  
*simed simula
```

Notice that the actual parameter or the response to a terminal query may be in lower case but that an upper case file specification is used in any event.

If the actual parameter of `find_infile` or a response to a corrective response is not a well formed file specification, a corrective message is printed. It is important to note that `find_infile` has the property that when it returns a value, it is possible to execute an `Open` and to read the file provided that the record length is known without a run time error.

In `SIMULA`, an output file is created by executing the statement

```
f :- NEW Outfile("DDN");
```

where "DDN" is a DD name as for a new Infile. The procedure `find_outfile` is very much the same as `find_infile` in that the following steps are needed to create and open an Outfile:

```
f :- find_outfile("MUMBLE DATA");  
f.Open(Blanks(132));
```

When a new file is opened, the `LRECL` of this file is taken from the parameter of `Open` so it is not necessary to know the `LRECL` of a file that is to be created.

The heading of the declaration of `find_outfile` is:

```
REF(Outfile) PROCEDURE find_outfile(t);  
NAME t; TEXT t;
```

When this procedure is executed, the actual parameter should be a file specification as for `find_infile`. When this procedure is called, the following occurs:

- (1) The actual parameter is checked to confirm that it obeys the syntactical rules for file specifications.
- (2) A unique DD name is composed; this name is of the form `DIALOGxx` where `xx` is a two digit integer.

- (3) A CMS FILEDEF command for this DD name and file specification (on disk) is executed.
- (4) A new object of type REF(Outfile) is created using the DD name and this object is the return value of find_outfile.

Notice that only a syntactical check of the actual parameter to find_outfile is performed; there is a remote chance that the open may fail.

Both of these procedures use an auxiliary procedure, dd_name which may be useful in some applications. The heading of the declaration of the procedure is:

```
TEXT PROCEDURE dd_name(t); NAME t; TEXT t;
```

When this procedure is executed, the following things happen:

- (1) A unique DD name of the form DIALOGxx where xx is a two digit integer is created.
- (2) The actual parameter of dd_name is appended to the string "FILEDEF DIALOGxx DISK" and a CMS FILEDEF command is executed.
- (3) The return value of dd_name is the string "DIALOGxx".

Note that no error checking is performed and, therefore, it is possible to have run time errors if this procedure is used without appropriate checking.

4. Utility Procedures

A number of procedures that make it much easier to write interactive applications are described in this section. Some of the procedures were written specifically for the IBM SIMULA environment; some are adaptations of procedures that are in the DEC-10 SIMULA library and others are taken directly from the DEC-10 SIMULA manual.

When constructing modules from SIMULA programs, it is very much more convenient and economical to be able to directly execute the module without embedding it in an EXEC procedure. The SIMULA procedures initialize terminal and restore_terminal are designed to make this possible.

These procedures were designed under the following assumptions:

- (1) Terminal input is in upper and lower case and all of these letters are different. [The CMS

10.

default is that lower case letters are mapped into upper case letters.]

- (2) At monitor level, CMS prompts with a period (.) and a running program prompts with an asterisk (*).
- (3) At monitor level, LINEND is escape or alt mode (ASCII 27, EBCDIC 39) but in a running program the escape character may be used as a normal character. [For example, it might be used to escape from a long sequence of questions by selecting the default answer to the remaining questions.]
- (4) All messages (from other users and from the operator) are unwelcome during program execution.

When the procedure `initialize_terminal` is executed, the following happens:

- (1) The terminal prompt character is set to *.
- (2) LINEND is set off.
- (3) WNG is set off.
- (4) MSG is set off.
- (5) SYSIN is closed and then opened again with the same record length. The only difference is that the FILEDEF now includes the option LOWCASE so that upper and lower case letters are transmitted to the program without modification.

When the procedure `restore_terminal` is executed, the following happens:

- (1) The terminal prompt character is set to period.
- (2) LINEND is set to escape.
- (3) WNG is set on.
- (4) MSG is set on.

The intended application of this procedure is that `initialize_terminal` is to be the first executable statement of the program and that `restore_terminal` is the last statement executed in a program.

The remaining utility procedures are described by exhibiting the heading of their declaration and following this with a brief description of the procedure.

BOOLEAN PROCEDURE Letter(x); VALUE x; CHARACTER x;

The library procedure Letter in IBM SIMULA returns the value TRUE only if its actual parameter is an upper case letter. This version returns TRUE if its actual parameter is an upper case or lower case letter. Note that it returns FALSE for the national letters in the ISO code (the international version of ASCII).

TEXT PROCEDURE frontstrip(t); TEXT t;

This procedure returns a subtext of t that has all leading blanks removed. The value of the expression

frontstrip(x.strip)

is the text x with both leading and trailing blanks removed.

TEXT PROCEDURE upcase(t); TEXT t;

The return value of this procedure is a new text object that is the same as its actual parameter except that all lower case letters are changed to the corresponding upper case letters. It does not map lower case national letters in the ISO standard into upper case national letters.

TEXT PROCEDURE rest(t); TEXT t;

The return value of this procedure is a subtext of t that begins at t.Pos and ends at t.Length.

BOOLEAN PROCEDURE no_blanks(t); TEXT t;

The return of this procedure is TRUE if the character blank (ASCII 32, EBCDIC 64) does not occur in the text t and otherwise returns FALSE.

TEXT PROCEDURE first_token(t); TEXT t;

This procedure returns a new text object whose value is an initial string of t. This returned string is terminated by the last character of t before the first blank in t or by the 8th character of t, whichever comes first. It is useful when parsing CMS command strings and other similar applications.

BOOLEAN PROCEDURE is_integer(t); NAME t; TEST t;

This procedure returns the value TRUE if the text object t is an integer and the value FALSE otherwise. A text object is an integer if the first character is '+', '-' or a digit and if the remaining characters in t are digits.

12.

```
TEXT PROCEDURE conc2(t1, t2); VALUE t1, t2;  
  TEXT t1, t2;
```

The return value of this procedure is a new text object that is the concatenation of its two text parameters in the order first parameter, second parameter.

```
PROCEDURE nextfile(f); REF(Infile) f;
```

This procedure closes file f and then opens it again with the same record length as it had before the close. This procedure is useful when writing programs that accept an empty line as selecting the default answer. In CMS, this empty line is treated as an end-of-file and an attempt to read another record results in an error termination. By executing nextfile after the read, the error termination can be bypassed. Here is an example of code to do this:

```
IF Sysin.Image.sub(1,2) = "/*"  
  THEN nextfile(sysin);
```

```
PROCEDURE Breakoutimage(f); REF(Outfile) f;
```

One of the traditional problems with SIMULA input/output to a terminal is the following. If the sequence of statements

```
Outtext("Input file:");  
Outimage;  
Inimage;
```

is executed, input to satisfy the call to Inimage will be expected before the output line forced by the previous Outimage is sent to the terminal.

The SIMULA Standards Group is currently considering a number of proposals for extending the language to include another output procedure that will avoid this problem and make it possible to receive an answer to a question on the same line. The most probable choice for this procedure will have the above heading in its declaration.

The current version simulates this new Breakoutimage by executing two calls on Outimage. This means that a blank line will appear after the response but this is a great deal better than having the response requested before the question is printed.

5. Directions

All of the procedures described above are included in a class dialog. The design of this class was motivated by the implementation of a class SAFEIO by Mats Ohlin of the Swedish Research

Institute of National Defense in DEC-10 SIMULA. The present design was tailored to meet the needs of IBM SIMULA users and, therefore, differs in many details from SAFEIO.

To incorporate these procedures in a program, the program structure should be as follows:

```
BEGIN
  EXTERNAL CLASS DIALOG:
  DIALOG BEGIN
    <text of program using dialog>
  END of dialog block;
END of program.
```

If this program is contained in a file named TEST SIMULA, it is compiled with the CMS command

```
SIMULA TEST (CLASS DIALOG LINECNT 60 <other options>
```

Before compiling a program that uses dialog, you should either copy the appropriate files onto your disk or link to the author's disk. To copy the source and simclass files onto your disk, execute the following commands:

```
LINK ORGASS 191 333 VPI
ACCESS 333 G
COPY DIALOG SIMULA G DIALOG SIMULA A
COPY DIALOG SIMCLASS G DIALOG SIMCLASS A
DETACH 333
```

If you prefer to use the author's copy of these files, simply execute the commands:

```
LINK ORGASS 191 333 VPI
ACCESS 333 G/A
```

Using the author's copy has the advantage that you will be using the most current copy of DIALOG. If corrections are made, they will be incorporated in your program when you next compile. There are also two disadvantages: If the author changes DIALOG between the time you compile a program and when you execute the program your program may not work correctly. [It is necessary to recompile SIMULA programs that use external classes when the external class is recompiled. The CMS implementation of SIMULA does not check this at load time.] When you execute the command

```
LIST * * *
```

you will have a long list of files from the author's disk printed on your terminal.

APPENDIX

EXAMPLE OF USE OF DIALOG

This appendix contains a fragment of a program that edits the text of SIMULA programs. This program can completely reformat the text of an input file and change all of the properties of the text that influence its appearance. In addition, it is capable of inserting tab characters to reduce the disk space required to store the text.

The program interacts with the user by asking a long sequence of questions that define the behavior of the program. Each of these questions has a default answer and it is possible to select the default answer by terminating the first response with the character escape or altmode.

The following paragraphs describe the statements in the program. The reader is encouraged to examine the program text that follows the explanation as the text provides a clear example of the use of DIALOG procedures. A sample terminal transcript follows the program text.

The variable fastflag is set to TRUE if the user indicates that he wishes to select the default answers to all questions. At the beginning of the dialog, this variable is set to FALSE.

The first question asks the user to provide the name of the file that contains the program. This is done with a call to text_request. Since there is no default file name, the second parameter of text_request is NOTEXT. There is a help procedure for this query called help1 elsewhere in the program.

The specifications of this program state that if the input file name is terminated with the character escape or altmode then the default answer to all of the remaining questions are selected. After the file name is read, leading blanks are removed with a call to the procedure frontstrip. Next the last character of the user's response is examined to find out if it is escape (EBCDIC code 39).

If this character is present, fastflag is set to TRUE and an instance of Outfile for the output is created. The specifications indicate that the default file specification of the output file has the same file name as the input file and file type SIMED. In the next statements, the escape character is removed from the file specification and then the variable outf is set to the appropriate instance of Outfile using the procedure find_outfile.

The specifications for this program also state that if the file type of the input file specification is omitted, then the file type of the input file will be set to SIMULA. The complete file name is composed in the next statement using the procedure conc2 to compose the complete file specification and find infile is used to set the variable prog to the instance of Infile that will read the input file.

At this point, the input file has been initialized and if fastflag is true, the output file has also been initialized. If fastflag is FALSE, it is necessary to read the name of the output file from the terminal. The specifications of this program also state that if the name of the output file is terminated by the character escape, then the default answers to the remaining questions will be assumed. This processing as well as assigning the variable outf the appropriate value is done in the IF statement that begins IF NOT fastflag.

These are the only two questions that permit the use of the escape character to select the default answers to the remaining questions. Therefore, the next statement sets these default values if fastflag is true and then skips the remaining questions.

The next question asks the user to specify the number of characters in each indentation step. A negative answer means that leading blanks in the program text will be retained and, therefore, the allowable range of this answer is from a negative number to a positive number.

The next question asks the user to provide the rightmost position on a line where an indented line of text may begin. The smallest possible value is indent and the largest possible value is outlength.

A user is permitted to ask to have tabs used when writing the output file. The next question, using boolean_request, asks for directions. In CMS, tabs are rather difficult to work with and the most convenient answer is "no" so the default value is false.

The remaining questions ask the user to select conversion modes for different syntactical objects in a SIMULA program. A correspondence between integers and the conversion modes is printed on the terminal. After these modes are described, the user is asked to specify a conversion mode for reserved words, standard identifiers, user identifiers, comments and options and, lastly, for text constants. The appropriate program variables are set to the response to these questions.

The label fast appears at the end of these questions. The remainder of the program text consists of the code to perform the conversion.

The program text follows.

16.

```
fastflag := false;
programe :- text_request("Enter program file name:",
                        NOTEXT,
                        help1);
programe :- frontstrip(programe);
IF programe.Sub(programe.Length,1).Getchar = Char(39)
  THEN BEGIN
    fastflag := TRUE;
    programe :- programe.Sub(1,programe.Length-1);
    programe :- programe.Strip;
    outf :- find_outfile(conc2(first_token(programe),
                                "SIMED"));
  END;

prog :- IF no_blanks(programe)
  THEN find_infile(conc2(programe, "SIMULA"))
  ELSE find_infile(programe);

IF NOT fastflag
  THEN BEGIN
    outname :- text_request("Enter output file name:",
                          conc2(first_token(programe),
                                "SIMED"),
                          help2);
    outname :- frontstrip(outname);
    IF outname.Sub(programe.Length,1).Getchar
      = Char(39)
      THEN BEGIN
        fastflag := TRUE;
        outname :-
          outname.Sub(1,Length-1).Strip
        END;
    outf :- find_outfile(outname);
  END;

IF fastflag
  THEN BEGIN
    outlength := 72;
    indent := 0;
    leftskip := FALSE;
    maxindent := 52;
    tabs := FALSE;
    convert(2) := 1;
    convert(3) := 3;
    convert(4) := 2;
    convert(5) := 0;
    convert(6) := 0;
    GO TO fast
  END;

indent := integer_request("Enter indentation step:",
                          0, (-outlength//2), outlength//2,help4);

leftskip:= indent > 0;  indent:= Abs(indent);
```

```

maxindent := integer_request("Enter max. indentation position:",
                             52, indent, outlength, help5);
IF maxindent < 1 THEN maxindent:= 1;
tabs := boolean_request("Tabs in indentation?:",
                        FALSE, help6);

Outtext("Conversion modes:"); Outimage;
Outtext("No change           0"); Outimage;
Outtext("Change to upper case 1"); Outimage;
Outtext("Change to lower case 2"); Outimage;
Outtext("Change to edit case  3");
Outimage; Eject(Line+1);
Outtext("Enter conversion modes for:"); Outimage;
convert(2) := integer_request("Reserved words:",
                              1, 0, 3, TRUE);
convert(3) := integer_request("Standard identifiers:",
                              3, 0, 3, TRUE);
convert(4) := integer_request("User identifiers:",
                              2, 0, 3, TRUE);
convert(5) := integer_request("Comment and options:",
                              0, 0, 2, TRUE);
convert(6) := integer_request("Text constants:",
                              0, 0, 2, TRUE);
fast:

```

A terminal transcript of the execution of this program follows. The first line is a CMS command to load and execute the program SIMED. The message "EXECUTION BEGINS..." is emitted by CMS when the loader finishes its work.

The next line of output introduces the program. Instructions to print this message precede the text considered above.

After this, the prompt for the program file name appears and the user answered "dialog2". Since the name was not followed by escape, the next question concerned the output file name. The default name "dialog2 SIMED" is provided. The user selected this answer by entering "return".

Next, the indentation step was requested. The user found the default value unacceptable and provided 4 as his response. A value other than the default value was also selected for the question concerning the maximum indentation position.

The user accepted the default of no tabs in the output file and then the program printed the conversion modes.

The user's responses to the questions concerning conversion modes appear in the transcript.

18.

After all the questions were answered, the program text was converted and the program printed a message summarizing its work. The next line is the CMS ready message and the period on the next line indicates that CMS is expecting another command.

The help procedures were not exhibited in this example because the text is of little interest here. The code of these help procedures is very similar to the example given in Section 2.

```
.go simed
EXECUTION BEGINS...
```

SIMED - SIMULA EDITOR AND INDENTATION PROGRAM. IBM VERSION 1.0.

Enter program file name:
*dialog2

Enter output file name:/dialog2 SIMED/:

*

Enter indentation step:/0/:

*4

Enter max. indentation position:/52/:

*48

Tabs in indentation?:/n/:

*

Conversion modes:

No change 0

Change to upper case 1

Change to lower case 2

Change to edit case 3

Enter conversion modes for:

Reserved words:/1/:

*1

Standard identifiers:/3/:

*1

User identifiers:/2/:

*1

Comment and options:/0/:

*

Text constants:/0/:

*

[SIMED - Number of BEGIN's (END's) found: 43]

R;

.

DIALOG EXTENSIONS

July 10, 1979

Two procedures have been added to DIALOG. The first, `cms_subset`, makes it possible to execute a sequence of CMS commands from the terminal while a SIMULA program is in execution. The second, `get_infile`, opens files with any RECFM and LRECL to be read by SIMULA programs.

CMS Subset Commands

When the procedure `cms_subset` is executed, the following happens:

- (1) The terminal prompt character is set to sharp (#).
- (2) Each input line typed by the user is transmitted to CMS for execution. After the command is executed, CMS output is typed on the terminal and the usual ready message is printed.
- (3) When the input line return is encountered, the terminal prompt is set back to asterisk (*) and control is returned to the calling program.

Opening Input Files

The procedure `get_infile` accepts a text object that is a file specification as its parameter. If a file with this name is accessible for reading, the file is opened. The return value is an opened object of type Infile. The length of the Image attribute of this object is the LRECL of the file. An appropriate filedef for RECFM F or V and for the correct LRECL is, of course, issued.

If there is an error in the parameter file specification, an error message is printed on the terminal and the user is given two options:

- (1) Enter a correct file specification.
- (2) Enter CMS subset mode to query his directory, etc. He may return to the running program by entering return. After this, he is again asked for the correct file specification.

This procedure preserves the following property of `find_infile`: It is not possible to execute a return without successfully opening a file.

This procedure finally makes it possible to read any CMS file without knowing anything about the file before the program begins execution!

The heading of the declaration of the procedure is:

```
REF(Infile) PROCEDURE get_infile(file_name);  
    TEXT file_name;
```