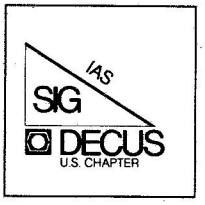
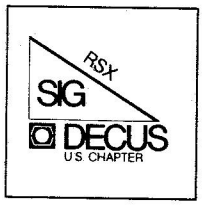
The DeVIAS Letter





The RSX Multi-Tasker

September 1983

Printed in the U.S.A.

The DeVIAS Letter

Volume No. 16

In This Issue

From the Editor

I hope some of those receiving this newsletter are using IAS or at least know someone who is and can pass the letter along.

For those of you who don't even know what an IAS is, my condolences. You may, however, find the enclosed matter of interest since much that affects IAS also affects RSX.

For those of you who didn't get the previous issue, I need submissions. I will take anything that can be printed in a free-press society and has some relevance to IAS, DEC or DECUS.

I hope to publish at the rate of one issue every month or two, so I need material.

Contributions

The DeVIAS Letter needs contributions in order to continue as an effective medium for exchange of information regarding IAS.

Contributions may be submitted in any form you wish. Originals on 8½ x 11 paper are preferred. However, even photocopies of relevant match-book covers would be appreciated.

Send all contributions to:

Ontario Hydro
700 University Avenue
Toronto, Ontario
CANADA, M5G 1X6

Attn: John W. Drummond
Mail Stop - M2E10

Department of Radiation Therapy
University of Pennsylvania
Room 410 - 133 South 36th Street
Philadelphia, Pennsylvania 19104
31 July 1983

Dear IAS SIG Member,

There are several issues that bear upon us directly. The most important, in my view, is the newsletter sharing with the RSX SIG. The DECUS management has decided that any new SIG will share the newsletter with an established SIG for the first year. This makes sense in a couple of ways, (a)it gives the new SIG time to get its newsletter act established before being forced into a paid subscription program on its own and (b)it allows DECUS to plan ahead in its Subscription Service budget. Thus, we will share the Newsletter with the Multi-tasker for this fiscal year. I have asked that the pages of each newsletter ("The Multi-Tasker" and "The DeVIAS Letter") be different, and that we alternate placement on top. The RSX SIG has already indicated a willingness to share and be helpful.

Scheduling for the Fall Symposium in Las Vegas is done. John Jenkinson did the preliminary work but was unable to attend the week scheduling meeting in Marlboro, Massachusetts. Both Jim Hopp (the Symposia Coordinator for the RSX SIG) and Steve Finch (of Emulex Corporation, Costa Mesa, California) did the on site work necessary for us to have IAS sessions in Las Vegas. Thank you both for taking up the gauntlet on short notice. This brings up an important SIG issue: we need a volunteer to be Symposia Coordinator. I would be glad to hear from you now or find me in Las Vegas and we can discuss the job (some DECUS paid travel is required).

The sessions on the schedule for Las Vegas are:

Nibbles and Bits of IAS, Ken Guralnik, Thursday, 27 October, 8-11pm

IAS Language Panel, Mike Reilly, Tuesday, 25, October 10-10:30am

IAS SIG Opening Session and Roadmap, Bob Curley, Monday, 24 October 8:30-9:30am

IAS SIG Closing Session and Wish List, Ken Guralkik, Friday, 28 October, 10-11am

IAS/VMS User panel, John Jenkinson, Tuesday, 25 October 12:30-1:30pm

IAS SIG Planning, SIG Steering Committee, Thursday, 27 October 7-8pm

IAS QA, Bob Curley, Monday, 24 October 8-11pm

IAS Product Panel, Tim Leisman, Monday, 24 October 2-3pm

The IAS QIO Unveiled, Mike Reilly, Thursday, 27 October 10-11am

Aerodynamic Laboratory Data System in IAS, Stephen L. Tomlin, Thursday, 27 October, 12-1pm

Enhancing System Security of IAS, Larry Barrett, Tuesday, 25 October 10:30-11am

IAS.RSTS/E.RSX.VMS, Curley et al, Tuesday, 25 October 5:30-6:30pm

You should have Autopatch E if you have any kind of software support. Even if you have not ordered Autopatch before. Ask your software support office if you have not already received it.

I would like to hear from you about talking to your IAS system with DEC personal computers. Really, any personal computers, but especially with DEC PC's. I have a DECmate II and will get a Rainbow "sometime soon". And, I KNOW that intercommunication is NEVER as simple as it should be. Anyone who has solved the problems -- please let me know. Thanks.

I noticed that there is a new contribution to the DECUS library, from New Zealand, that is IAS specific. Why is it that we contribute so little to the Library? I have heard that question often, in the halls of DECUS. I would like to hear your answers or opinions. Please.

The RSX SIG has an active group working on SRD. It would be nice, since most of us use it, if our SIG helped. Please call or send me your name, to be connected with this worthwhile effort.

I have asked Ken Guralnik (EG&G, Las Vegas) to start up a mechanism that will permit us to focus our desires for IAS on DEC. It is often called a "Wish List" or "SIG Menu", but Ken has come up with a better name. He'll tell you about it shortly, but if you can't wait, call him at 702-647-5551. There are, of course, wishes that never come true. But, if DEC never knows, your wishes will never come true. I have found the Development group very helpful and reasonable in my requests.

The [14,*] UFD tapes that Paul Clayton spoke of in St.Louis are ready. A very cooperative, but anonymous, person has created three BRU tapes (two at 1600bpi and one at 800bpi) of the whole collection. Send me a request, I'll send you a tape, you copy it and return the original to me. If you missed St. Louis, Paul Calyton described a herculean effort whereby he collected over 100 individual programs or collections from DECUS sources (and his own contributions). All run on IAS. His sources include the "Best of ICR", of Stodola, Wood and Cael; "Reece's Pieces" of Frank Borger and many Symposia RSX/IAS Tapes. Some required extensive modification of command files, and the like, to work. Paul has offered to share them with us.

Sandy Krueger, a long time DeVIAS member, has been elected Chairman of the SIG Council. As many of you know, Sandy has been Chairman of the DMS SIG for a couple years, succeeding Sat Mohan (also a DeVIAS Member). The SIG Chairmen were invited to a meeting last weekend at the O'Hare Hilton. We voted to form a group, that consists mainly of SIG Chairmen, to be called the SIG Council and elected Sandy to chair it. The SIG Council is an effort to make DECUS more representative, since you talk to me and I'm supposed to talk to DECUS management. To make it all work you must talk to me or the SIG

chairman you feel will listen best. There are many important (and devisive issues) that must be addressed and solved by this level of the membership so that DECUS can continue to be the helpful organization that it has been for so long. Contribute your ideas and criticisms to your SIG Chairmen.

I was asked a few weeks ago for items to form a "handout" at the Las Vegas meeting. I responded that we didn't have that act together yet. I'd like to hear from you who think that handouts are a good idea for this SIG. I think that our newsletter is a good forum for those things (few as they are) and the Proceedings the right place for formal papers. Many of the SIG's use the handouts as a mechanism to earn money for SIG activities. Any opinions?

My work address has changed. My new office is at 36th and Walnut Streets in Philadelphia and the address above does not usually enter the University nor Hospital Mail System. My Old address, 3400 Spruce Street, ususlly entered the Hospital Mail and sometimes got to me. Sometimes it would enter the University Mail and never get to me. (I was approached in San Diego a couple of years ago, at a DECUS Symposium, with the words: "Oh, You're Bob Curley --- I get your mail" by someone else whose badge said "University of Pennsylvania"). Thus, you may use the new address above, or the Post Office Box that I rent to avoid the issue altogether: P.O. Box 322, Flourtown, Pennsylvania 19031-0322. Or you may phone, 215-662-3083 (8am-4:30pm Eastern Time).

Another letter that I received recently asked if we wanted to have items for sale in the DECUS Store at the Symposia. You've all seen the VAX Ties and various Sig Tee Shirts. Do we wish to sell (or more practically, buy) IAS items? A volunteer to organize and maintain that efort would be appreciated. I answered this letter that we would have no items this Symposium. Again, this is a mechanism for SIG's to make a few bucks for SIG activities.

I must thank, again, the many people who make this newsletter possible. The obvious one is John Drummond. But there are you who send in items and the DECUS Staff who get it printed and mailed. There are you who send in your subscription dollars to support all the efforts and ultimately make it possible. Thanks to all of you who contribute.

Robert F. Curley
Chairman
IAS SIG

Bell Technical Operations TEXTRON

Bell Technical Operations Corporation
Subsidiary of Textron Inc.

1050 East Valencia Road
Tucson, Arizona 85706
602/294-2651
TWX 910-952-1103

16 February 1983

To: Robert Curley
Delaware Valley IAS Local
P.O. Box 322
Flourtown, PA 19031

RE: Disk and Tape Management System for PDP 11/70 IAS

Dear Mr. Curley:

We are looking for an automated tool which would assure that the most recent version of a file is accessed and provide information for configuration control purposes.

Specifically, the tool should (1) automatically record file name, version, date, and device ID, (2) prompt the operator for the correct disk or tape, and (3) then only accept the specified disk or tape. The PDP 11/70 IAS operating system now selects the most recent version from the mounted device, but what we want is the most recent version anywhere.

I would appreciate hearing from other users who have solved or are trying to solve this problem through disk and tape management systems or other methods.

Very truly yours,

BELL TECHNICAL OPERATION CORPORATION

Mary K Roberson

Mary Roberson
Systems Analyst

28 June 1983

Mr. Robert F. Curley, Chairman
IAS SIG
Hospital of the University of Pennsylvania
3400 Spruce Street
Philadelphia, Pennsylvania 19104

Dear Mr. Curley:

As you know, DECUS is a valuable vehicle for information exchange between Digital and our customers. Our objective is to insure that Digital customers realize the maximum benefit to better meet the needs of the marketplace. Our decision to extend support of the IAS product set was based on the clear feedback we received from your organization regarding the need for continued support.

I am pleased to honor your request to have Tim Leisman serve as the "Digital Counterpart" to the newly formed IAS SIG. Tim brings to this position his experience as a "Digital Counterpart" to the RSX-IAS SIG, his enthusiasm for IAS and a sincere interest in the needs of the IAS user community.

We believe that IAS has a bright future. This is based on the quality of the product and the quality of the communication between Digital and the IAS user community. With your continued support and the support of the IAS community we look positively to the future.

Sincerely,

William P. Ferry
Corporate Manager
Software Product Services

WPF:j

John Guidi
The Computing Center
The Jackson Laboratory
Bar Harbor, Maine
04609
phone: (207)288-3371 X-391

Bob;

Question: Does anyone have the DECUS C system running on an IAS V3.0 system? If so, would you please contact me at the above address?

We are running IAS V3.0 and recently we have had some hardware problems which prompted a visit by DEC Regional Support. To our delight, the person sent up was intimately familiar with IAS. Some of his suggestions and comments may be of interest to other IAS users, so I pass them to you to do with as you see fit.

(Ed. see attached)

- Bruce Wright of Duke University published a patch for the IAS V3.0 terminal handler in the April 1981 issue of the Multi-Tasker (Vol 14, Number 4, pp.106-107). The patch causes parity errors in the type-ahead buffer to be handled the same as breaks or framing errors. Without this patch, parity errors in the type-ahead buffer can fill up the terminal buffer, locking out all terminals.

- The BAD utility distributed with IAS V3/V3.1 does not inhibit retries of marginal sectors. The error log may fill up when you run BAD, and if so, you should include any sectors where retries occured in a subsequent run of BAD.
  The BRUSYS.SYS image on the IAS V3.1 distribution contains the RSX-11M version of BAD which has been referred to as a "snowplow" - it does not stop to perform retries. We created a bootable tape by building the Virtual MCR (VMR) utility and running the BRUSYS.SYS image through it to tape. When booted, the tape loads a system containing the CNF configuration program to describe devices, the FMT disk formatter, the "snowplow" BAD, and a copy of BRU. To do volume maintenance, we bring the system down, boot this tape, run the CNF configuration program, and then run the appropriate utilities.
  NOTE: Once you have BRUed onto a volume, you will be unable to use DSC to make a copy of that volume. BRU writes information into the home block which DSC does not know how to handle. We have been told that this is no great sacrifice as BRU is said to perform better than DSC. Also with BRU, you can specify the location of the index file.

Sincerely,

John Guidi

ignored. Various alternate coding sequences will cause the problem to disappear from the test program, but instances where "real" programs encounter this bug are much harder to find.

```
      COMMON ID(1),IT
      DIMENSION IO(2)
      EQUIVALENCE (IO(1),ID(1))
1     FORMAT (I3)
      IT = 1
      WRITE (5,1) IO(2)
      IT = 2
      WRITE (5,1) IO(2)
      END
```

```
>FOR TEST=TEST
.MAIN.
>TKB TEST=TEST
>RUN TEST
 1
 2
TT11  --  STOP

>FOR TEST=TEST/CD:EIS
.MAIN.
>TKB TEST=TEST
>RUN TEST
 0
 2
TT11  -- STOP
```


HELP YOURSELF

"Help Yourself" is a place for you to get your tough questions answered. Each month, questions from readers will be published. If you have a question, send a letter to the Multi-Tasker at one of the addresses listed on the cover.

We would also like to publish the answers to questions. If you can help someone, send a letter to the Multi-Tasker or call Ralph Stamerjohn at (314) 694-4252. Your answer will be sent directly to the person in need and published in the next edition of the Multi-Tasker.

ANSWERS TO PREVIOUS QUESTIONS

IAS TERMINAL LOCKOUT

Bruce Wright from the Duke University Medical Center had an answer for the IAS terminal lockout question in the January, 1981 edition.

The IAS terminal lockout problem is a very well-known problem. The basic cause is that the IAS terminal handler will buffer up parity errors without regard for the typeahead buffer size. This is not done for framing errors (a bit is set indicating that a framing error has occurred, but it doesn't buffer up framing error messages). If a line is generating enough parity errors without any task doing a read on the line to get the parity error report, the ENTIRE terminal buffer area can be filled up with parity error reports for ONE terminal! The result of this is (as was observed) a <bel> response from any terminal whenever anything is typed on a terminal. It has nothing to do with system load static, etc., although parity errors on terminal lines can be generated by line cross-talk.

We have been trying (> 1 year) to get DEC to respond to this problem, but the only response we ever get is that this is considered a feature (!!) because it is possible for a task to get a report on EVERY parity error which occurs on a line. This may be desirable for communications applications, but for other uses of terminals it is UNACCEPTABLE! This is especially true as the fix is quite easy. The following code was developed for IAS V3.0, and works for the field test release of IAS V3.1 as well since the appropriate module has not been changed. The effect of the patch is that only the first parity error in typeahead is reported (if the terminal is not in typeahead all parity errors will be reported). This is the way breaks or framing errors are currently handled. In the module ISRRTN, the subroutine UARTER (Uart error), the following patch:

```
-40
;
; PATCH AUTHOR          DATE            REASON
; ----- ------          ----            ------
;
; L01   BRUCE C. WRIGHT 10-SEP-80       TREAT PARITY ERRORS LIKE
;                                       FRAMING ERRORS (COPIED
;                                       FROM IAS V3.0 LOCAL MODS)
-416,422
        BMI     20$             ;++L01 SKIP ON DATA OVERRUN.
10$:    ASL     R5              ;++071 TRY FOR BREAK (FRAMING ERROR)
        BPL     12$             ;++L01 J IF NOT
        MOV     #TE.BCC,R5      ;++071 ELSE GET CODE
        BR      14$             ;++L01 AND JOIN COMMON CODE
12$:    MOV     #TE.VER,R5      ;++L01 GET PARITY ERROR CODE
14$:    BIT     #SC.BRK,(R4)    ;++L01 ALREADY SEEN A BREAK?
-428,428
20$:    MOV     #TE.DAO,R5      ;++L01 GET DATA OVERRUN CODE
/
```

THIS MONTH'S QUESTIONS

6502 CROSS ASSEMBLER

I have been attempting to locate a 6502 cross assembler for the PDP-11 for either RT-11 or RSX-11M for quite some time now. Rumor has it that there is "more than one version out there" which is probably a safe assumption. If

NASA/Goddard Space Flight Center
Code 933
Greenbelt, MD 20771
301-344-5003

July 18, 1983

Bob Curley
Dept. of Radiation Therapy
University of Pennsylvania Hospital
3400 Spruce Street
Philadelphia, PA 19104

Dear Bob:

The enclosed materials were sent to the SRD Working Group members and others who have contacted me concerning SRD. I thought the RSX SIG Steering Committee would be interested in the status of SRD. The questionnaire concerns the emphasis that the SRD Working Group should place on two program versions under consideration. It is being sent to you for your information, but feel free to return it if you want.

If you would like a copy of the versions of SRD under consideration, please send me a tape. I will assume 1600 bpi BRU format unless you state otherwise.

Sincerely,

Bob Turkelson

Bob Turkelson

SRD Working Group Questionnaire

SRD Version 6.3 has been submitted to the Spring 1983 RSX SIG tape ([352,4]). This version was produced by merging Glen Everhart's recent modifications which appeared on the Spring and Fall 1982 tape (/HD, /SM, and /BK switches) into V6.2.

At the recent Spring 1983 Symposium in St. Louis, the working group discussed desired enhancements to this version of SRD. Enclosed is a wish-list generated by suggestions made by those interested in SRD.

Henry Tumblin, to whom I had submitted changes which went into V5.0 on the Fall 1979 tape when he was in the Files-11 Working Group, has sent me his new version of SRD. He has done much work in cleaning up the code and source code documentation, as well as adding some useful features. His version has a sort on date switch, a multi-column listing format, a delta date specification feature (such as for files created the last 5 days, or the last 2 weeks, for example), and it uses $EDMSG to generate messages. Unfortunately, he started with SRD V5.0. The process of bringing his version up to date with the V6.3 features would be much easier and faster had he modified V6.0 from Phil Stephensen-Payne, who merged V5.0 with Ray Van Tassle's version, for the Spring 1981 tape. Henry's recent suggestion was to modify his version, with his help, to incorporate features added since V5.0 which he has not yet included. Our goal would be to have this version distributed on the Spring 1984 tape, since only a few months remain before the Fall 1983 Symposium. Of course, we can still try for the Fall tape.

As an example of how Henry's version is documented, I have attached listings of modules SRDINI and SRDLST from both versions. (ED. NOT INCLUDED)

I was only recently able to try this version (after Henry explained that he had a solution for the problem for the RSX-11M version he had sent to me). With limited testing so far, his version works well.

An example of the multi-column format he uses to display file names is attached. We would need to allow the generation of the current V6.3 format by a switch (which could be defaulted) so programs and command files would continue to work.

Some people have suggested ignoring Henry's version for now, since he started with an early version and to bring it up to date with adequate testing would be quite time consuming. On the other hand, since he has made many valuable contributions it might be worth the effort to merge them.

Please let me know your feelings on how the working group should proceed by returning the enclosed questionnaire.

SRD Working Group Questionnaire

Please give your opinions on how the SRD Working Group should proceed:

1. Should the working group add features to V6.3 for the Fall 1983 SIG tape?

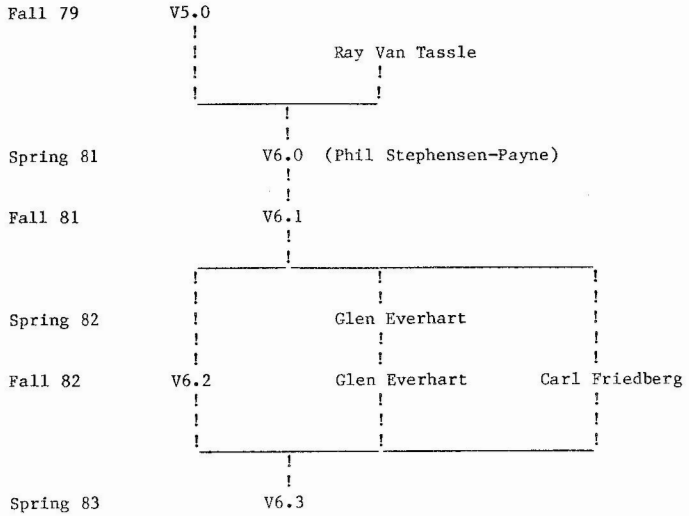
2. Should the working group plan to adopt Henry Tumblin's version for the Spring 1984 tape?

3. If so, should the working group be modifying only Henry Tumblin's version now, even though we may not be ready with a compatible version by the Fall 1983 SIG tape? (The code required for many of the desired modifications would be the same for either version, so our work upgrading V6.3 would not be entirely "lost.")

Example Directory Listing From Henry Tumblin's SRD

```
[ Directory of VDO:[352,004] 4-JUL-83  17:12 ]
SRD.CMD;1           SRD.COR;1           SRDATA.COR;1        SRDDBF.COR;1
SRDINI.COR;1        SRDLST.COR;1        SRDNUD.COR;1        SRDOPR.COR;1
SRDPRE.COR;1        SRDREP.COR;1        SRDROT.COR;1        SRDSRT.COR;1
SRDSUB.COR;1        SRDTRP.COR;1        SRDTST.COR;1        SRD.DOC;1
SRDMOD.DOC;1        SRD.HLP;1           SRDATA.MAC;1        SRDDBF.MAC;1
SRDINI.MAC;1        SRDLST.MAC;1        SRDNUD.MAC;1        SRDOPR.MAC;1
SRDPRE.MAC;1        SRDREP.MAC;1        SRDROT.MAC;1        SRDSRT.MAC;1
SRDSUB.MAC;1        SRDTRP.MAC;1        SRDTST.MAC;1        SRDXX1.MAC;1
SRDXX2.MAC;1        SRDXX3.MAC;1        SRD.RNO;1           README.1ST;1
[ 36 files listed out of 36 files in VDO:[352,004] ]
```

SRD



July 15, 1983

Suggestions for SRD Modifications

- For multi-header files, show the correct file size.
- Add a line showing the switch defaults when /ID specified.
- Include the capability to handle version numbers ;0 and ;-1 as PIP does.
- Merge in the /FO switch from Ray Van Tassle's version of SRD, so that file selection may be based upon file owner. Also it would be nice if /-FO:[g,m] displayed all files not owned by [g,m].
- When displaying the SRD command line, display the original command before substitution of special characters.
- Merge in the file sorting features from the version in the U. S. Forest Service collection of programs on the SIG tapes (Spring and Fall 1982 [307,120]). This version allows sorting by file name, file type, and version in ascending or descending order. It also has a nice command file generator feature.
- Merge in the single character wildcard in UIC capability, as found in DIR (Spring 1982 [350,300]) (and perhaps other SRD versions).
- Fix some problems which have been around for a while. For example, when an error occurs reading a file header (such as a file-id, sequence number check), SRD displays an error message, but then proceeds through whatever testing would have been done to determine if the file name should be listed, which means that the name of the file in error may not be displayed, or that meaningless tests are made. When using one of the date switches, if a header contains an invalid date format, SRD displays a fatal error message telling the user to enter the date in a correct format, without giving the offending file name, and then quits working on that directory. The same routine (CVDATE in module SRDSUB) which verifies a date specified in the command line switches is used to check the date within the file header. If the error is from a file header date, SRD shoud give an appropriate error message and continue going through the directory.
- The addition of an /ER switch which displays the file names which cause an error while reading the file header.
- A change should be made so that the following won't happen: Specifying the /AF switch with a date will find no files if the system date and time have not been set. SRD looks for all files between the specified date and the "current" system date. This is OK under most circumstances, but it surprised one user who was checking a disk he had just copied (to make sure the copy was performed in the correct direction).

- Add a switch to allow selection based upon allocated file size (all files equal to or greater than a specified size).

- Add a /TB switch to always print the total number of blocks in the selected files, even when it would not otherwise be calculated (for example, when neither the middle nor full listing is desired).

- Modify the command file to generate SRD:

  - Reduce the number of questions necessary for choosing the default switches for most systems by grouping one or two sets of options most people choose, and asking the user if that set of options is desired. For example, many people choose /LI/SR/-WD and /-NA/-RD/WI/AT/M2.

  - Perhaps show a default set of default switch settings, giving an experienced user the option of entering any changes to this by typing the switch settings desired. The full question and answer method of choosing default switch settings would still be available.

  - Supply the source files in a universal library, from which SRD.CMD extracts them, does the assembly, and inserts the object modules into an object library which the task builder references.

  - Create SRDDEF.HLP showing the default switches, which SRD.HLP could reference.

- Generate a DCL interface - either external or internal to SRD. Paul Sorenson's DIR program (Spring 1982 [350,300]) has internal DCL type qualifiers. Henry Tumblin has a parsing module to handle DCL type qualifiers, but this module was not completed.

- Several other desirable features are in Henry Tumblin's version now (such as sorting by date, appending to an output file, limited multiple file specifications).

- Investigate handling named directories and subdirectories for future versions of RSX-11M-PLUS.

- The working group should decide what header and trailer formats should be used.

- The working group should decide whether to keep the /BE, /BF, /AE, and /AF switches for inclusive and exclusive date specifications, or find a better method.

- ... and documentation, help files, and Multi-tasker articles ...

IAS V3.2

PRODUCT DESCRIPTION

IAS is a general purpose operating system that runs on PDP-11/23 Plus through 11/70 central processors. It is a multiuser timesharing system that supports concurrent interactive, batch, and real-time applications.

The goals of the IAS V3.2 product are to:

- Enhance the useabiiity, reliability, and supportability of the IAS system.

- Keep IAS and its supported dependent products (BASIC-PLUS-2, PDP-11 COBOL, FORTRAN IV, PDP-11 FORTRAN 77, FMS-11, DATATRIEVE-11 AND DECnet) as current as possible with their RSX-11M counterparts.

- Support the Installed Base Marketing Group (IBG) add-on market sales strategies through adding support for new hardware.

- Increase hardware and file compatibility with VAX/VMS to facilitate migration to VAX/VMS.

To accomplish these goals, a maintenance release will be developed. IAS V3.2 will be an updated release of the currently offered IAS system. The principle features which comprise V3.2 are:

- Fix all known bugs.

- Provide a private node pool area for the IAS executive thereby increasing the number of nodes available in the System Common (SCOM) node pool for user tasks.

- Incorporate the additional functionality in the Files-11 ACP needed for RMS V2.0.

- Fully support all current PDP-11 CPU's including the 11/23, 11/23 Plus, and 11/24. The "extended addressing" RLV22 and RXV22 controllers will be the only supported disk devices on the 11/23 Plus.

- Provide support for additional hardware including the UDA50/RA60,80,81 fixed disks; the TS11 unibus tape drive; the RM80 and RP07 massbus disk drives; and the TU77 massbus tape drive.

- Include the common utilities from RSX-11M V4.1 needed for

  BASIC-PLUS-2/IAS
  DECnet-IAS
  FORTRAN IV/IAS
  PDP-11 FORTRAN 77/IAS

  (Test sites will be licensed for these layered products and will be provided them as they become available during the test.)

- Enhance the backup and restore utility (BRU) to allow backups of larger disks onto multiple smaller disks.

IAS will be provided on identical format distribution kits as V3.1 with the addition of a 1600 bpi tape kit for the TS11 tape drive.

PREREQUISITES

Hardware - The minimum hardware configurations currently listed in the IAS V3.1 SPD will be supported with the addition of:

- the 11/23, 11/23 Plus, and 11/24 CPU's
- the RM80 and RP07 disks
- the TS11 and TU77 tape drives

Software - None. IAS V3.2 will contain all of the software that is currently furnished with V3.1 with the exception of the unoverlayed macro assembler (PURMAC). This includes the executive services, file system, MCR, privileged and non-privileged utilities and I/O drivers. The terminal handler will be updated to support the TC.TBF characteristic, the IO.RST and IO.RTT functions, and RSX-11M compatible character AST's.

SITE CONFIGURATIONS DESIRED

The following should be included among the chosen test sites:

CPU's: 11/23 Plus
11/24
11/44

Peripheral Devices: RM80 disk drive
RP07 disk drive
multiple DR or DB type controllers on a single CPU
TS11 tape drive

For phase 2:

UDA50 controller with one or more RA60/80/81 drives

CONTENTS OF RSX F82 SIG TAPE - LISTED BY UIC

[300,001] - DOCUMENTATION ON CONTENTS OF THIS TAPE, INCLUDING THIS FILE.
[300,002] - BIGTPC.TSK [AND BIGTPC.OBJ] - PROGRAM TO MAKE DUPLICATES OF THIS AND OTHER ARBITRARY FORMAT TAPES.
[300,120] - JOHN OSUDAR LATEST VERSION OF VS: VARIABLE SEND DATA DRIVER, WITH SPEED ENHANCEMENTS, AND ASSOCIATED VSUTIL UTILITY PROGRAM.
[300,134] - WAYNE BAISLEY PAPER ON "AST'S AND SST'S IN AN OVERLAY ENVIRONMENT".
AUTOLOAD OVERLAY ROUTINES FOR SYSLIB.
AUTOLOAD OVERLAY TRACE (AUT) AND FORMATTER (ATF) PROGRAMS.
[300,135] - MANARD STEWART OLYMPIC SIZED POOL FOR RSX-11M V4.0 - FILES AND DOCUMENTATION FOR MOVING SOME OF THE CRASH CODE FROM THE EXEC TO EXCOM2, GAINING ABOUT 1/2 K OF POOL.
[300,136] - STEPHEN DOVER PAC GAME WITH REVISIONS SINCE S82 TO REDUCE IMPACT ON SYSTEM PERFORMANCE WHEN SEVERAL COPIES ARE RUNNING AT THE SAME TIME. ALSO SAVES HIGH SCORE.
[300,137] - STEPHEN DOVER CEN GAME -- MODIFIED FROM S82 TAPE TO REDUCE IMPACT ON SYSTEM WHEN TWO OR MORE COPIES ARE RUNNING AT THE SAME TIME.
[300,140] - RAY VAN TASSLE MEMORY-RESIDENT DISK FOR 22-BIT SYSTEMS.
C PROGRAMS: RANDOM NUMBER GENERATOR, QUICKSORT, FILE SORTER.
[300,201-214] - DAVID BURCH FERMILAB COMMUNICATIONS SOFTWARE PACKAGE FOR INTERPROCESSOR COMMUNICATIONS VIA A DR11-W LINK.
[301,067] - DAVID BARSKY ENTRY - FULL-SCREEN DATA ENTRY SYSTEM, MODELED AFTER KED/EDT.
[303,040] - MICHAEL OOTHOUDT SOURCES FOR FLECS AND ALECS STRUCTURED LANGUAGE PREPROCESSORS FOR FORTRAN AND MACRO.
[307,020] - GARY MAXWELL UPDATED USGS PACKAGE; INCLUDES LATEST OF:
CSH - CHECKPOINT SPACE HANDLER - DISPLAY & EMPTY CHECKPOINT FILE,
CWD - CHANGE WORKING DIRECTORY,
DVCDAT - DEVICE DATABASE DISPLAY,
SNAP - GET PMD TO TAKE A SNAPSHOT OF A TASK WHILE IT'S STILL RUNNING,
WHO - WHO IS ON SYSTEM AND WHAT ARE THEY RUNNING.
/BAC:SVENTOOLS [307,30-37] - JOE SVENTEK - ON 2ND BRU BACKUP SET=SVENTOOLS, A NEW RELEASE OF THE COMPLETE SOFTWARE TOOLS KIT.
[307,050] - CHARLES SPALDING NEW RELEASE OF S82 RUNOFF BY THIS AUTHOR.
[307,100] - PHILIP KURJAN ACCOUNT -- ACCOUNTING PROGRAM FOR RSX-11M V3.2; NOT AS COMPREHENSIVE AS KMS ACCOUNTING, BUT NO SYSGEN REQUIRED.
GRAF -- PLOTTING PROGRAM FOR HP 7220 PLOTTER.
[307,120] - DONALD MCCOY MISCELLANEOUS UTILITIES FOR RSX-11M V4.0 - INCLUDES INTERUSER MAIL SYSTEM, CVT (RADIX CONVERTER), SNOOPY CALENDAR, COOKIE, CCL, VIRTUAL DISKS, WHO, OPA, REW (REWIND MM:), RNO, SRD, OTHERS.
[307,131] - STEPHEN REINIER BLP - "BLOOPER" TERMINAL LOCKUP PROGRAM.
[307,132] - STEPHEN REINIER DVC - UPDATING DEVICE STATUS UTILITY - (LIKE DEV BUT UPDATING CONTINUOUSLY).
[307,133] - STEPHEN REINIER RRU - REMOTE TERMINAL RUN PROGRAM.
[307,134] - STEPHEN REINIER TRM - TERMINAL STATUS REPORT PROGRAM.
[307,135] - STEPHEN REINIER PTT - TERMINAL "SPECIAL EFFECTS" PROGRAM FOR VT-100'S.
[307,136] - STEPHEN REINIER RSX CILUS - PROGRAM FOR MANIPULATING DOS CIL FILES.
[307,211] - DENNIS PULSIPHER
MC2 -- CATCH-ALL TASK DESIGNED TO DO "FLYING INSTALLS",
CPA -- CRASHED POOL ANALYZER,
OPA -- ON-LINE POOL ANALYZER.
[312,022] - RICHARD KIRKMAN MISCELLANEOUS UTILITY ROUTINES FOR BASIC, CORAL, ... ?

[312,315] - GLENN EVERHART
BIGTPC -- LATEST VERSION, WORKS ON VMS AS WELL AS RSX,
DDT -- WORKS WITH I/D SPACE ON M+ V2,
FLOATING POINT EMULATOR FOR M AND M+,
FILE RECOVERY PROGRAM,
TECO MACRO TO EMULATE EDT V2,
TRUNC FILE TRUNCATE AND FFL FAST FLX UTILITIES.

[312,322] - GLENN EVERHART VEDRV - VIRTUAL DISK DRIVER WITH MULTIPLE FILES PER VIRTUAL DISK UNIT. THIS VERSION HAS ADDITIONAL SECURITY ENHANCEMENTS.

[312,332] - GLENN EVERHART SRD REVISED FROM S82 TAPE, WITH /BK SWITCH TO INSERT EXTRA BLANKS BEFORE FILENAME (FOR COMPATIBILITY WITH OLDER SRD FORMATS USED BY POSTPROCESSORS).

[312,345] - GLENN EVERHART PORTACALC - PORTABLE (FORTRAN-BASED) SPREADSHEET CALCULATOR PROGRAM.

[312,347] - GLENN EVERHART MODIFICATIONS TO 2 MEMORY-RESIDENT DISK DRIVERS, ORIGINALS ON THIS TAPE ALSO.

[326,*] - JOHN JENKINSON
SPELL -- SPELLING CHECKER PROGRAM,
FLOPPY -- FLOPPY DISK STRUCTURE ANALYSIS PROGRAM,
MSTRMD -- MASTERMIND GAME,
DOCEXT -- DOCUMENTATION EXTRACTOR PROGRAM.

[330,002] - FRANK KEEFER FDT - FORTRAN SYMBOLIC DEBUGGING TOOL - UPDATED FOR 11M V 4.0, F77 W/ STRING MANIPULATION FEATURES.

[332,012] - JOHN CLEMENTS RUNOFF -- ENHANCED, WITH MANY DSR FEATURES AND SOME EXTRAS; WILL RUN ON RSX, IAS, OR VMS INCOMPATIBILITY MODE.

[332,115] - JACK LEES CAT - CATCHALL TASK FOR MCR. IMPLEMENTS CHECK FOR SPECIAL COMMANDS, PASSES LINE TO DCL IF NOT RECOGNIZED.

[333,100] - ALLEN WATSON
SLP FILE TO MAKE CCL EITHER A CLI OR CATCHALL UNDER M-PLUS,
HELP FILES FOR TECO AND RUNOFF,
PAPERS ON M TO M-PLUS CONVERSION AND SRD FROM ANAHEIM SYMPOSIUM,
COMMAND FILE TO SET UP SYSMVR.CMD FOR M-PLUS V2,
EDT SETUP FILE FOR SETTING UP DEFINED KEYS.

[333,101] - ALLEN WATSON SRD DESIGNED FOR BUILDING AS MULTIUSER M-PLUS TASK, NON-OVERLAID.

[333,102] - ALLEN WATSON SLP FILE FOR UNSUPPORTED M-PLUS V1 LOG UTILITY (CREATES COPY OF TERMINAL OUTPUT IN A FILE).

[333,103] - ALLEN WATSON UTILITIES IN C, TECO MACROS.

[333,104] - ALLEN WATSON TMV - TAPE MOVE UTILITY, FOR MOVING FOREIGN-MOUNTED TAPE FORWARD, BACKWARD, REWINDING, WRITING MULTIPLE TAPE MARKS, ETC.

[334,002] - SCOTT SNADOW UTILITIES TO MODIFY TASK IMAGE FILES:
MODLUN -- CHANGE LUN ASSIGNMENTS,
MODTSK -- MODIFY OTHER TASK OPTION INFORMATION,

[343,031-32] - BRUCE MITCHELL MEMORY-RESIDENT PSEUDO-DISK FOR 22-BIT SYSTEMS.

[343,033] - BRUCE MITCHELL IDLE TERMINAL MONITOR PROGRAM TO LOG OFF IDLE TERMINALS.

[344,*] - JIM DOWNWARD KMSKIT ENHANCEMENTS PKG FOR RSX-11M V4.0 - INCLUDES CCL, KMS ACCOUNTING PACKAGE, MANY OTHERS.

[350,050] - KITTY BETHE MISCELLANEOUS TROUBLESHOOTING UTILITIES:
FILEID (GIVEN FILE ID, FINDS FILE),
FNDBLK (FINDS OWNER OF DISK BLOCK),
DSKZAP (DISPLAY/MODIFY CONTENTS OF DISK BLOCK),
TCI (TASK IMAGE COMPARE UTILITY),

VT100TST (TEST PROGRAM FOR VT100 FEATURES),
NUMEROUS COMMAND FILES TO DO NIGHTLY BACKUPS, ERROR LOG LISTINGS, ETC.

[350,060] - ROSS AMANN PSZ - PGM TO CREATE/REMOVE COMMON PARTITIONS.

[350,061] - ROSS AMANN RE-RELEASE OF RUNOFF FROM S81 TAPE.

[351,010] - JOHN LLOYD DISPLAY TOP CPU USERS ON VT100 JUST LIKE ON A VAX.

[351,020] - BENSON ACKERMAN COMPLEX RADIX2 FFT AND BI-CUBIC SPLINE PROGRAMS.

[351,030] - BRIAN NELSON WHYTED TEXT EDITOR FOR RSX AND RSTS.

[351,040] - DENNIS COSTELLO DESCRIPTION OF [351,41-51].

[351,041] - DENNIS COSTELLO ATP - ACTIVE TASK LIST WITH PRIORITIES.

[351,042] - DENNIS COSTELLO STOP - ABORT ALL ACTIVE TASKS AT TI:

[351,043] - DENNIS COSTELLO REW - REWIND A MAGTAPE.

[351,044] - DENNIS COSTELLO TAPE - SHOW STATUS OF TAPE DRIVE.

[351,045] - DENNIS COSTELLO PRV - SET/RESET TERMINAL PRIVILEGE.

[351,046] - DENNIS COSTELLO FILEDEF -- PROGRAM TO SET UP LUN-TO-FILE ASSIGNMENTS FOR FORTRAN-77 PROGRAMS WHICH DO NOT SPECIFY FILENAMES IN OPEN STATEMENTS.

[351,047] - DENNIS COSTELLO BATCH SYSTEM, WITH VT: DRIVER AND BATMAN MANAGER FOR RSX-11M V3.2 (SHOULD ALSO WORK ON V4.0).

[351,050] - DENNIS COSTELLO ACCOUNTING AND MISCELLANEOUS.

[351,051] - DENNIS COSTELLO UCB - DISPLAY UCB, DCB, SCB ADDRESSES OF ALL DEVICES.

[351,070] - VINCENT GRAHAM LIST - PROGRAM TO LIST FILES AT THE TERMINAL, WITH FILENAME DISPLAYED ON SCREEN OF VT52/VT100/TEKTRONIX SCOPE.

[351,71-73] - VINCENT GRAHAM RSXNET - PROGRAM FOR ASYNCHRONOUS COMMUNICATIONS BETWEEN SYSTEMS, USING A NULL MODEM OR A DIALUP LINE.

[351,074] - VINCENT GRAHAM VTM - PROGRAM TO FORMAT MESSAGES FOR DISPLAY ON THE VT100 SCREEN, USING ALL VT100 VIDEO ATTRIBUTES.

[351,075] - VINCENT GRAHAM RSXMSG - RSX ERROR MESSAGE MODULE, WILL RETURN MESSAGE STRING IN A BUFFER OR PRINT IT ON THE TERMINAL.

[351,076] - VINCENT GRAHAM WHO - PROGRAM TO SELECTIVELY LIST ENTRIES FROM THE ACCOUNT FILE. NOTE: THIS IS NOT THE SAME WHO AS IS DISTRIBUTED IN THE KMSKIT.

[351,077] - VINCENT GRAHAM USERS - PROGRAM TO DISPLAY LIST OF USERS CURRENTLY LOGGED IN TO THE SYSTEM.

[351,110] - J. F. VIBERT DATABASE MANAGEMENT FOR BIBLIOGRAPHIC REFERENCES, DOCUMENTATION, ETC. IN FRENCH.

[351,111] - J. F. VIBERT ANOTHER IMPLEMENTATION OF CCL, WRITTEN IN FORTRAN, DOCUMENTATION ETC. IN FRENCH.

[351,120] - DEREK FRANKS PATCH TO KED FOR MULTIUSER VERSION ON M+.

[351,130] - BARRY BREEN ADVENTURE WRITTEN IN OMSI PASCAL.

[352,002] - BOB TURKELSON TALK PROGRAM V06.00 FOR COMMUNICATION FROM SYSTEM WITH A FULL DUPLEX TERMINAL DRIVER VIA AN ASYNCHRONOUS SERIAL INTERFACE TO A REMOTE SYSTEM.

[352,004] - BOB TURKELSON BOB DENNY'S UPDATED VERSION OF SRD.

[370,130] - MICHAEL LEVINE
INDEX -- FORTRAN CROSS-REFERENCER,
SUPERMAC -- MACROS TO MAKE MACRO STRUCTURED LANGUAGE,
FRAG -- DISK FRAGMENTATION DISPLAY UTILITY,
HP11C PROGRAM LISTINGS FOR CONVERTING BETWEEN DECIMAL AND PDP/VAX BINARY FLOATING POINT FORMATS,
3D PLOTTING.

[374,001] - BOB DENNY BINARY KIT AND DOCUMENTATION FOR THE DECUS C COMPILER, WITH DEBUGGER, TOOLS, AND UTILITIES.

[374,003] - BOB DENNY LOADABLE XDT FOR RSX-11M V4.0, WITH BUGS FIXED AND TESTED WITH BL32.

[374,004] - BOB DENNY VIRTUAL DISK WITH ERROR LOGGING SUPPORT.

[374,005] - BOB DENNY FILES-11 REPAIR TOOLS & INFO.

THE MAGIC OF SYSGENING IAS

Mike Reilly - Development Manager with Digital for IAS.

Mike Garcia - Development Engineer with Digital for IAS

For the next few minutes we're going to be giving you a general overview of the IAS system generation procedure and the functions performed by each of the tasks which are included, or which are invoked, as part of the IAS system generation procedure. We are going to go into a little bit of detail in some areas but, in most cases, the concepts of what is done will be fairly straight forward and there is no sense for us to go into great detail to show, for example, how to search through a list, or how to create an entry in the list, and so forth, which is done in the standard manner.

The order in which we will be presenting the individual tasks will be the order in which you would normally see them if you were going through an IAS system generation. Beginning with the initial hardware boot of the system, we will go through a little bit about the boot block and the boot procedure to bring up an IAS system which has already been generated and then we will talk about System Generation Phase I, which is the task that creates a file on the disk. This file is what is going to be later brought into memory and is going to become your running IAS system. To bring up this system is an MCR task called BOO (for boot), so that will follow. After BOO we will talk about Sysgen Phase II, the second part of Sysgen which actually executes in the system which you have generated and then finally saving the system which you have generated so you will be able to hardware boot it and start the procedure all over again.

System generation is used to create a file on a disk which is a system image. A system image is simply a byte by byte copy of what is in memory. If you view a file as just the contiguous, or continuous, series of bytes beginning with Ø and continuing out through the end of the file, that is exactly what it is in this case. It is simply a byte that would be loaded into the memory location is in the identical location in the disk image. This file normally is called IAS.SAV and is normally created in a UIC of [11,17]; as we will see you have options of changing both of these.

As I said, Sysgen Phase I is the task that will create this file and turn it into an IAS system image. Sysgen Phase I is the most complicated part of the process because it must perform all the functions that would normally be done by an executive and several installed tasks on a running system. It performs all of its functions on a disk image, but it must do the same things that happen if you, for example, enter at a terminal the INStall command. Sysgen has to handle the install, parse the command, install the task and activate it if needed. There are a couple of tasks that it actually has to activate, so it even has to act as if it is the executive when it is writing out the disk image. The task BOO, which is normally called MCR BOO in the source listings, is a task that will simply read a block, the first 512 bytes of a disk image, into memory and then begin executing it. That disk image will continue the process of reading in the rest of the system image.

Sysgen Phase II, which is very simple, straight forward, is simply a command file process.

Finally, SAV. SAV has the task of taking an IAS system which is running in memory, writing it out to the disk in such a way that it can be brought back in at a later time and continue to run as if it had never been written onto disk at all.

As I said we will go through the tasks in the order in which you would normally see them, so we will begin with the BOOT process and Mike Garcia will stand.

The following information is a summary of what actually happens as a user is ready to bring up a running system. The information contained here is the boot block, information on reading the system image into memory, and starting the system. These points will be expanded upon, as Mike said, as we follow along throughout the talk. The information here is common for bootstrapping all RK, RL, RM and RP disks. Also included here is the power recovery trap vector used to restart the system in re-entering SAV, after boot of a saved system. The PDP-11 ROM bootstrap program reads the first block from the disk into memory at real address Ø. It then transfers control at address Ø. On the slide you can see the boot code being put up to real Ø including the address that it points to in the SAV entry file.

System generation has created an IAS image file on the disk with the first block of that file being a device specific bootstrap. Sysgen will have set several parameters into the bootstrap code including the base address of the exec, the disk address and size of the image file it is creating, in the first register address for the disk controller.

The MCR BOO function has copied this first block of the image file to block Ø of the target disk. When the ROM loads memory, the bootstrap program overlays itself with a copy of itself and, therefore can continue executing.

These following items will be initialized by Sysgen Phase I. The first one is two words for a logical block number of the SAVed file. Four words are reserved for power recovery trap vector and some space for the base address of the exec and the size of the SAV file in one K words. Some other additional parameters are required both by BOO and SAV. One parameter is provided to be stored by the BOO function to allow a boot to be performed from a unit other than Ø. And the following information is required by SAV, three bytes for a write-data function code, an offset to a place to insert that function code, an offset to the code to execute after booting a saved image and an offset to write the saved image. Two parameters, one for a saved user PARØ, another for the address it saved to return to, have been initialized by the MCR SAV function and used after boot of the SAVed image file. On the slide we can see the address in the boot code which points to the module, SAVENT, which is the address of re-entry to the SAV module. MCR SAV and BOO functions rely on all the parameters mentioned above being fixed offsets within the boot block, since they do not read in the symbol table. Sysgen Phase I, on the other hand, does read in the symbol table.

When the system is booted into memory, via either the MCR BOO function or the ROM, the bootstrap program does an indirect jump to the power recovery vector at real address 24. This causes the execution of code which will set up the kernel active page registers followed by a jump to the power recovery routine of the exec, called POWER.UP. The first time the output of Sysgen Phase I is booted, Sysgen Phase II will eventually be found active and control passed to it.

As mentioned previously the start of the bootstrap is at address Ø. The RAD50 word sys, S-Y-S, identifies the boot block. At this point we are now ready for real booting to be done. Note that SAV will also use this code to write the saved image back out to disk, by changing the function code to a write. Disk specific stuff is done at this point to finish reading in the rest of the system image. If there is non-existent memory you must start all over again.

If all is successful and there is no more memory to transfer, we now set up the PSW

and return to restart the system. By moving kernel priority seven into the PSW, we simulate a trap via the power recovery vector. Power recovery trap vector must be addresses 24 and 26. SAV modifies this trap vector by putting the address of a second routine in the bootstrap code into location 24 so that the bootstrap will transfer control via that routine to the SAV task. When SAV gets control back it will restore the vector to its original contents.

The SAV subroutine sets kernel APRØ and APR1 for 4K read/write and maps both to real memory Ø. It then sets up user PARØ, 4K read/write, and maps it using a saved APR value and also sets up APR7 to the I/O page. Memory management is now started, followed by setting up of the executive stack pointer. PS and the PC are set up to return to user mode via an RTI and SAV will then bring up the system.

The third and last routine in the bootstrap will be used in the event of a real power recovery. This routine sets up kernel APRØ, APR1 and APR7 for 4K read/write operation and maps APRØ and APR1 to real zero. APR7 is set to external page and memory management is then enabled. It will jump to APR1 by adding 20,000 to the program counter, restores kernel APRØ and goes to the executive power-up routine.

Now, that is just a summary of what's going to be mentioned throughout and Mike [Reilly] will continue with Sysgen one.

As soon as an IAS system image is read into memory, either one block at a time, as some of the boot blocks do, or the entire system image in one read request, if possible, otherwise in some cases a large system is broken up into as large pieces as possible. As Mike mentioned the bootstrap code sets up some kernel mapping registers so that it is possible for SAV, which is the next part of the bootstrap process, to access both itself, which is its task code, and various parts of the system that it needs to get to. Initially control is transferred from the boot block to SAV through the SAVENT entry point which was mentioned. It is simply hardwired into the boot block and when the code reached that point returns it to SAV. SAV is running in user mode, with access to the boot block. SAV uses the boot block to determine the system device and the unit number which it will use later. The first thing SAV does, when it begins executing, is what it calls an ECO test. There is a test to determine that an 11/40 processor has been properly ECO'd, which means there is a necessary change to the hardware that has been applied. If that test passes, then SAV will continue to set up memory parity registers and the stack limit register if they exist. These are options of various PDP 11's, so SAVE will determine if, either the parity registers, or the stack limit register or both exist, and set them up appropriately. SAV is also capable of generating values for these registers if the system initially was used on a machine that did not have these registers so there are no saved contents.

Following this, SAV will restore the power fail vector. As Mike mentioned, the power fail vector is used to gain entry to SAV when it saves the system. When it writes it out to the disk image it over writes the contents of the power fail vector with the entry point into SAV. It will now restore the original power fail vector, so that if a power recovery occurs it will be handled normally. Mike mentioned that the boot block is read in at real memory address zero, the executive, however, does not begin at the bottom of memory. As shown in this diagram, and this one will be used later on also, the executive virtual address zero begins at the end of the boot code. So on a power fail, the virtual zero, actually virtual 24, which is an offset to virtual zero, will be used and then the value that SAV restores is used on power recovery.

Following this, SAV restores the memory management registers. When the system was

initially saved, written out to the disk for the first time, SAV created a stack of all the hardware registers that it could find at that time, included were memory management registers. So these are restored both in kernel mode and user mode so that the hardware is set up in the same manner, the same fashion it was when the system was previously running. Following this there is a check for an 11/44, 11/70 processor, which are 22 bit CPU's that require special registers, called UMRs (for Unibus Mapping Registers). These registers allow Unibus peripherals access to the entire memory available in the larger machines. SAV has been set up to restore the UMR's on the machine that you originally ran SAV, in other words when you first saved the system, if you were on a machine with UMRs they were saved. SAV will use those values, if you are again running on a machine that requires the unibus mapping registers. If not, then SAV will generate UMR values and load these into the registers so that you can take an IAS system, generated onto another machine (for example an 11/40, 11/34), save it and then, when you boot it on a 22 bit processor (an 11/44, 11/70), SAV will generate the necessary UMR content values and the machine will run as if you had initially used it on the 22 bit machine. SAV will also perform the opposite. If you save the system on a 22 bit processor and run it on one of the smaller machines it will skip the UMR values that it saved and simply not restore them since there is no place to put them.

The next thing that SAV does is size memory, determine the size of memory in the hardware configuration it has, and adjust any partitions as needed. If your last partition in the system is the GEN partition it will expand and contract it as needed, setting up the appropriate data structures to indicate the current size of the partition. SAV is also capable of completely eliminating any partitions that no longer exist. For example, if you have a partition which starts beyond the end of memory, on your current configuration, SAV will simply eliminate that partition, it will just not exist any more. It will also have to remove any tasks which were installed to run in that partition.

Following this SAV has to check for the system clock. When the system is generated you can specify one of two types of clocks, a line clock or a programmable clock. SAV will initially check to see if the same clock that you generated for is present in the system, if so, it will be used. If not SAV will test for the other type of clock, the one you did not specify. If that is present, it will be used. It will be properly initialized by SAV and the system will continue as if the correct clock had been found.

Following this, SAV uses the information that it picked up from the boot block on the device which was bootstrapped (the unit number, device name), searches through the system data base and redirects the device SY: to whatever physical device was booted, so that if you save on a unit 3 of an RP06 for example and then you boot on unit zero, SAV will modify the system so that the booted device SY is now unit zero.

At this point the system is ready to run, so SAV enables task switching, declares a power fail AST and then, its final step, the one which seems to take the longest time and if you watched your disk, if it has any type of indication as to what it is doing, you'll see it's very busy. What SAV does is it reinstalls each task which was installed in the system when it was saved. Not reinstalled as if you had installed it, but what it actually does is it has to modify an indication stored in the system which points to the task image so that it now points to the disk address. When a system is moved as part of the copy procedure for DSC or BRU, it is necessary that the next time you boot the system all of the tasks which were installed can be located without the system having to know the file name of each task, go out and locate it through the file system and reinstall it. So what is done is the file system identification number is saved in the image on the disk. SAV goes through and takes each task and its identification number, converts it to the disk address, and sets the disk address into the data base in memory. So that at this

point it is now possible to find any task image directly on the disk, as it was when the system was previously running.

The last thing that happens is SAV prints out its message, its identification, IAS version 3.1, memory size, it tells you whether a partition or partitions were expanded or contracted and gives you other information that is needed. For example if it had switched the system clock it will tell you that it did so. The last thing that it will do will prompt for date and time, allow you to enter a date and time, and then exits. At that point you have an identical IAS system with what was running before you initially entered the SAV command and wrote it out to disk. The power fail AST which was declared by SAV will cause the executive to enter any task or device handler which has declared a power fail AST, that task or handler will be able to reinitialize itself to the new hardware configuration. If it is a device handler it will normally determine if the peripheral that it is communicating with is still present, it will determine whether it is still running with UMR's as it was before, if it needs to allocate them and so forth. And the system continues as if it had never been written out to the disk.

Now, the next thing that you are going to do, or that we are going to assume is going to happen, is that you wish to generate a new IAS system image. This is done with a task which is called SYSGEN Phase I or SGN1. As I mentioned before, SYSGEN Phase I or SGN1 is the most complex part of the system generation procedure. It is working with a file which it has to use as if that file was memory. It has to know that this file is a copy of memory and that it will contain both real and virtual addresses. As shown here, it has to know when it writes things into the disk image, it has to know where on the disk to write them which corresponds to the real address, when it is setting up the pointers in the system it has to know the virtual addresses that the executive will later use to access all the data bases that it needs to get to. Normally a sysgen procedure begins by installing a task which is given the name INV, this is a virtual install task. This is a task that installs other tasks just as the INS command does on a normal IAS system, except this task reads and writes the disk file. All the processing which it must go through is the same processing that must go through when you install a task from a terminal. After that task is installed assuming you have edited the command file which is going to be given to SYSGEN Phase I or you are going to enter the commands one by one from the terminal, you will run the task SGN1.

SGN1 will begin by prompting for a device and file name. This allows you to specify the name of the IAS system and what disk it is to be written on. Normally you're going to do a system generation onto the current system disk. However if you wish to make a system bootable on another disk, then you specify that disk in the initial prompt to SYSGEN Phase I and it will use files on the disk you specify, instead of on the system disk. The system generation manual for IAS describes all of the individual directives, the commands that are given to SYSGEN to indicate what devices you wish to include, what tasks are to be installed, what processor you are using. All the necessary information to create an IAS system. These parameters are entered either through the command file or a terminal, they are parsed by SYSGEN Phase I for correct syntax to make sure all is valid and to make sure that you don't do things like trying to allocate the executive and a partition to the same area of memory. SYSGEN then creates an internal data base of all the things it's going to be doing to this disk file.

Following this, SYSGEN1, having read the commands either from the file or from your terminal, knows which disk you are going to be using for the IAS system you are going to build and it looks for a file which contains the bootstrap code for that disk. It will read the STB file, which was created when this boot code was task-built. It will look for the offsets from the beginning of the boot block where various bytes of information need

to be plugged in. SYSGEN Phase I will always read the STB file for the corresponding disk that you're going to gen onto.

Following this, it will look for a file called EXECUTIVE.STB, a list of all the executive symbols that it needs to know. The last file it looks for at this point is the executive task image itself. The IAS executive is overlaid, so that SAV has to know the size of the main segment of the executive and of each of the overlays, so it reads this by reading the executive task image file. After it completes this, it will determine whether you have used a base address for the executive in one of the parameters, it will set up the proper real memory address with the executive to be loaded into, and take into account the size of the bootstrap. As shown in the diagram, the boot code always remains in memory, below the executive. SYSGEN knows the size of the boot code, it is one of the symbols it reads from the STB file, so it knows where to place the executive.

Following this, it allocates the mapping for the overlays in the system. SGN1 finds where each overlay will be placed and what memory management information is needed, so that when the executive is running it will be able to access the overlay it needs.

Once SAV has all of this information it is ready to create the system image. It begins by opening the system image file, actually creating a new one, using either the name you specified or the default IAS.SAV. It then attempts to write the last 32 words of the file which causes the file system to allocate the file to the proper size that you specified. A system image must be contiguous and SYSGEN has to have access to any word in the file in any order, so it has to make sure that all of the file is allocated and it does that by simply writing the last 32 words.

The first thing that gets written out into this file is the executive. It is copied, block by block, from the EXECUTIVE.TSK file into the IAS.SAV file. Using the virtual addresses that SYSGEN 1 calculates, the overlays and the executive will be moved into the disk image, into their proper positions. Also in the executive, there are certain values that have to be filled in by SYSGEN and these are filled in as the appropriate block is copied out to the disk.

The next thing it does is begin to generate the data base, beginning with the devices that were specified during the beginning portion of SYSGEN, when it read commands that were entered. It takes a standard physical unit directory entry, which it maintains in the task image, fills in values from tables it has, or from input given during the SYSGEN process, and writes these out into the area in the system that will become the system common area. It allocates these physical unit directories, or PUDs, entries in the reverse order that they were specified during the initial reading of commands, so that the last device which was specified will have the first PUD entry. In addition the entries for a pseudo-device CO:, which is the console terminal, CL: which is the console list device, and TTØ: will be created. TTØ: should have been specified as one of the parameters entered during the early parts of SYSGEN, so what the SYSGEN process will do at this time is simply test to make sure you have the TTØ:. If TTØ: is not found it is considered a diagnostic, not a fatal, error and SYSGEN will continue. It really doesn't care if you have a terminal handler or not. However, it doesn't give instructions in what to do with a system without one.

Following this, it installs all of the global common areas that were specified. Normally this is an FCS common area, possibly an IAS common area and SYSRES, the system common area. These are installed by SAV going through the procedure of creating the data structures necessary in the system common area so that when the system is brought up, the tasks for these common areas look as if they had been installed by install commands from

the terminal. Once this is done SAV can start working on the partitions and it will go out and generate the data base needed for each partition which was specified. Depending upon the number and type of partitions there are many different things that SAV will do. The end result is it will have the necessary data base set up for all of the partitions or it sets a special flag byte indicating it can not set up some of the partitions and it will be handled after the system is booted, during SYSGEN Phase II. There is a check then for the system disk being installed. It is a fatal error if there is no system disk and a fatal error if SYSGEN Phase II has not been installed.

Memory is then allocated for two tasks. SAV has to install, load and activate two tasks in the disk image so that when it is brought into memory these two tasks will become active. One of them is the system disk handler, the other one is SYSGEN Phase II. They are both placed in a special state indicating that they have just been loaded so that when the system is brought into memory the executive will see this state and it will begin to activate the tasks; allow them to run.

SYSGEN then goes back to the system common area, creates an alpha table, which is a fixed length table of tasks by name and a pointer to the data structure. This list is alphabetical so that the task can be found quickly when you wish to find an installed task. To go with each of these tasks that is installed is an entry called an STD, which is simply the data base for a task which is installed in the system. This is also created by SYSGEN.

Finally, SYSGEN has to go through and create another data base for tasks which are active, this is called a ATL. It has to do this for the system disk, and for SYSGEN 2. It also does it for the special tasks used for timesharing. There are 3 of them. One of them, TSS1, the second one TSS2 and the third one is a TSNUL1. These 3 ATL entries are always created in SCOM, whether you generate timesharing system or not. They are not marked as active. There are special pointers in the system common area that will allow the executive to find these ATL entries when you bring up timesharing.

After this the data base is complete and SYSGEN writes out the remainder of SCOM. It has a copy of the system SCOM data base in the sysgen task. All it does is take this copy and write it out to the disk. All through the SYSGEN1 process it has been filling in whatever values were needed, so it nows writes out all of the pointers, list heads, all of the little bytes of information that are needed when the system is going to begin running.

The last thing that SYSGEN Phase I does is it takes the boot block, the boot file which it already found and copies the boot block itself to the beginning of the IAS system image it just created so that when you are finished you will have on the disk, a copy of the boot block: It is hardware dependent, it will only work on the device which you specify. This is followed by the IAS image which is going to be loaded into memory by this boot code. SYSGEN has completed all of the necessary bits and pieces it's supposed to handle. It prints out its last message which says 'End of SYSGEN Phase I' and it is completed.

During this process it has communicated with the virtual install task INV. SYSGEN behaves as if it is MCR in this case, in that it will insert its own command into the MCR queue for INV and then request that INV be activated. INV will read whatever install command is present and execute it. Before INV is requested for the first time, SYSGEN Phase I allocates a node in SCOM on the running system, so this is now in real memory, and it fills it in with the values that are needed by INV to find various locations and offsets in the disk image. Things like the logical block number of the system image, the size of the boot code, the size of the various executive overlays, and where to find the

information it needs in the disk image. INV will go and read this node, to pick up all the information it needs each time it's called to install a task.

After SYSGEN1 has closed all of its files and printed out its message it is then ready for you to bootstrap the system. The MCR BOO command is what will handle that and Mike [Garcia] will take over.

Ok, now that SYSGEN I had completed we're ready for the boot process using the MCR BOO command, to boot up the image just created by SYSGEN 1 and to get the bootstrap up to real memory zero. The intent of the MCR function BOO is to perform an initial boot of an IAS system image. It will simulate the function of a hardware ROM from any device. Alternatively, it will write the bootstrap block zero on a specified device, by default the latest version of IAS.SAV is the image booted into the memory. The specified file to be booted must be an IAS system image created by SYSGEN 1. Boot will check this out to see if it is the correct file.

Before going into detail about what boot does, a summary of its operation is it first operates validates command syntax, it reads block one of the IAS image file, verifies it as an IAS system image, closes the IAS image file, and checks the privileges of the requesting terminal to see if it is privileged. If the user is attempting to boot, the terminal must be a privileged terminal. If the user is attempting to write a bootstrap block, the object device must not be SY: or redirected to SY:, unless the terminal is privileged.

There are 2 choices on performing the MCR boot function. First choice is the MCR command BOO with the optional file specification, which inserts the device unit number into the bootstrap block to be moved. It moves the whole block to real zero and up. You see the boot code going up to real zero [on the diagram]. It moves what we call special instructions to real 1000 and up and jumps to the special instructions which enters kernel mode, disables memory management, and then jumps to real zero. The second choice is to issue the MCR command BOO along with the /WB switch where virtual block one is to be copied from the specified file to bootstrap block zero of the disk device. As Mike mentioned BOO is linked to the STB file for whatever device is being genned so it knows offsets in the boot block.

The boot procedure starts by getting the boot command line. It will validate the syntax and parse the command. If there is na syntax error, BOO will exit. It will then check to see if the device is a random an access device and also if it is a directory device since we can not boot say a line printer or a terminal. If all is valid we now proceed to open a file to read the boot block, which is the first block of the IAS image file. We now have the first block. If it was written by SYSGEN or SAV, the first word is a Branch instruction and the second word must be "SYS" in RAD50. One of the offsets in the boot block is the base APR address of the base of the executive. This address must be between 200 and 777700, therefore the offset itself must be between 2 and 7777. Boot validates the specified file image by checking all those facts. If all is OK, the starting LBN of the system image is now written in the boot block. In other words, the address of the system image is stored into the copy of the boot block just read in. On the slide again we can see the LBN in the boot code which gets moved up.

Boot checks the privileges of the requesting terminal. If the terminal is privileged, boot will allow anything to be done. If it is not privileged it will disallow the actual boot or the writing of block zero of the system device. If the privilege is OK and the user specified the WB switch, the boot block is then written. If the WB switch was not specified we move to the routine that moves the device unit number into the bootstrap

block. It then moves it all into low memory, starting with real zero, followed by some special boot instructions. When done, we jump to real zero to perform the actual boot.

Now to get at real zero, we need to map an APR to real zero, 4K read/write. Since we are running under user mode APRØ, we will use APR 1 and map it to real zero and inhibit all interrupts. Twenty thousand is then moved into APR 1 before the routine goes into a loop moving one word at a time to real zero. We then jump to the special instructions at real address memory 1000, but since it is mapped to APR 1 it is 21000 for the boot procedure. Special instructions are there to get us into kernel mode by clearing the current mode bits that are in the PSW. They set kernel APRØ and 1 to real memory address zero and set 4K read/write for both. The twenty thousand bit in the PC is cleared and we then set the starting address to the bootstrap, do a reset and jump to the starting address to start the boot. That is the end of the special instructions and that is the end of the MCR BOO procedure and we are ready to get SYSGEN Phase II into the system.

Believe it or not, you sit there and you are ready to do SYSGEN2 and the message comes up for SYSGEN generation Phase II, but there is an awful lot that goes on before that message is printed. SYSGEN II is installed and loaded by SYSGEN I such that booting the new system disk results in the running of SYSGEN II. SYSGEN II is loaded at the top of the specified partition to avoid memory fragmentation. You can see where SYSGEN II resides in memory on the slide.

SYSGEN II performs the following functions automatically. It checks to make sure there is sufficient memory available, if there isn't it will just exit. It checks to see that system disk handler is active, requests and loads a TTY handler and mounts the system disk, fixes Files-11 ACP, and after mounting the system disk it will open a file called SYSBLD.CMD and execute all the commands found in that file. Note that any task required by SYSGEN II must be installed before SYSGEN II uses it. Also note that after SYSGEN Phase II exits, the user should do a SAVE to preserve the system as generated. If the SAVE is not performed, the IAS image file will still contain the output of SYSGEN Phase I. That is, a runnable IAS system with SYSGEN Phase II activated. Note that if it is desirable to boot the IAS system, the MCR function BOO must be used with the WB switch to initialize the boot block zero of the system image.

SYSGEN II begins by finding the system size as specified in SYSGEN I. Previous mode bits of the PSW are again set to kernel, and the PSW is then saved. If we run out of memory, that is less memory is specified during SYSGEN I, SYSGEN II traps at priority 7 and prints the diagnostic message: "All memory specified does not respond." If memory is OK, SYSGEN II will inhibit interrupts, set APR 3 to map over the bootstrap, in other words set PAR 3 to map to zero for booting. A call is made to subroutine SPD 3 to prevent APR 3 from being modified during task switching. APR 3 must be restored to finish.

The IAS system image has now been booted and a call is made to a subroutine to redirect the PUD for SY: to the appropriate unit of the system disk as it must be the device and unit number that will specify during SYSGEN I. It extracts this information from the offset in the boot block. At this point SYSGEN II gets a saved image size from the bootstrap and converts it to 32 word core blocks from memory management. SYSGEN II will scan the task partition directories setting up what SYSGEN I could not set up. If the SG flag is set it must be cleared. The SG flag is a bit from SYSGEN I that SYSGEN II has to look at to set up its hole pointers. The hole pointers are now adjusted and manipulated in a loop until all free space in the partitions is found. SPD 3 is called again to reset APR 3 and the stack is also reset at this point. This section of SYSGEN Phase II is completed by enabling interrupts.

SYSGEN Phase II continues by requesting the TTY handler. LUN 2 is assigned to TTY and LUN 3 is assigned to SYØ:. Here the "GET LUN" directive is used to find the physical system device. The routine will point to the buffer in order to move the device name into it. It then appends the period (.) as the third character in the device name, converts those 3 characters to RAD 50 and puts them back into the buffer. A check is made on the name of each task in the ATL to find the system disk handler. It was put into the ATL by SYSGEN Phase I. It will loop through the ATL notes until it finds the system disk handler. When the system disk handler is found, another check is done to see if it is active. If the system disk handler is not found the message: "Unable to find ATL or system disk handler" is printed.

When we find the system disk handler, SYSGEN II moves along and tries to find the STD entry of the TT handler in the ATL, and will loop until it finds it. When it finds the TT handler, SYSGEN II waits for the load request state to change, that is the TT handler to be loaded. It will check every 10 ticks until it is finally loaded. Once it is loaded, we wait for the TT handler to become active. Again, we wait 10 ticks until it becomes active. Once it is in memory, we wait 10 more ticks to give the executive time to complete the activation of the task. The TT handler is now in memory and in active state and resides directly below where you see the system disk handler. When the TT handler loads successfully, its ATL node will be charged to SYSGEN II. Consequently, as long as the TT handler is resident and therefore active, SYSGEN II can not be removed from the system. To circumvent this, it will now charge the TT ATL to TT handler itself, as SYSGEN I does when SYSGEN II and the system disk handler.

SYSGEN II is now ready, finally, to type the message: "System Generation Phase II." The Files-11 ACP is fixed in memory, and the fixed task list is scanned until it finds F11ACP. The routine will loop until it finds the STD for F11ACP and when it does find it, the ATLnode is charged to F11ACP and sets F11ACP as its own requester. SYSGEN II will decrement its own pool usage count. If F11ACP can not be fixed in memory we have the diagnostic message: "Error fixing F11ACP." That is only a warning, if it does not find that ACP, SYSGEN II just continues along. At this point SYSGEN II gets the PUD of the TI: in use and sets the UIC to [1,1]. TI: is set as logged on and privileged, and as the console terminal COØ:.

We now go to the subroutine to obey the contents of the SYSBLD command file to build the IAS system. In the SYSBLD subroutine, the first step is to create the command to mount the system disk, "Mount SYØ:/OVR". SYSGEN II now opens [11,17] SYSBLD.CMD, which is the default command input file and goes through each command one at a time. If there is an open error or read error on SYSBLD.CMD a fatal error message is issued and SYSGEN II will exit.

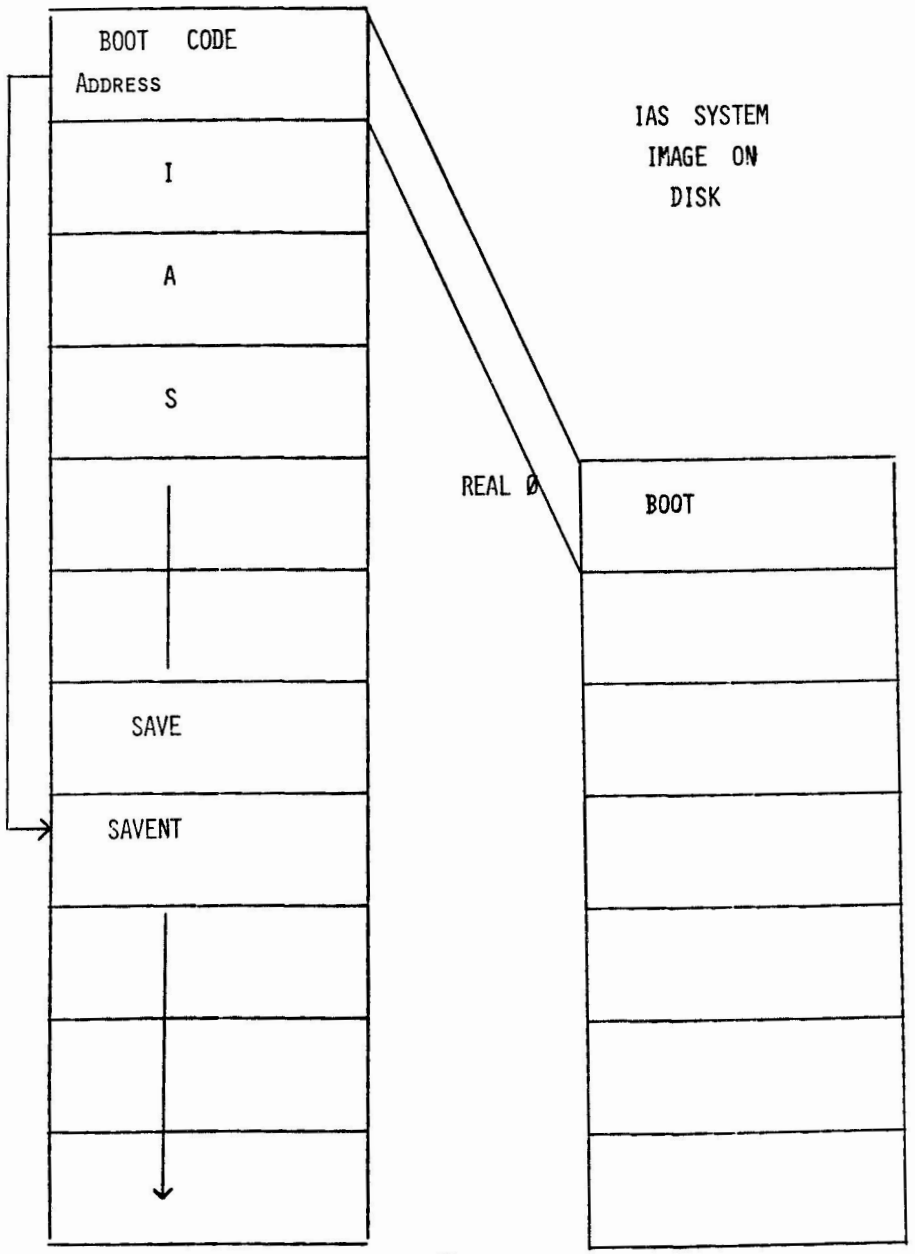
Another routine is called to obey an MCR type command line. This routine checks each command in SYSBLD.CMD to see if it is comment. It checks for both types of comments, the semicolon and the exclamation point, both types of these comments are ignored. It also ignores 2 special cases, the asterisk (*) Delay command and the "Log" command. These two commands may appear in older versions of IAS. The routine sets up the SPAWN DPB and executive request SPAWN is issued. It converts the first 3 letters of the command line to RAD50 and stores them in the SPAWN DPB as the name of the task to be SPAWNed, for example, PIP and INSTALL. If the MCR task is not installed, an attempt is made to run MCR's multifunction task, ...MFT, to do the job instead, and it will use MFT as the task name. If all this fails, a diagnostic error message: "Task not installed" is printed and SYSGEN II exits with a Fatal request error.

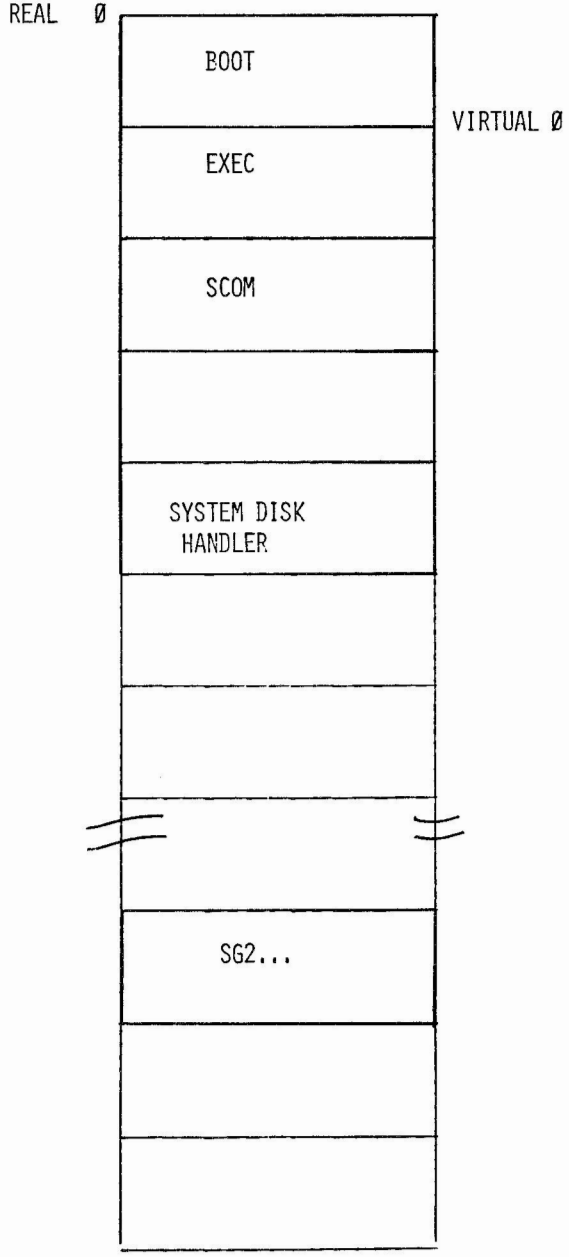
Finally SYSBLD.CMD is closed when end of file is reached. SYSGEN II is now ready to

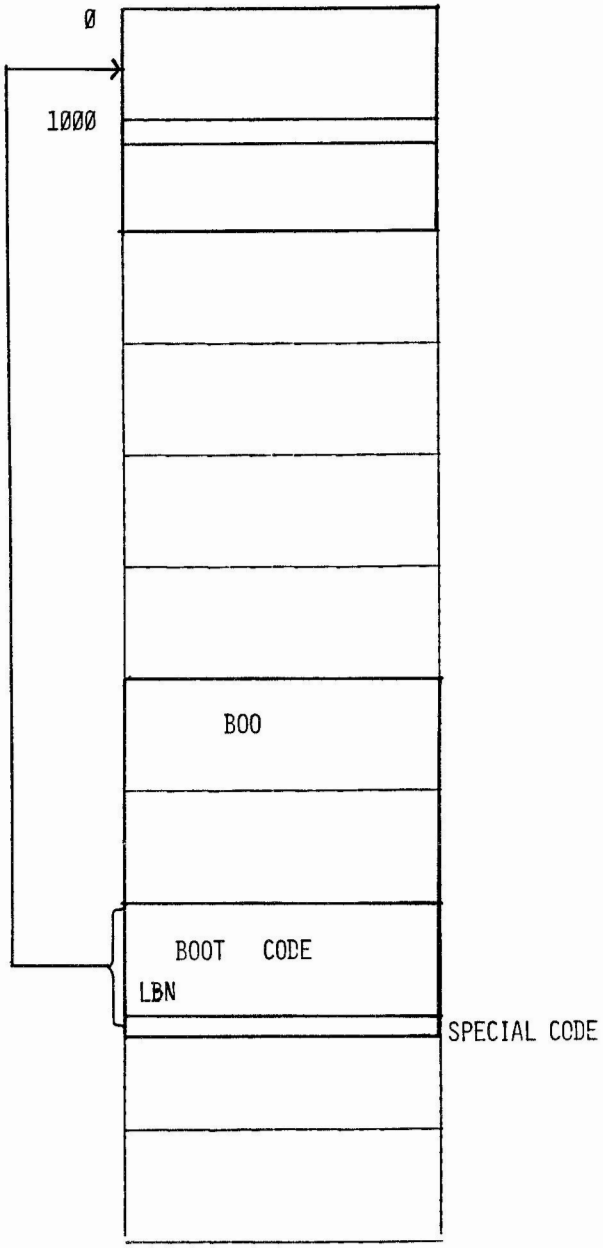
print the "End of System Generation Phase II" message and will then exit. After SYSGEN II completes and exits the user should always perform typical post SYSGEN2 functions and then save the system. Mike [Reilly] is ready to discuss SAV.

We started with the SAV task and we end with the SAV task. This time the job of SAV is to write the contents of memory out to the disk. SAV will do this by verifying that the system is quiet, that nothing is happening, that all users except the user who is actually attempting to save to the system are logged out, all disks have been dismounted, that there is no activity in the SEND/RECEIVE queues, no tasks that are waiting to be loaded into memory and basically, no activity within the system. There are qualifiers to the SAV command when you enter the SAV command to override some of these checks. SAV then will map itself to the boot block that was left in memory by the BOO command, use the logical block number found in the boot block to know where on the disk memory is to be written, use the size of the system image found in the boot block to know how much memory to write, and it will convert the read function code in the boot block to a write function code. Since it is identical code, it will just write instead of read. SAV then builds a stack to store all of the hardware registers that are going to be restored when the system comes back up. It then jumps to the boot code so that the boot code will be executed, and the system image will be written out to the disk. That completes the System Generation discussion.

Cervantes Convention Center
St. Louis, Missouri
Thursday, 26 May 1983




REAL Ø
BOOT
EXEC
SCOM
SYSTEM DISK
HANDLER
SG2...
VIRTUAL Ø


Ø
1ØØØ
BOO
BOOT CODE
LBN
SPECIAL CODE

LIST File Listing Utility

Written by:

William Wood
The Institute For Cancer Research
7701 Burholme Ave.
Philadelphia, Pa. 19111
(215) 728 2760

Version 2

LIST is a utility for displaying selected portions of a file. It provides facilities for displaying lines, positioning in the file, and searching for character strings. In addition, output from LIST can be redirected from the terminal to a file.

This manual explains how to use LIST. Throughout the manual, optional parts of a command are enclosed in square brackets.

1.0 Running LIST

The LIST command line has the following syntax:

LIST[/switches] fspec[,fspec...] [commands][>[>]file]

2.0 Switches

/HD	Write a header record containing each file's name and the date to the standard output before listing each file.
/GO	List all files selected by fspec without prompting (see below: Fspec).

3.0 Fspec

Fspec may be a simple file name, an indirect command file name preceded by @, or, in some installations, a file name containing SRD wild card characters or switches. If the file name is an indirect file name, LIST will list the named files in the indirect file one at a time. If it is an SRD-type file name, LIST will list the files selected by SRD. In either case, LIST will prompt you with each of the selected file names to see if you want it listed, unless /GO was selected (see above: Switches). Responses to the prompt are:

Y	List the file.
<cr> or N	Don't list the file.
G	List this and remaining files with no prompting.
^Z	Don't list this file and remaining files.

If commands are given on LIST's command line, they will be executed once for each file selected by the fspec. If the standard output is redirected by a command line, it will remain redirected for each file selected.

Note: Normal SRD wild card characters and switches may be used

in an SRD-type file name; however, do not use the /SE: switch, as this is the default switch. Also, only the most recent versions will be selected unless ;* is specified.

3.1 Default Fields in File Names

More than one file may be listed during the same LIST session by typing LIST and then a carriage-return. LIST will prompt for a file name. After each file has been listed, LIST will prompt for a new file name.

Whenever LIST attempts to open a file, the device, uic, name, and extension are remembered and become the defaults. The defaults are used to replace missing fields of subsequent file names entered for listing. For example:

```
PDS> LIST
FILE? LB:[22,2]LIST.DOC
```

When you are finished listing LB:[22,2]LIST.DOC, if you wanted to see LB:[22,2]VG3.DOC, you would only have to type:

```
FILE? VG3
```

and LIST would supply the missing fields of the file name.

If LIST can't open a file after supplying the defaults, it tries the file name as entered. Initially, the defaults are SY: for the device and .LST for the extension.

4.0 Numbers

Numbers are used to position LIST in the file and as arguments to commands. Numbers always precede the command they affect. A number may be a simple number, one of several special line number variables, or a search pattern. The value of a search pattern is the number of the line that matches the pattern. Some commands optionally take two numbers as arguments; when specifying more than one number, separate the two numbers by a comma, e.g. 1,45 . In addition, numbers may be added and subtracted from one another, e.g. $-9 is the 10th line from the end of the file.

LIST scans the input on a command line from left to right. Whenever it encounters a number, dot (the current line number) is set to that number. Multiple line numbers may appear next to each other; LIST positions to each in turn, e.g. 1/SUBR/ searches for SUBR after line 1.

4.1 Line Number Variables

.	Dot is the current line number. Whenever LIST encounters a line number, the value of dot is reset to that line number.
$	Dollar is the last line of the file.
#	Sharp is the line number displayed by the most recent prompt.
@	At-sign is the line number of the top of the last screen displayed by LIST.
*	Star is set by the = command and is equal to the value of dot when the = command was last issued. * serves to mark a line of interest for future reference.
;	Semi-colon is equal to the current screen size.

4.2 Search Patterns

A search pattern is a string of characters bracketed by slashes (/) or backwards slashes (\):

/string/	causes LIST to search forward from dot for string.
\string\	causes LIST to search backwards from dot-1 for string.

There are several characters which have a special meaning when they appear inside a search string:

'	means that the following character is to be treated as itself, not as a special character.
%	specifies that the match must occur at the beginning of the line only. % itself must appear at the beginning of the pattern, otherwise it has no special meaning.
^	causes the next character to be interpreted as a control character.

Once specified, a pattern becomes the default pattern and may be invoked by // or \\. Patterns may be used anywhere a number may appear; the value of a pattern is the line number of the matched line.

4.3 Examples of Line Numbers

.-10	10 lines before the current line.
1,$	Line 1 and the last line of the file.
/SUBR/	First line which matches SUBR.
/%SUBR/	First line which matches SUBR at the beginning of the line.
\\	First line which matches the default pattern while scanning backwards.
/''/He''s/	First line which matches /He's
2/HELLO/-5	5 lines before first occurrence of HELLO at or after line 2.

It is possible to specify fairly complex line numbers; the value of a line number is always the last number evaluated. For instance,

```
1/SUBR/+1//,.+3/END/+1//
```

specifies two line numbers; the first is the second occurrence of SUBR at or after line one; the second is the second occurrence of END at or after 3 after the first line number.

5.0 Commands

LIST accepts commands on it's command line or when it prompts with the current line number and a ">" character. Commands are single characters and are preceded by zero, one, or two numbers which are arguments to the command. More than one command may be entered on a command line; LIST scans the command line from left to right, positioning to line numbers and executing commands.

There are two types of commands; those that print and those that affect LIST's state without printing. Commands that print usually print from dot (the current line number) unless two line numbers were specified, in which case printing occurs from the first line number. Other commands "eat up" the number(s) that are their arguments; in other words, dot is reset to the value it had before the command, with it's argument(s), was executed.

If a line number appears at the end of the command line with no command after it, the default command (P) is executed. Thus /SUBR/ is equivalent to /SUBR/P ; both cause printing to begin at the first occurrence of SUBR.

5.1 Commands That Print

P Displays lines from the file. Dot is always left at the last line printed plus 1. P is the default command, and is optional at the end of the command line.

P	Print as many lines as will fit within the current screen size (initially 23), starting at the current line (dot).
nP	Print as many lines as will fit within the current screen size starting at line n.
n1,n2P	Print lines n1-n2.

G Like P, G displays lines from the file, however with zero or one arguments G only prints one line, and the value of dot is not changed.

G	Print the current line.
nG	Print line n.
n1,n2G	Print lines n1-n2.

? Displays only lines matching the current search pattern.

?	Display a screen-full of lines matching the current pattern. If the end of the file is reached, dot is left at line 1, else dot is left at the next line matching the current pattern.
n?	Display a screen-full of lines matching the current pattern, starting at line n. If the end of the file is reached, dot is left at line 1, else dot is left at the next line matching the current pattern.
n1,n2?	Display all lines matching the current pattern between lines n1 and n2. Dot is left at n1.

Note: LIST usually changes most control characters to nulls before printing a record, however when two line numbers are specified or when the G command is executed, all characters are written out unchanged. Thus it is possible to write out portions of a file without losing any characters.

5.2 Other Commands

L The L command causes the last command line to be reexecuted.

= The = command sets the line number variable, *, to dot. * serves to mark a line for later reference.

=	Set * to dot.
n=	Set * to n.

S Sets the screen size.

S	Reset the screen size to 23.

nS Set the screen size to n.
n1,n2S Set the screen size to (n2-n1)+1.

C Sets the column range which will be read from the input file.
C Reset the column range to 1-512.
nC Set the column range to 1-n.
n1,n2C Set the column range to n1-n2.

F Create a "virtual file" by restricting LIST to a contiguous subset of the lines in the file.
F Reset the virtual file to correspond to the actual file.
nF Make a virtual file between dot and dot+n-1. Dot becomes the new line 1 of the virtual file.
n1,n2F Make a virtual file between n1 and n2. N1 becomes the new line 1 of the virtual file.

N Turn prompt mode on/off. When prompt mode is off, LIST will not prompt with the current line number, but will leave the cursor after the last line printed. If the screen size is set to one and prompt mode is set off (ISN accomplishes this) then LIST will be in line-by-line mode, in which one line is printed for each carriage-return.

R Resets the screen size to 23, the screen width to it's original value, the column range to 1-512, the virtual file to the full, actual file, and prompt mode to on.

X Finish listing the file. X is identical to ^Z (EOF).

" " Space (blank) is the null command, and does nothing.

5.3 Installation Specific Commands

Not all installations have the following commands.

W Sets the screen width.
W Reset the screen width to it's original value.
nW Set the screen width to n.

H Invokes LIST as a subtask to print a help file.

V Routes all the following output from the command line to the printer port of the DT80 (a VT100 look-alike).

6.0 Defining and Using the Macro

LIST has a simple text replacement macro facility. The macro is remembered text from a line of LIST commands. The macro is defined by enclosing the text in square brackets ([]). When M is specified on the command line, the M is replaced by the macro text.

For example:

```
[@+1//P]
```

defines a macro to search for the current pattern starting from the top of the last screen + 1, and print from there. To invoke the macro, type M on the command line.

The text of the macro is not executed when it is defined by enclosing it in square brackets. It is not executed until an M is typed. M may appear with other commands and line numbers on the command line; first the macro text replaces the M, then the command line is executed. In addition, M may be used more than once on a command line.

7.0 Redirecting Output From LIST

The output from a line of LIST commands, which normally goes to the terminal, may be redirected to a file. To write a new file, put >FILE at the end of the commands, where FILE is a standard file name. To append to an existing file, put >>FILE at the end of the command line. If FILE doesn't exist, it will be created. The carriagecontrol type of newly created files is identical to the carriagecontrol type of the file being listed, while that of appended files is the same as that of the file being appended to.

Once a file name has been specified using >FILE or >>FILE, that file name becomes the default file name for file redirection only, and need not be specified again, i.e. > or >> alone is all that is needed to redirect output to that file.

8.0 Examples of Commonly Used Commands

<cr>	Hitting return prints the next screen.
45	Start printing at line 45.
.-5	Print from 5 lines back.
-	Print starting half a screen back.
@-	Start printing one and one-half screens back.
<esc>	Hitting escape prints from 2 screens back.
$-9	Print the last 10 lines in the file.
/SUBR/	Locate the string SUBR and print from there.
//	// alone defaults to the last search pattern specified.
45/SUBR/	Locate SUBR at or after line 45 and display a screen.
/ERR/-5	Display starting 5 lines before next occurrence of ERR.
$-50/SUBR/	Look for a subroutine in the last 51 lines of the file.
1,10	Print lines 1-10.
/SUBR/,/END/	Print the next subroutine.
1/SUB/,.+5/END/	Print from SUB at or after 1 to END at or after 5 after SUB
/SUBR/?	Display lines containing SUBR.
?	Display lines containing the current pattern.
10,$>T.TMP	Write lines 10 through the end of the file to T.TMP.

LIST *.FTN /XVAL/,$?X would print all occurrences of XVAL in *.FTN.

DeVIAS Questions - Answers

Name: _____

Mailing
Address: _____
_____
_____

Phone
Number: ( ) _____-_____

Instructions

1) Complete relevant part of form
2) Mail to Editor, The DeVIAS Letter
3) Question and/or Answer will be published in next newsletter.

Date Submitted: _____/_____/_____

Question:

Answer:

NOTES