ALIAS MAINTENANCE AND EXPANSION GUIDE
VOLUME II

Submitted to:
Scientific Officer
Naval Center for Acquisition Research
NAVMAT 08
Washington, D.C. 20360

Attention: Dr. Thomas C. Varley

This document has been approved for public release and sale; its distribution is unlimited.
ALIAS MAINTENANCE AND EXPANSION GUIDE
VOLUME II

M.S. CAREY
J.C. KRUPP
J.M. KABAT

Distribution unlimited per Dr. Thomas C. Varley, Navy Office for Acquisition Research, ATTN: NAVMAT-08, Washington, DC 20360

Submitted to:
Scientific Officer
Naval Center for Acquisition Research
NAVMAT 08
Washington, D.C. 20360
Attention: Dr. Thomas C. Varley
This documentation explains the structure of the Acquisition and Logistics Information and Analysis System (ALIAS). With this documentation, the experienced programmer should be able to easily maintain and expand the ALIAS system. In addition, the manuals explain all standards to which ALIAS extensions should conform. For the non-programmer these manuals describe the philosophy of ALIAS and its extent and limitations.
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>iv</td>
</tr>
<tr>
<td>10.0 ALIAS UTILITIES AND COMMON DATA STRUCTURES</td>
<td>10-1</td>
</tr>
<tr>
<td>10.1 General Purpose FORTRAN Routines</td>
<td>10-2</td>
</tr>
<tr>
<td>10.2 FORTRAN Utilities for DBMS Usage</td>
<td>10-147</td>
</tr>
<tr>
<td>10.3 Builder-Callable FORTRAN Utilities</td>
<td>10-212</td>
</tr>
<tr>
<td>10.4 Builder Utility Screens</td>
<td>10-235</td>
</tr>
<tr>
<td>10.5 FORTRAN Include Files</td>
<td>10-242</td>
</tr>
<tr>
<td>10.6 Miscellaneous System Resources and Their Current Utilization</td>
<td>10-242</td>
</tr>
<tr>
<td>11.0 FORCE IMPACT ASSESSMENT MODULE</td>
<td>11-1</td>
</tr>
<tr>
<td>11.1 Purpose</td>
<td>11-1</td>
</tr>
<tr>
<td>11.2 Summary of Structure</td>
<td>11-1</td>
</tr>
<tr>
<td>11.3 Inputs and Outputs</td>
<td>11-5</td>
</tr>
<tr>
<td>11.4 Data Structures</td>
<td>11-15</td>
</tr>
<tr>
<td>11.5 Processing Logic</td>
<td>11-20</td>
</tr>
<tr>
<td>11.6 Files Used by the Force Impact Module</td>
<td>11-36</td>
</tr>
<tr>
<td>11.7 Summary of Interfaces</td>
<td>11-37</td>
</tr>
<tr>
<td>11.8 Subroutine Abstracts</td>
<td>11-37</td>
</tr>
<tr>
<td>12.0 MANUAL ASSIGNER MODULE</td>
<td>12-1</td>
</tr>
<tr>
<td>12.1 Purpose</td>
<td>12-1</td>
</tr>
<tr>
<td>12.2 Summary of Structure</td>
<td>12-2</td>
</tr>
<tr>
<td>12.3 Inputs and Outputs</td>
<td>12-7</td>
</tr>
<tr>
<td>12.4 Data Structures</td>
<td>12-28</td>
</tr>
<tr>
<td>12.5 Processing Logic</td>
<td>12-35</td>
</tr>
<tr>
<td>12.6 Files Used by the Assigner</td>
<td>12-62</td>
</tr>
<tr>
<td>12.7 Interfaces</td>
<td>12-62</td>
</tr>
<tr>
<td>12.8 Subroutine Abstracts</td>
<td>12-63</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS
VOLUME II (Continued)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPENDIX</td>
<td></td>
</tr>
<tr>
<td>-A- CROSS REFERENCE FROM AUTOMATED DATA SYSTEMS DOCUMENTATION STANDARDS CONTENTS TO ALIAS GUIDES CONTENTS</td>
<td>A-1</td>
</tr>
<tr>
<td>A.1 System/Subsystem Specifications</td>
<td>A-2</td>
</tr>
<tr>
<td>A.2 Program Specifications</td>
<td>A-3</td>
</tr>
<tr>
<td>A.3 Data Base Specifications</td>
<td>A-4</td>
</tr>
<tr>
<td>A.5 Computer Operation Manual</td>
<td>A-7</td>
</tr>
<tr>
<td>APPENDIX</td>
<td></td>
</tr>
<tr>
<td>-B- SUMMARY OF ALIAS HOST SYSTEM DEPENDENCIES</td>
<td>B-1</td>
</tr>
<tr>
<td>B.1 Relate Dependence</td>
<td>B-1</td>
</tr>
<tr>
<td>B.2 Builder Dependency</td>
<td>B-3</td>
</tr>
<tr>
<td>B.3 HP 300 FORTRAN Dependency</td>
<td>B-3</td>
</tr>
<tr>
<td>B.4 MPE Dependencies</td>
<td>B-4</td>
</tr>
<tr>
<td>Table Number</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>10-1</td>
<td>Types of General-Purpose FORTRAN Utility</td>
</tr>
<tr>
<td>10-2</td>
<td>General-Purpose FORTRAN Utilities by Type</td>
</tr>
<tr>
<td>10-3</td>
<td>General-Purpose FORTRAN Utilities by Name</td>
</tr>
<tr>
<td>10-4</td>
<td>FILOPN Directives</td>
</tr>
<tr>
<td>10-5</td>
<td>DBIFA High-Level Routines</td>
</tr>
<tr>
<td>10-6</td>
<td>DBIFA Internal (Low-Level) Routines</td>
</tr>
<tr>
<td>10-7</td>
<td>BUILDER-Callable Routines in the SL</td>
</tr>
<tr>
<td>10-8</td>
<td>ALIAS Extra Data Segment Usage</td>
</tr>
<tr>
<td>10-9</td>
<td>ALIAS LPRNTS Usage</td>
</tr>
<tr>
<td>11-1</td>
<td>Include Files Used by the Force Report Generators</td>
</tr>
<tr>
<td>11-2</td>
<td>Alphabetical Listing of Routines in FLRP</td>
</tr>
<tr>
<td>11-3</td>
<td>Annotated List of Force Level and Battle Group Report Generator Routines</td>
</tr>
<tr>
<td>11-4</td>
<td>Alphabetical Listing of Routines in BGRP Program</td>
</tr>
<tr>
<td>12-1</td>
<td>Include Files Used by the Assigner</td>
</tr>
<tr>
<td>12-2</td>
<td>Annotated List of Assigner Routines</td>
</tr>
<tr>
<td>Figure Number</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>10-1</td>
<td>MPECOMMAND Utility Screen</td>
</tr>
<tr>
<td>10-2</td>
<td>RUNTDP Utility Screen</td>
</tr>
<tr>
<td>10-3</td>
<td>RUNEDITOR Utility Screen</td>
</tr>
<tr>
<td>10-4</td>
<td>SEARCH Utility Screen</td>
</tr>
<tr>
<td>11-1</td>
<td>Sample Force Level Report Generator Output</td>
</tr>
<tr>
<td>11-2</td>
<td>Sample Battle Group Report Generator Output</td>
</tr>
<tr>
<td>11-3</td>
<td>Structure of Force Analysis Modules</td>
</tr>
<tr>
<td>11-4</td>
<td>Command System Parameter and List Menus Serving the Force Module</td>
</tr>
<tr>
<td>11-5</td>
<td>Sample Force Level Report Format Control File</td>
</tr>
<tr>
<td>11-6</td>
<td>Sample Battle Group Report Format Control File</td>
</tr>
<tr>
<td>11-7</td>
<td>FLRP Calling Tree Diagram</td>
</tr>
<tr>
<td>11-8</td>
<td>FFLTBL Flow of Execution Summary</td>
</tr>
<tr>
<td>11-9</td>
<td>BGRP Calling Tree Diagram</td>
</tr>
<tr>
<td>12-1</td>
<td>Assigner Flow of Control Structure</td>
</tr>
<tr>
<td>12-2</td>
<td>Assigner Initialization Structure</td>
</tr>
<tr>
<td>12-3</td>
<td>Narrative Summary of Assigner Initialization</td>
</tr>
<tr>
<td>12-4</td>
<td>Summary of Assigner Editing Commands</td>
</tr>
<tr>
<td>12-5</td>
<td>A Typical Assigner Display Page</td>
</tr>
<tr>
<td>12-6</td>
<td>Assigner Editor Structure</td>
</tr>
<tr>
<td>12-7</td>
<td>Assigner Data Base Update Overall Structure</td>
</tr>
<tr>
<td>12-8</td>
<td>Schedule Creation and Update Structure</td>
</tr>
<tr>
<td>12-9</td>
<td>Sample Schedules from Ncjdat.proj Relation</td>
</tr>
<tr>
<td>12-10</td>
<td>Text of INIASN.SYSRO Configuration File</td>
</tr>
<tr>
<td>12-11</td>
<td>Sample Assigner Parameter Menu</td>
</tr>
<tr>
<td>12-12</td>
<td>Sample Assigner Valid Job Types List Menu</td>
</tr>
<tr>
<td>12-13</td>
<td>Sample Job Description Records from Ncjdat.descj</td>
</tr>
<tr>
<td>12-14</td>
<td>Assigner Initialization Phase Calling Tree</td>
</tr>
<tr>
<td>12-15</td>
<td>Assigner Editing Phase Calling Tree Diagram</td>
</tr>
<tr>
<td>12-16</td>
<td>Assigner DB Update Phase Calling Tree Diagram</td>
</tr>
<tr>
<td>12-17</td>
<td>Text of NEWHUL.RPROCS Execute File</td>
</tr>
</tbody>
</table>
10.0 ALIAS UTILITIES AND COMMON DATA STRUCTURES

Within the ALIAS system, a utility is a FORTRAN subroutine or BUILDER screen which performs a well-defined, limited task in such a way that it can be used by many system processors. For example, the CCAT2 subroutine is used by virtually all ALIAS FORTRAN programs when they want to concatenate two character strings.

A routine is typically not considered a utility if the function it performs is of interest to only one program, or if it requires that a complex global data structure be in place if it is to work properly. The main motivation for using utilities is that they save the programmer time; if the programmer must go to a lot of trouble getting set up to use a given "utility", he is less likely to find it a time-saver.

That said, some ALIAS utilities do require that a global data structure be in place if they are to work properly, but this structure is almost always: (1) the System Core data structure, which is always in place during an ALIAS run, or (2) a structure that can be initialized by a call to single initialization routine.

This Section will list and describe ALIAS utilities. It is meant to be a reference for programmers engaged in development work.

The Section will also present all ALIAS FORTRAN include files (which typically contain common block definitions); these are like utilities in that they are global resources often used by more than one processor.

Miscellaneous system resources, such as extra data segments, will also be covered.
The theme for Section 10, then, is coverage of shared system resources: anything used by more than one ALIAS module or by more than one part of the Core will be covered here (but not the database—see the ALIAS Data Base Reference Manual).

ALIAS FORTRAN utility routines fall into three categories:

1) Linkable general-purpose routines, where "linkable" means their object code is included in a program at PREP time.

2) Data Base management system InterFace routines (DBIF). These buffer requests for DBMS services. They are more programmer-friendly than RELATE HLI routines, and make ALIAS more convertible by isolating the calls that depend on the particular DBMS being used.

3) BUILDER-callable routines. Residing in the account Segmented Library, these routines are designed to serve BUILDER screens (via the BUILDER CALL PROCEDURE command). They are linked at or after RUN time. It is possible to call some of them from normal FORTRAN programs, but this is not usually advisable.

The next three subsections will discuss each class of FORTRAN utility in turn. Section 10.4 will discuss BUILDER utility screens. Section 10.5 presents ALIAS FORTRAN include files, and Section 10.6 discusses miscellaneous global resources.

10.1 GENERAL PURPOSE FORTRAN ROUTINES

ALIAS general-purpose FORTRAN utilities reside in the UTLO, UTLR, and RECOMP libraries. Source code is in utlo.src, recomp.src, and utlr__.src (there are several utlr source files; routines appear in them in alphabetical order according to the usual naming conventions). Object code is in utlo.obj and recomp.obj (normal object code files whose contents must be copied into an object code file about to be PREPed), and in utlr.obj. Utlr.obj is a Relocatable Library (RL), a special HP file which can be specified as a place for PREP to look for unsatisfied externals. This file should never be specified as the target for compilation; object code must be moved into this file from a regular object code file by explicit SEGMENTER commands.
Note that any routine called by a routine in an RL must also be in the RL (or else in the SL). This is why UTLO is maintained in addition to UTLR. UTLO contains utilities which are inconvenient to keep in the RL, typically because they use common blocks which change occasionally. It is tedious to have to do the recompilation and then replace the object code copy in the RL as well.

If utilities are self-contained it is more convenient to keep them in the RL because the amount of SEGMENTER work necessary to build up an object code file suitable for processing by PREP is reduced.

This Section is meant as a reference to allow programmers to quickly locate utilities of use to them, or to find more information about utilities they are having difficulty with. The utilities will be divided into about 20 groups by purpose, and a brief introduction to each group will be provided. Table 10-1 lists the groups; Table 10-2 is an annotated listing of the utilities in each group, the annotations describing the purpose of each routine. Programmers looking for a utility to perform a specific task will hopefully be able to find it quickly by consulting Tables 10-1 and 10-2.

Detailed information about any given utility will be found in Section 10.1.2, which contains the abstract/header from each utility routine in alphabetical order. These describe what arguments are required and the operation of the routine in more detail.

Programmers with the opposite problem, a specific utility which they want to know more about, can find out which group the routine belongs to and where its source code is located by referencing the alphabetical listing of Table 10-3.
Table 10-1. Types of General-Purpose FORTRAN Utility

<table>
<thead>
<tr>
<th>PURPOSE</th>
<th>DISCUSSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT MANIPULATION</td>
<td>Bit comparison, bitwise and/or</td>
</tr>
<tr>
<td>CHARACTER STRING MANIPULATION</td>
<td>String operations, e.g. concatenation, parsing, uppercasing, etc.</td>
</tr>
<tr>
<td>DATA MOVEMENT</td>
<td>Transfer data from one array to another. Also array initialization.</td>
</tr>
<tr>
<td>DATA RANGE CHECKING</td>
<td>What kind of characters in string? Number too big?</td>
</tr>
<tr>
<td>DATA TYPE CONVERSION</td>
<td>ASCII to numeric and vice versa.</td>
</tr>
<tr>
<td>DATE MANIPULATION</td>
<td>Any date-oriented operation you can imagine.</td>
</tr>
<tr>
<td>DEVICE CONTROL</td>
<td>Aids for sending hard copy output.</td>
</tr>
<tr>
<td>DIAGNOSTICS</td>
<td>1prnt setting aids.</td>
</tr>
<tr>
<td>ERROR MESSAGE OUTPUT</td>
<td>Means to tell the user things are messed up.</td>
</tr>
<tr>
<td>FILE OPEN/INPUT/OUTPUT</td>
<td>A near-F77 OPEN and some direct access helpers</td>
</tr>
<tr>
<td>FORMATTERS</td>
<td>For paged output and for bulk text output.</td>
</tr>
<tr>
<td>INITIALIZATION</td>
<td>Never hurts to call these, often helps.</td>
</tr>
<tr>
<td>LINE INPUT</td>
<td>Retrieve the next input line from anywhere.</td>
</tr>
<tr>
<td>MATHEMATICS</td>
<td>Mainly vector operations.</td>
</tr>
<tr>
<td>MEMORY MANAGER</td>
<td>An interface to extra data segments.</td>
</tr>
<tr>
<td>MISCELLANEOUS</td>
<td>Various goodies.</td>
</tr>
<tr>
<td>OPERATING SYSTEM INTERFACE</td>
<td>These make the intrinsic calls for you.</td>
</tr>
<tr>
<td>SORTING SEARCHING</td>
<td>Find a match, sort an array.</td>
</tr>
<tr>
<td>STACK DATA TYPE</td>
<td>Implementation of a stack data type.</td>
</tr>
<tr>
<td>USER INTERACTION</td>
<td>Prompting utilities.</td>
</tr>
</tbody>
</table>
### Table 10-2. General-Purpose FORTRAN Utilities By Type

<table>
<thead>
<tr>
<th>NAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BIT MANIPULATION</strong></td>
<td></td>
</tr>
<tr>
<td>IAND</td>
<td>IOR is an entry point in IAND. These do AND and OR tests on the least significant bit of a 16-bit number. They are integer functions, returning a 1 or 0.</td>
</tr>
<tr>
<td>LBIT</td>
<td>Logical function returning true if a given bit in a 16-bit word is set.</td>
</tr>
<tr>
<td><strong>CHARACTER STRING MANIPULATION</strong></td>
<td></td>
</tr>
<tr>
<td>CCAT2</td>
<td>CCAT4 is an entry point in CCAT3. These concatenate 2, 3, or 4 separate strings into a single output string. Input strings may be delimited. An input string may be specified as the output target.</td>
</tr>
<tr>
<td>CCAT3</td>
<td></td>
</tr>
<tr>
<td>CCAT4</td>
<td></td>
</tr>
<tr>
<td>CEQ</td>
<td>Logical function which strips trailing blanks only before performing an equality test on two strings. Use CIF in general.</td>
</tr>
<tr>
<td>CHNalo</td>
<td>These support string-chain data types which manages a string buffer space, allowing efficient storage of large strings. See CHNINI abstract.</td>
</tr>
<tr>
<td>CHNDEA</td>
<td></td>
</tr>
<tr>
<td>CHNFRE</td>
<td>CHNINI Note the routines can support many buffers; the buffer data structure is not built into them.</td>
</tr>
<tr>
<td>CIF</td>
<td>Logical function which strips leading and trailing blanks from two input strings and then compares them. Useful since HP automatically considers two strings of different length or with different blank-padding not equal.</td>
</tr>
<tr>
<td>DELIM</td>
<td>Useful in extracting from a delimited string.</td>
</tr>
<tr>
<td>DINDEX</td>
<td>Integer function which performs an index on a delimited string.</td>
</tr>
<tr>
<td>ELIMBL</td>
<td>ELIMinate BLanks. Left-justifies a string and returns its non-blank length.</td>
</tr>
<tr>
<td>LOWERC</td>
<td>Converts all letters in a string to lower case. An entry point in UPPERC.</td>
</tr>
<tr>
<td>LSTRNG</td>
<td>Undelimits a string and left-justifies it in the output buffer.</td>
</tr>
<tr>
<td>LTRIM</td>
<td>Integer function giving the location of the leftmost non-blank character. Returns length+1 if all blanks.</td>
</tr>
</tbody>
</table>
Table 10-2. General-Purpose FORTRAN Utilities By Type

<table>
<thead>
<tr>
<th>NAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPDCOD</td>
<td>Parser which splits a delimited string further delimited internally by commas into its constituent parts, placing them in an output array. Now used only by filopn.</td>
</tr>
<tr>
<td>MSTRNG</td>
<td>Makes the input string into a delimited string. This is an entry point in LSTRNG.</td>
</tr>
<tr>
<td>NCFLW</td>
<td>All these routines are entry points in NCFW.</td>
</tr>
<tr>
<td>NCFW</td>
<td>They convert array length in number of *2, *4, and *8 words into lengths in bytes, and vice versa.</td>
</tr>
<tr>
<td>NCSW</td>
<td></td>
</tr>
<tr>
<td>NLWFC</td>
<td></td>
</tr>
<tr>
<td>NSWFC</td>
<td></td>
</tr>
<tr>
<td>NSWFW</td>
<td></td>
</tr>
<tr>
<td>NWFC</td>
<td></td>
</tr>
<tr>
<td>NWFSW</td>
<td></td>
</tr>
<tr>
<td>RTRIM</td>
<td>Integer function returning the location of the rightmost non-blank character. Returns 0 if all blanks.</td>
</tr>
<tr>
<td>UPPERC</td>
<td>Uppercases all letters in the string.</td>
</tr>
</tbody>
</table>

**DATA MOVEMENT**

| XMIT   | These routines are just assignment loops which transfer data from one array to another. Xmit and xmit4 do reals or integer*4, xmit2 integer*2, and xmitc characters. They can reduce the volume of code in your routines by doing the work of loops with only one line. Also, if their number-of-words-to-transfer argument is negative, they expect the source to be a single word (byte) which they are to fill the target with. They can thus be very handy for array initialization. Note xmit expects to loop a *2 number of times, xmit4 a *4 number of times, so be careful specifying arguments. Also, the regular entry points can left-shift data (i.e. move second element of an array into first element, third into second, etc.) while the "B" entry points can right-shift. |
| XMITB  |                                                                                                                                          |
| XMIT2  |                                                                                                                                          |
| XMIT2B |                                                                                                                                          |
| XMIT4  |                                                                                                                                          |
| XMIT4B |                                                                                                                                          |
| XMITC  |                                                                                                                                          |
| XMITCB |                                                                                                                                          |

**DATA RANGE CHECKING**

<p>| ASCINT | Returns true if string has ASCII integer characters only.                                                                            |
| ASCPRN | Returns true if string contains only printing ASCII.                                                                                  |</p>
<table>
<thead>
<tr>
<th>NAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCREL</td>
<td>Same as ASCINT but permits &quot;.&quot;, thus allowing real numbers.</td>
</tr>
<tr>
<td>BETWN</td>
<td>Returns true if an integer lies between specified low and high values.</td>
</tr>
<tr>
<td>CRNGI</td>
<td>Generates an abort if an integer's value lies outside a specified range. The 4 version is for *4 integers.</td>
</tr>
<tr>
<td>CRNGI4</td>
<td></td>
</tr>
<tr>
<td>LETNUM</td>
<td>Returns true if a string contains letters and numbers only, i.e. no &quot;%&quot;, etc.</td>
</tr>
<tr>
<td>LETONL</td>
<td>Returns true if a string contains letters only.</td>
</tr>
</tbody>
</table>

**DATA TYPE CONVERSION**

<table>
<thead>
<tr>
<th>NAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLT</td>
<td>Converts a string to a real number. Same as the corresponding FORTRAN intrinsic, but this routine returns an error flag instead of a system abort if it can't do the job.</td>
</tr>
<tr>
<td>KFIX</td>
<td>Converts a string to an integer (*2). Returns an error flag true (rather than an abort) if string cannot be converted.</td>
</tr>
<tr>
<td>NUMASK</td>
<td>Converts an integer number into a character string AND right-justifies it into a given character string. E.g., 234 and &quot;000000&quot; come out as &quot;00234&quot;; -12 and &quot;00000&quot; as &quot;0-12&quot;. If the number is too big to fit then &quot;****...&quot; are output, conforming to the usual FORTRAN convention. Useful in output construction.</td>
</tr>
<tr>
<td>NUMSFX</td>
<td>Character function returning a labeling suffix for a number, in caps or lower case. E.g., 5 leads to output of &quot;th&quot; or &quot;TH&quot;; 1 to output of &quot;st&quot; or &quot;ST&quot;. Useful in constructing custom-formatted output.</td>
</tr>
<tr>
<td>PLURAL</td>
<td>Similar in purpose to numsfx. Character function returning &quot;s&quot; or &quot;S&quot; if number input is 1, blank otherwise.</td>
</tr>
<tr>
<td>STRN</td>
<td>Like the FORTRAN intrinsic STR, converts a number to a string. This version returns the output length, though.</td>
</tr>
</tbody>
</table>

**DATE MANIPULATION**

<table>
<thead>
<tr>
<th>NAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDTODD</td>
<td>Convert &quot;MM/DD/YYYY&quot; version of date into standard YYYY/MM/DD format.</td>
</tr>
<tr>
<td>NAME</td>
<td>PURPOSE</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CKDATE</td>
<td>Ensures a string contains a valid date representation. Can be called before to cdtodd to avoid errors. Logical function.</td>
</tr>
<tr>
<td>CKDATI</td>
<td>Checks a 3-integer version of a date to ensure it's valid (date in mm, dd, yy integers). Logical function.</td>
</tr>
<tr>
<td>CVTDAT</td>
<td>Converts a &quot;MM/DD/YYYY&quot; date into a 3-integer form, returning an error flag if it can't do it. This routine is obsolete; a combination of CKDATE, CDTODD, and DDTOID should be used instead.</td>
</tr>
<tr>
<td>DTEMK</td>
<td>Low-level date subsystem utility which converts the subsystem date representation (*4 Julian since 1601) into a 3-integer form. Meant to be called only by higher-level date utilities.</td>
</tr>
<tr>
<td>DATEPI</td>
<td>Increments a 3-integer version of a date by one day.</td>
</tr>
<tr>
<td>DATSTR</td>
<td>Returns today's date in a &quot;MM/DD/YY&quot; format.</td>
</tr>
<tr>
<td>DCLRFY</td>
<td>Takes a RELATE representation of a date (*4 word) and sets all unused bits to 0. Good insurance against date subsystem aborts, since RELATE appears to set these bits randomly, causing some of our routines to have problems.</td>
</tr>
<tr>
<td>DDATE</td>
<td>Integer*4 function returning today's date in the RELATE *4 format.</td>
</tr>
<tr>
<td>DDTODT</td>
<td>Character*10 function which converts from a ddate format to &quot;MM/DD/YYYY&quot;.</td>
</tr>
<tr>
<td>DDTOID</td>
<td>Converts from a ddate format to a 3-integer format.</td>
</tr>
<tr>
<td>DEARLY</td>
<td>Function returning true if first argument earlier than second (both arguments in ddate form).</td>
</tr>
<tr>
<td>ERLDAT</td>
<td>Returns the earliest possible ddate. This is an entry point in LATDAT.</td>
</tr>
<tr>
<td>FDDATE</td>
<td>Returns the first date in a given period in a ddate format, for a wide variety of period types.</td>
</tr>
<tr>
<td>GDATEP</td>
<td>Returns to first day of the i-th period in a ddate format, for a given fddate and period type.</td>
</tr>
<tr>
<td>GPERN</td>
<td>Given a ddate, returns the number of the period it falls in, for a given fddate and period type.</td>
</tr>
</tbody>
</table>
Table 10-2. General-Purpose FORTRAN Utilities By Type

<table>
<thead>
<tr>
<th>NAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDAYS</td>
<td>Integer function returning the number of days between two dates, specified in 3-integer format.</td>
</tr>
<tr>
<td>IDTODD</td>
<td>Converts from 3-integer format to ddate format.</td>
</tr>
<tr>
<td>JDAYS</td>
<td>Like IDAYS but returns a *4 number instead of *2.</td>
</tr>
<tr>
<td>LATDAT</td>
<td>Returns the latest possible ddate.</td>
</tr>
<tr>
<td>LMONTH</td>
<td>Integer function returning the number of days in a given month.</td>
</tr>
<tr>
<td>MRKDAY</td>
<td>Converts a 3-integer date into a ddate. Low-level utility meant to be called only by high-level date routines.</td>
</tr>
<tr>
<td>NWDATE</td>
<td>Integer*4 function returning a ddate N days later than a given ddate.</td>
</tr>
<tr>
<td>NWDATU</td>
<td>Integer*4 function returning the ddate N periods after a given ddate.</td>
</tr>
<tr>
<td>NWIDAT</td>
<td>Same as NWDATE but input and output in 3-integer format.</td>
</tr>
<tr>
<td>RDATE</td>
<td>Returns today's date in RELATE Real storage format. Obsolete, use DDATE instead.</td>
</tr>
<tr>
<td>RDFSTR</td>
<td>Converts from an RDATE format into a &quot;MM/DD/YY&quot; format. Obsolete, standardization mandates ddate formats.</td>
</tr>
</tbody>
</table>

DEVIICE CONTROL

| LPSEND | Closes a spooled output file (opened with LPSET), causing actual printing to commence.                                                 |
| LPSET  | Returns a FORTRAN i/o unit number opened on the device specified by the user in his user environment parameter menu.                   |
| SCLEAR | Clears the screen. Depends on the current terminal type setting on the user environment parameter menu being correct.                  |
| SETCCL | Reads the user environment parameter menu terminal type setting a stores the proper screen clear character sequence for use by SCLEAR. |
| SETTTY | Attempts to discover the user's terminal type by figuring out what port he's logged on through. Port number logic is hard-wired into the routine. |
Table 10-2. General-Purpose FORTRAN Utilities By Type

<table>
<thead>
<tr>
<th>NAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUG</td>
<td>Logical function which reads the LPRNTON job control word. Generally used in a statement like IF (debug) CALL setlpr.</td>
</tr>
<tr>
<td>SETLPR</td>
<td>Prompts user for changes to current lprnts settings.</td>
</tr>
<tr>
<td>SLPRNT</td>
<td>Takes an lprnts array index and a true/false argument and sets that lprnt to that value.</td>
</tr>
<tr>
<td>ERRMSG</td>
<td>Writes a delimited text string to the screen, preceded by &quot;*** &quot;.</td>
</tr>
<tr>
<td>LABORT</td>
<td>Constructs an abort message which includes &quot;AT line number&quot; plus a user message. Useful when an input file is being processed and you want to tell the user what line the problem occurred at.</td>
</tr>
<tr>
<td>LWARN</td>
<td>Like LABORT but just prints the message without aborting.</td>
</tr>
<tr>
<td>MABORT</td>
<td>Prints an error message contained in a delimited text string and calls ZABORT.</td>
</tr>
<tr>
<td>ZABORT</td>
<td>General abort routine. Prints an abort notice and STOPs execution.</td>
</tr>
</tbody>
</table>

**DIAGNOSTICS**

**ERROR MESSAGE OUTPUT**

**FILE OPEN/INPUT/OUTPUT**

| DWRITE | These are all entry points in DWRITE. They do direct access reads/writes of a specified record to a specified location on a specified unit number. The regular entries abort on an error, the "l" entries set an error flag and return. |
| DWRITE1 | |
| DREAD | |
| DREAD1 | |
| FEXIST | Logical function which returns true if the file named in the argument exists. |
| FILCLS | Closes a file opened via FOPEN. |
| FILOPN | Opens a file for FORTRAN access. Files include devices in this context. See the text on file i/o for an exposition of all the possible file specifiers—any kind of file can be created/opened. |
Table 10-2. General-Purpose FORTRAN Utilities By Type

<table>
<thead>
<tr>
<th>NAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>UREAD</td>
<td>UWRITE is an entry point in UREAD. These do unformatted sequential-access reads/writes between a given integer array and an active unit number. No error checking.</td>
</tr>
<tr>
<td>UWRITE</td>
<td></td>
</tr>
</tbody>
</table>

**FORMATTERS**

<table>
<thead>
<tr>
<th>NAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>EJECT</td>
<td>Does a page eject on the given unit number.</td>
</tr>
<tr>
<td>PGINIT</td>
<td>These routines comprise the page printing system. They are in UTLO. You can set up a header and a page size and other attributes and just send lines to this system, letting it worry when to do the page breaks. See the text on formatters for a fuller description.</td>
</tr>
<tr>
<td>PGRSET</td>
<td></td>
</tr>
<tr>
<td>PGSEND</td>
<td></td>
</tr>
<tr>
<td>PGWRIT</td>
<td></td>
</tr>
</tbody>
</table>

**PRINTERS**

<table>
<thead>
<tr>
<th>NAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRTHLP</td>
<td>This is useful for printing bulk help text or static menus. It expects a unit number which is connected to a sequential ASCII file in 80-column editor format, and a section header label. It reads through the file, finds the section by getting a match on the header, and prints the section. Much easier than putting things into format statements.</td>
</tr>
<tr>
<td>TRECOL</td>
<td>Prints a list (array) of character elements in three columns onto a specified unit.</td>
</tr>
</tbody>
</table>

**INITIALIZATION**

<table>
<thead>
<tr>
<th>NAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFINIT</td>
<td>Initializes the Core command system's stored commands subsystem. The routine MUST be called before the READLN utility can be used.</td>
</tr>
<tr>
<td>GETGRP</td>
<td>Determines whether the user is running the development or production version of ALIAS, and sets the variable that holds the group name where menu system files and relations will be.</td>
</tr>
<tr>
<td>INIPRC</td>
<td>Does general initialization for a FORTRAN module being executed as a son process by the Core, including swap-in of the Core common blocks generally of interest (e.g., /uzrprv/, /scenar/, and /pvalue/).</td>
</tr>
<tr>
<td>INIOOC</td>
<td>Together these routines will initialize i/o for the utilities and for a FORTRAN module in general. Mainly they set the integer variables which hold the standard input and output unit numbers. Mabort, zabort, etc. will not work if these are not called.</td>
</tr>
<tr>
<td>INITIO</td>
<td></td>
</tr>
</tbody>
</table>
Table 10-2. General-Purpose FORTRAN Utilities By Type

<table>
<thead>
<tr>
<th>NAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTYINI</td>
<td>A terminal-type detection utility which works by querying the user. Sometimes useful during module debugging when you don’t want to be hooked up to the Core; the screen clear sequence is placed in the /tty/ block.</td>
</tr>
<tr>
<td>RDLN</td>
<td>Reads a 72-character line from a given unit without uppercasing.</td>
</tr>
<tr>
<td>RDLNC</td>
<td>Reads 72-character lines without uppercasing and keeps track of the number read in the /readc/ block.</td>
</tr>
<tr>
<td>RDLNCU</td>
<td>Like RDLNC but uppercases as well.</td>
</tr>
<tr>
<td>READLN</td>
<td>The main System Core line-read routine. This routine knows about the stored commands subsystem; it will automatically close a command file and reset to normal terminal i/o operation (via a call to stopcf) when the end of the command file is reached. Always use this routine for obtaining user input in the Core, and for any module linked into the Core that you want to be serviced by the stored commands subsystem. Note that READLN uppercases all input.</td>
</tr>
<tr>
<td>IXSUM</td>
<td>Sums up all elements of a 1-dimensional vector (array). Integer*2 function.</td>
</tr>
<tr>
<td>RANF</td>
<td>Random number generator. Provides numbers along up to RANGET 10 sequences; specify sequence when calling RANF.</td>
</tr>
<tr>
<td>RANSET</td>
<td>Initialization of seed for a specific sequence done by call to RANSET. RANSTI initializes all sequences.</td>
</tr>
<tr>
<td>RANTRP</td>
<td>RANGET returns the current status of all sequences for saving.</td>
</tr>
<tr>
<td>VSUMNI</td>
<td>Vector sum for two 1-dimensional vectors.</td>
</tr>
<tr>
<td>VSUBNI</td>
<td>Vector difference for two 1-dimensional vectors. This is an entry point in VSUMNI.</td>
</tr>
<tr>
<td>FINMEM</td>
<td>The memory manager supports use of extra data segments for extended global storage. Inimem initializes for a</td>
</tr>
<tr>
<td>GETMEM</td>
<td></td>
</tr>
</tbody>
</table>

10-12
Table 10-2. General-Purpose FORTRAN Utilities By Type

<table>
<thead>
<tr>
<th>NAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIMEM</td>
<td>given segment, putmem and getmem allow transfer of arrays between the segment and directly addressable memory, and finmem releases the segment. These utilities are very useful if your program requires more than 64K bytes of data memory. You can page the big arrays to one or more 64K segments.</td>
</tr>
<tr>
<td>PUTMEM</td>
<td></td>
</tr>
<tr>
<td>LISTON</td>
<td>Given a menu name and a scenario key field value, this routine returns a list of candidates on the given list menu and their on/off statuses.</td>
</tr>
<tr>
<td>MODCOR</td>
<td>An alternative modulo function.</td>
</tr>
<tr>
<td>STOFCF</td>
<td>Really part of the Core command system's stored commands subsystem, this must be in the RL because it is called by READLN. The routine just resets i/o and command system units and flags on end-of-command-file.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>OPERATING SYSTEM INTERFACE</td>
<td></td>
</tr>
<tr>
<td>CPUTILM</td>
<td>Current system cpu clock time in milliseconds. Two calls (this is a real function) give an interval.</td>
</tr>
<tr>
<td>MONCOM</td>
<td>Executes a monitor command (i.e. an MPE command).</td>
</tr>
<tr>
<td>USRINF</td>
<td>Returns id information about the user, including name and log-on group. Has 3 entry points: USRNAME, USRGRP, USRACCT.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SORTING SEARCHING</td>
<td></td>
</tr>
<tr>
<td>CHASH</td>
<td>Does a hash-type sort on a character array. Does not actually sort the array, just returns an array of integers that give the sorted order of the character array elements.</td>
</tr>
<tr>
<td>CHASHV</td>
<td>Used by CHASH.</td>
</tr>
<tr>
<td>JHASH</td>
<td>Same as CHASH, but operates on a *4 integer array.</td>
</tr>
<tr>
<td>MATCH2</td>
<td>Integer function returning the location in an array of integers of a given target integer.</td>
</tr>
<tr>
<td>MATCHC</td>
<td>Same as MATCH2, looks for a match for a character string.</td>
</tr>
</tbody>
</table>
Table 10-2. General-Purpose FORTRAN Utilities By Type

<table>
<thead>
<tr>
<th>NAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTCHOC</td>
<td>Same as MATCHC but assumes that character array it is to search is sorted. More efficient if this is true.</td>
</tr>
<tr>
<td>QSORTC</td>
<td>Returns a character array sorted. Uses heap sort method.</td>
</tr>
</tbody>
</table>

**STACK DATA TYPE**

<table>
<thead>
<tr>
<th>NAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSINIT</td>
<td>See text discussion of the stack data type for purpose and organization of these routines. CSINIT initializes the stack system.</td>
</tr>
<tr>
<td>CSPOP</td>
<td>Pops an item off the stack</td>
</tr>
<tr>
<td>CSPOPR</td>
<td>Pops and returns an item from the stack.</td>
</tr>
<tr>
<td>CSPSH</td>
<td>Pushes an item onto the stack</td>
</tr>
<tr>
<td>CSPSH2</td>
<td>Pushes two items onto the stack</td>
</tr>
<tr>
<td>CSPSHL</td>
<td>Pushes many items onto the stack</td>
</tr>
<tr>
<td>CSRD</td>
<td>Reads the top item on the stack and leaves it there.</td>
</tr>
<tr>
<td>CSRD2</td>
<td>Reads two items at top of stack and leaves them there.</td>
</tr>
<tr>
<td>CSRDL</td>
<td>Reads many items and leaves them there.</td>
</tr>
</tbody>
</table>

**USER INTERACTION**

<table>
<thead>
<tr>
<th>NAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNTINU</td>
<td>Stops execution and prompts the user to hit return to continue. Useful when a message has been put up and the user needs to have the leisure to read it before more output is done.</td>
</tr>
<tr>
<td>DPAUSE</td>
<td>Does an N-second process pause.</td>
</tr>
<tr>
<td>QUERY</td>
<td>Logical function prompting user for yes-or-no answer to a given (delimited) text query. Uses YESNO.</td>
</tr>
<tr>
<td>YESNO</td>
<td>Logical function which forces the user to answer Y, YES, N, or NO. Calls READLN for input, so cfinit and initio and inioc must be called before this is used. Does not print prompt.</td>
</tr>
</tbody>
</table>
Table 10-3. General-Purpose FORTRAN Utilities By Name

<table>
<thead>
<tr>
<th>NAME</th>
<th>SOURCE FILE</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCINT</td>
<td>UTLRA</td>
<td>DATA RANGE CHECKING</td>
</tr>
<tr>
<td>ASCPRN</td>
<td>UTLRA</td>
<td>DATA RANGE CHECKING</td>
</tr>
<tr>
<td>ASCREL</td>
<td>UTLRA</td>
<td>DATA RANGE CHECKING</td>
</tr>
<tr>
<td>BETWN</td>
<td>UTLRA</td>
<td>DATA RANGE CHECKING</td>
</tr>
<tr>
<td>CCAT2</td>
<td>UTLRA</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>CCAT3</td>
<td>UTLRA</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>CCAT4 (e)</td>
<td>UTLRA</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>CDTODD</td>
<td>UTLRA</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>CEQ</td>
<td>UTLRA</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>CFINIT</td>
<td>UTLRA</td>
<td>INITIALIZATION</td>
</tr>
<tr>
<td>CHASH</td>
<td>UTLRA</td>
<td>SORTING SEARCHING</td>
</tr>
<tr>
<td>CHASHV</td>
<td>UTLRA</td>
<td>SORTING SEARCHING</td>
</tr>
<tr>
<td>CHNALO</td>
<td>UTLRCHN</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>CHNDEA</td>
<td>UTLRCHN</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>CHNFRE</td>
<td>UTLRCHN</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>CHNINI</td>
<td>UTLRCHN</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>CIF</td>
<td>UTLRCI</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>CKDATE</td>
<td>UTLRCI</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>CKDATI</td>
<td>UTLRCI</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>CNTRNU</td>
<td>UTLRCI</td>
<td>USER INTERACTION</td>
</tr>
<tr>
<td>CPUTIM</td>
<td>UTLRCI</td>
<td>OPERATING SYSTEM INTERFACE</td>
</tr>
<tr>
<td>CRNGI</td>
<td>UTLRCI</td>
<td>DATA RANGE CHECKING</td>
</tr>
<tr>
<td>CRNGI4</td>
<td>UTLRCI</td>
<td>DATA RANGE CHECKING</td>
</tr>
<tr>
<td>CSINIT</td>
<td>UTLRCS</td>
<td>STACK DATA TYPE</td>
</tr>
<tr>
<td>CSPOP</td>
<td>UTLRCS</td>
<td>STACK DATA TYPE</td>
</tr>
<tr>
<td>CSPOPR</td>
<td>UTLRCS</td>
<td>STACK DATA TYPE</td>
</tr>
<tr>
<td>CSPSH</td>
<td>UTLRCS</td>
<td>STACK DATA TYPE</td>
</tr>
<tr>
<td>CSPSH2</td>
<td>UTLRCS</td>
<td>STACK DATA TYPE</td>
</tr>
<tr>
<td>CSPSHL</td>
<td>UTLRCS</td>
<td>STACK DATA TYPE</td>
</tr>
<tr>
<td>CSRD</td>
<td>UTLRCS</td>
<td>STACK DATA TYPE</td>
</tr>
<tr>
<td>CSRD2</td>
<td>UTLRCS</td>
<td>STACK DATA TYPE</td>
</tr>
<tr>
<td>CSRDLD</td>
<td>UTLRCS</td>
<td>STACK DATA TYPE</td>
</tr>
<tr>
<td>CVTDAT</td>
<td>UTLRCV</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>DATEMK</td>
<td>UTLRCV</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>DATEP1</td>
<td>UTLRCV</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>DATSTR</td>
<td>UTLRCV</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>DCLRFY</td>
<td>UTLRCV</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>DDATE</td>
<td>UTLRCV</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>DDTOCD</td>
<td>UTLRCV</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>DDTOID</td>
<td>UTLRCV</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>DEARLY</td>
<td>UTLRCV</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>DEBUG</td>
<td>UTLRCV</td>
<td>DIAGNOSTICS</td>
</tr>
<tr>
<td>DELIM</td>
<td>UTLRCV</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>DINDEX</td>
<td>UTLRDI</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>DPAUSE</td>
<td>UTLRDI</td>
<td>USER INTERACTION</td>
</tr>
<tr>
<td>DREAD (e)</td>
<td>UTLRDI</td>
<td>FILE OPEN/INPUT/OUTPUT</td>
</tr>
<tr>
<td>DREAD1 (e)</td>
<td>UTLRDI</td>
<td>FILE OPEN/INPUT/OUTPUT</td>
</tr>
<tr>
<td>DWRITE1 (e)</td>
<td>UTLRDI</td>
<td>FILE OPEN/INPUT/OUTPUT</td>
</tr>
<tr>
<td>DWRITE</td>
<td>UTLRDI</td>
<td>FILE OPEN/INPUT/OUTPUT</td>
</tr>
<tr>
<td>EJECT</td>
<td>UTLRDI</td>
<td>FORMATTERS</td>
</tr>
</tbody>
</table>

10-15
<table>
<thead>
<tr>
<th>NAME</th>
<th>SOURCE FILE</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELIMBL</td>
<td>UTLRDI</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>ERLDAT</td>
<td>UTLRDI</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>ERRMSG</td>
<td>UTLRDI</td>
<td>ERROR MESSAGE OUTPUT</td>
</tr>
<tr>
<td>FDATE</td>
<td>UTLRF</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>FEXIST</td>
<td>UTLRF</td>
<td>FILE OPEN/INPUT/OUTPUT</td>
</tr>
<tr>
<td>FILECLS</td>
<td>UTLRF</td>
<td>FILE OPEN/INPUT/OUTPUT</td>
</tr>
<tr>
<td>FILOPN</td>
<td>UTLRF</td>
<td>FILE OPEN/INPUT/OUTPUT</td>
</tr>
<tr>
<td>FINMEM</td>
<td>UTLRFIN</td>
<td>MEMORY MANAGER</td>
</tr>
<tr>
<td>FLT</td>
<td>UTLRFIN</td>
<td>DATA TYPE CONVERSION</td>
</tr>
<tr>
<td>GDATEP</td>
<td>UTLRFIN</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>GETGRP</td>
<td>UTLRFIN</td>
<td>INITIALIZATION</td>
</tr>
<tr>
<td>GETMEM</td>
<td>UTLRFIN</td>
<td>MEMORY MANAGER</td>
</tr>
<tr>
<td>GPERN</td>
<td>UTLRFIN</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>IAND</td>
<td>UTLRFIN</td>
<td>BIT MANIPULATION</td>
</tr>
<tr>
<td>IDAYS</td>
<td>UTLRFIN</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>IDTODD</td>
<td>UTLRFIN</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>INIOC</td>
<td>UTLRFIN</td>
<td>INITIALIZATION</td>
</tr>
<tr>
<td>INIMEM</td>
<td>UTLRFIN</td>
<td>MEMORY MANAGER</td>
</tr>
<tr>
<td>INIPRC</td>
<td>UTLRFIN</td>
<td>INITIALIZATION</td>
</tr>
<tr>
<td>INITIO</td>
<td>UTLRFIN</td>
<td>INITIALIZATION</td>
</tr>
<tr>
<td>IOR</td>
<td>UTLRFIN</td>
<td>BIT MANIPULATION</td>
</tr>
<tr>
<td>IXSUM</td>
<td>UTLRFIN</td>
<td>MATHEMATICS</td>
</tr>
<tr>
<td>JDAYS</td>
<td>UTLRFIN</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>JHASH</td>
<td>UTLRFIN</td>
<td>SORTING SEARCHING</td>
</tr>
<tr>
<td>KFIX</td>
<td>UTLRK</td>
<td>DATA TYPE CONVERSION</td>
</tr>
<tr>
<td>LABORT</td>
<td>UTLRK</td>
<td>ERROR MESSAGE OUTPUT</td>
</tr>
<tr>
<td>LATDAT</td>
<td>UTLRK</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>LBIT</td>
<td>UTLRK</td>
<td>BIT MANIPULATION</td>
</tr>
<tr>
<td>LETNUM</td>
<td>UTLRK</td>
<td>DATA RANGE CHECKING</td>
</tr>
<tr>
<td>LETONL</td>
<td>UTLRK</td>
<td>DATA RANGE CHECKING</td>
</tr>
<tr>
<td>LISTON</td>
<td>UTLR</td>
<td>MISCELLANEOUS</td>
</tr>
<tr>
<td>LMONTH</td>
<td>UTLRK</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>LOWERC(e)</td>
<td>UTLRK</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>LPSEN supp</td>
<td>RECOMP</td>
<td>DEVICE CONTROL</td>
</tr>
<tr>
<td>LPSET</td>
<td>RECOMP</td>
<td>DEVICE CONTROL</td>
</tr>
<tr>
<td>LSTRING</td>
<td>UTLRK</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>LTRIM</td>
<td>UTLRK</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>LWARN</td>
<td>UTLRK</td>
<td>ERROR MESSAGE OUTPUT</td>
</tr>
<tr>
<td>MABORT</td>
<td>UTLRM</td>
<td>ERROR MESSAGE OUTPUT</td>
</tr>
<tr>
<td>MATCH2</td>
<td>UTLRM</td>
<td>SORTING SEARCHING</td>
</tr>
<tr>
<td>MATCHC</td>
<td>UTLRM</td>
<td>SORTING SEARCHING</td>
</tr>
<tr>
<td>MODCOR</td>
<td>UTLRM</td>
<td>MISCELLANEOUS</td>
</tr>
<tr>
<td>MONCOM</td>
<td>UTLRM</td>
<td>OPERATING SYSTEM INTERFACE</td>
</tr>
<tr>
<td>MPDCOD</td>
<td>UTLRM</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>MRKDAY</td>
<td>UTLRM</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>MSTRNG(e)</td>
<td>UTLRM</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>MTCOC</td>
<td>UTLRM</td>
<td>SORTING SEARCHING</td>
</tr>
<tr>
<td>NCFNW</td>
<td>UTLRM</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>NCSW(e)</td>
<td>UTLRM</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>NAME</td>
<td>SOURCE</td>
<td>TYPE</td>
</tr>
<tr>
<td>----------------</td>
<td>----------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>NLWFC(e)</td>
<td>UTLRM</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>NSWFC(e)</td>
<td>UTLRM</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>NSWFW(e)</td>
<td>UTLRM</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>NUMASK</td>
<td>UTLRM</td>
<td>DATA TYPE CONVERSION</td>
</tr>
<tr>
<td>NUMSFX</td>
<td>UTLRM</td>
<td>DATA TYPE CONVERSION</td>
</tr>
<tr>
<td>NWDATE</td>
<td>UTLRNW</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>NWDATU</td>
<td>UTLRNW</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>NWFC(e)</td>
<td>UTLRNW</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>NSWFW(e)</td>
<td>UTLRNW</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>NWIDAT</td>
<td>UTLRNW</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>PGINIT</td>
<td>UTLQ</td>
<td>FORMATTERS</td>
</tr>
<tr>
<td>PGRSET</td>
<td>UTLQ</td>
<td>FORMATTERS</td>
</tr>
<tr>
<td>PSGEND</td>
<td>UTLQ</td>
<td>FORMATTERS</td>
</tr>
<tr>
<td>PGWRIT</td>
<td>UTLQ</td>
<td>FORMATTERS</td>
</tr>
<tr>
<td>PLURAL</td>
<td>UTLRNW</td>
<td>DATA TYPE CONVERSION</td>
</tr>
<tr>
<td>PRTHLP</td>
<td>UTLRNW</td>
<td>FORMATTERS</td>
</tr>
<tr>
<td>PUTMEM</td>
<td>UTLRNW</td>
<td>MEMORY MANAGER</td>
</tr>
<tr>
<td>QSORTC</td>
<td>UTLRQ</td>
<td>SORTING SEARCHING</td>
</tr>
<tr>
<td>QUERY</td>
<td>UTLRQ</td>
<td>USER INTERACTION</td>
</tr>
<tr>
<td>RANF</td>
<td>UTLRQ</td>
<td>MATHEMATICS</td>
</tr>
<tr>
<td>RANGET(e)</td>
<td>UTLRQ</td>
<td>MATHEMATICS</td>
</tr>
<tr>
<td>RANSET(e)</td>
<td>UTLRQ</td>
<td>MATHEMATICS</td>
</tr>
<tr>
<td>RANST1(e)</td>
<td>UTLRQ</td>
<td>MATHEMATICS</td>
</tr>
<tr>
<td>RANTRP</td>
<td>UTLRQ</td>
<td>MATHEMATICS</td>
</tr>
<tr>
<td>RDATE</td>
<td>UTLRQ</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>RDFSTR</td>
<td>UTLRQ</td>
<td>DATE MANIPULATION</td>
</tr>
<tr>
<td>RDLN</td>
<td>UTLQ</td>
<td>LINE INPUT</td>
</tr>
<tr>
<td>RDLCN</td>
<td>UTLQ</td>
<td>LINE INPUT</td>
</tr>
<tr>
<td>RDLCNU</td>
<td>UTLQ</td>
<td>LINE INPUT</td>
</tr>
<tr>
<td>READLN</td>
<td>UTLRQ</td>
<td>LINE INPUT</td>
</tr>
<tr>
<td>RTRIM</td>
<td>UTLRQ</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>SCLEAR</td>
<td>RECOMP</td>
<td>DEVICE CONTROL</td>
</tr>
<tr>
<td>SETCCCL</td>
<td>RECOMP</td>
<td>DEVICE CONTROL</td>
</tr>
<tr>
<td>SETLPRT</td>
<td>UTLRQ</td>
<td>DIAGNOSTICS</td>
</tr>
<tr>
<td>SETTTY</td>
<td>RECOMP</td>
<td>DEVICE CONTROL</td>
</tr>
<tr>
<td>SLPRNT</td>
<td>UTLRQ</td>
<td>DIAGNOSTICS</td>
</tr>
<tr>
<td>STOPCF</td>
<td>UTLRQ</td>
<td>MISCELLANEOUS</td>
</tr>
<tr>
<td>STRN</td>
<td>UTLRQ</td>
<td>DATA TYPE CONVERSION</td>
</tr>
<tr>
<td>TRECOL</td>
<td>UTLRQ</td>
<td>FORMATTERS</td>
</tr>
<tr>
<td>TTYINI</td>
<td>UTLRQ</td>
<td>INITIALIZATION</td>
</tr>
<tr>
<td>UPPERC</td>
<td>UTLRQ</td>
<td>CHARACTER STRING MANIPULATION</td>
</tr>
<tr>
<td>UREAD</td>
<td>UTLRQ</td>
<td>FILE OPEN/INPUT/OUTPUT</td>
</tr>
<tr>
<td>USRACL</td>
<td>UTLRQ</td>
<td>OPERATING SYSTEM INTERFACE</td>
</tr>
<tr>
<td>USRGRP</td>
<td>UTLRQ</td>
<td>OPERATING SYSTEM INTERFACE</td>
</tr>
<tr>
<td>USRINF</td>
<td>UTLRQ</td>
<td>OPERATING SYSTEM INTERFACE</td>
</tr>
<tr>
<td>USRNAME</td>
<td>UTLRQ</td>
<td>OPERATING SYSTEM INTERFACE</td>
</tr>
<tr>
<td>UWRITE(e)</td>
<td>UTLRQ</td>
<td>FILE OPEN/INPUT/OUTPUT</td>
</tr>
<tr>
<td>VSUBN1</td>
<td>UTLRQ</td>
<td>MATHEMATICS</td>
</tr>
<tr>
<td>VSUMNI</td>
<td>UTLRQ</td>
<td>MATHEMATICS</td>
</tr>
<tr>
<td>XMIT/B</td>
<td>UTLRX</td>
<td>DATA MOVEMENT</td>
</tr>
</tbody>
</table>
Table 10-3. General-Purpose FORTRAN Utilities By Name

<table>
<thead>
<tr>
<th>NAME</th>
<th>SOURCE FILE</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMIT2/B</td>
<td>UTLRX</td>
<td>DATA MOVEMENT</td>
</tr>
<tr>
<td>XMIT4/B</td>
<td>UTLRX</td>
<td>DATA MOVEMENT</td>
</tr>
<tr>
<td>XMITC</td>
<td>UTLRX</td>
<td>DATA MOVEMENT</td>
</tr>
<tr>
<td>YESNO</td>
<td>UTLRX</td>
<td>USER INTERACTION</td>
</tr>
<tr>
<td>ZABORT</td>
<td>UTLRX</td>
<td>ERROR MESSAGE OUTPUT</td>
</tr>
</tbody>
</table>
10.1.1 Discussion By Type of Utility

This section will discuss some (not all) of the utility types listed in Table 10-1. Most are self-explanatory, but some exist because of particular features of the HP or of RELATE; these features need to be elucidated.

10.1.1.1 Character String Utilities

Many of the existing character string oriented utilities would be unnecessary if the HP 3000 had an ANSI 1977 standard FORTRAN compiler. Concatenation (CCAT routines) would be done using FORTRAN syntax, and the string chain data type (CHN routines) would be less necessary because strings would not be limited to 255 character lengths. The various delimit-undelimit utilities wouldn't be needed.

10.1.1.2 Date Manipulation Utilities

Most ALIAS modules must work with dates. The extensive array of ALIAS date utilities makes this a straightforward rather than maddening task by allowing the programmer to convert between many date formats, to compare dates and calculate intervals, and to work in terms of periods (months, weeks, years, etc.) as well as days.

A date can be stored and/or manipulated in four formats:

1) 10-character ASCII (MM/DD/YYYY), convenient for user i/o.

2) 3-integer format, i.e. 3 two-byte integers each holding one of month, day, year.

3) RELATE double-integer word (D) date storage format. RELATE stores a date in a double integer by reserving ranges of bits within the 32-bit word for the month, day, and year quantities.

4) "Ddate" format. This is the date utilities' preferred format. Stored in a double integer word, dates are expressed in Julian form with a basis date of 31 Dec 1600.
Although the large variety of formats may seem unwieldy, the first two are very convenient, and the last two are necessary.

10.1.1.3 Device Control

ALIAS programs typically want to work with two kinds of device: the user's terminal, and spooled printers. The only action supported with respect to terminals is a screen-clear. Programs which require sophisticated screen management should be written in BUILDER if possible.

The screen-clear utilities consult the contents of the TTYTYP parameter on the User Environment Parameters menu of the Command System every time a clear is requested. They contain hard-wired logic which converts the code names found there into screen-clear character sequences. Should additional terminal types come into use on ALIAS, the code of these routines will need to be changed.

10.1.1.4 File Open/Input/Output

One of the major weaknesses of HP FORTRAN is its lack of an OPEN statement. Dynamic opening of a file involves some very messy calls to MPE Intrinsic routines. To make direct use of these avoidable, the filopn/filcls utilities were written. Filopn takes three arguments: the unit number the file should be opened on, a logical flag which is returned .true. if the operation succeeded, and a delimited string of directives separated by commas. Syntax of a typical filopn call might be:

CALL filopn(unit,ok,"+name=myfil.grp,new,ascii,write,fixlreces+")

which requests creation of a new ascii fixed record length file in the .grp group named myfil.

Table 10-4 gives all directives that filopn accepts. Filcls requires similar arguments, but has directive options of
Table 10-4. FILOPN Directives

<table>
<thead>
<tr>
<th>DIRECTIVE</th>
<th>DEFAULT VALUE</th>
<th>EFFECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME=filnam</td>
<td>FTN##</td>
<td>Specifies name of file to open. If name does not include a group, log-on group is assumed.</td>
</tr>
<tr>
<td>DEVICE=##</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEW</td>
<td>NEW</td>
<td>Use NEW to create a new file, OLD to open an existing permanent file, OLDTEMP for an existing temporary file.</td>
</tr>
<tr>
<td>OLD</td>
<td>OLD</td>
<td></td>
</tr>
<tr>
<td>OLDTEMP</td>
<td>OLDTEMP</td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td>READ</td>
<td>Specifies the types of operations you will be allowed to perform on the file.</td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APPEND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPDATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>READWRITE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXCLUSIVE</td>
<td>SHARED if READ LOCKABLE applies only if SHARED LOCKABLE applies only if SHARE if READ LOCKABLE applies only if SHARED LOCKABLE applies only if READ LOCKABLE applies only if</td>
<td></td>
</tr>
<tr>
<td>SHARED</td>
<td>else EXCLUSIVE</td>
<td></td>
</tr>
<tr>
<td>LOCKABLE</td>
<td>SHARED not recommended when you are going to write to the file. EXCLUSIVE will cause an error message if someone else already has it open, but no abort. Detect open failure by flag returned by filopn.</td>
<td></td>
</tr>
<tr>
<td>ASCII</td>
<td>BINARY</td>
<td>You must choose the form of the file when it is created. Binary files may not be edited. Ignored if file is OLD.</td>
</tr>
<tr>
<td>BINARY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KSAM</td>
<td>SEQUENTIAL</td>
<td>No one knows how to use KSAM.</td>
</tr>
<tr>
<td>SEQUENTIAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIXLRECS</td>
<td>VARLRECS</td>
<td>Variable-length records save space, especially when the file is ASCII, but the editors work best with fixed-length records.</td>
</tr>
<tr>
<td>VARLRECS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECLEN=#####</td>
<td>133</td>
<td>Record length in bytes.</td>
</tr>
<tr>
<td>MAXRECS=#####</td>
<td>1023</td>
<td>Maximum number records in file.</td>
</tr>
<tr>
<td>RECPBLK=#####</td>
<td>system decides</td>
<td>Blocking factor.</td>
</tr>
<tr>
<td>LABELED</td>
<td></td>
<td>Indicates a labeled tape. Filopn not so far used with tape.</td>
</tr>
</tbody>
</table>

10-21
SAVE, DELETE, and TEMPORARY only. Note that every filopn call directive list except those including DEVICE= must specify a file name and an access type (e.g. READ or WRITE).

Note that a file opened as "NEW" will not in fact be created in the permanent file domain until successfully closed by a filcls call with the "SAVE" option.

The utilities which perform direct-access file i/o are convenient because they automatically perform error checking during the read/write, producing either a "nice" abort or else returning a status flag to the caller.

10.1.1.5 **Formatters**

The output formatting utilities can be extremely useful. The prthlp routine is used by the scenario system for display of menus, and by the assigner for display of all help text. It is a good means for shoving large volumes of text at the user.

The PG routines form a subsystem that can make production of reports much easier from FORTRAN. A common problem in report generation is the necessity to count output lines so that page ejects can be given at appropriate points, and so that headers can be written at the top of each page. Also, it is often desirable to prompt the user before each page eject when output is going to the screen. The PG routines handle the details of all of this for the programmer. Output can be generated and sent to the subsystem line by line with no worries.

To use the subsystem, call pginit and pgrset. Pgrset can be called at any time to begin output of a new report. Pgrset wants such things as the unit number of the output file or device, its record length and page length, the formfeed character, and the mode the PG subsystem should operate in.
The subsystem works by storing each line sent to it in a buffer until it has a full page. Lines are sent by calls to the PGSEND routine, whose arguments include a page header of as many lines as the developer chooses. Its action when the page is full depends on the operating mode. There are four mode choices, specified by number:

1) PG routines send output to unit when buffer has a full page, user is prompted before output sent. This is most appropriate for screen output.

2) Same as 1 but output is continuous, user is not prompted. Most appropriate for line printer output.

3) Same as 2 but the header (specified as argument to PGSEND) is printed only at the top of the first page, not at the top of every page as in 1 and 2.

4) PGSEND does not send output to the unit automatically. Instead, it returns a flag when there is a full page in the buffer, leaving it to the user to call PGWRIT to print the buffer contents.

The variety of operating modes makes the subsystem configurable to most situations.

10.1.1.6 Initialization

Programs which intend to use any of the utilities should always call, in this order, the INITIO, INIIOC, and CFINIT initialization routines. These set certain key global i/o variables, mostly unit numbers, which are relied on by some utilities.

The convenience of using iniprc to initialize FORTRAN modules executed as son processes was discussed in Sections 8 and 9.

10.1.1.7 Line Input

The READLN routine MUST be used to retrieve ALL terminal input in System Core routines which are to be serviced by the stored commands subsystem. This ensures that i/o redirection takes place properly. The other line-read routines can be useful when processing a text input file, since some of them will keep a
running count of the number of lines read for use in error or progress messages.

10.1.1.8 Stack Data Type

The stack data type was discussed in Section 3.1.6 and in Section 8. The stack utility routines are currently capable of implementing only one stack per process; the stack is reserved for use by the Command System in the System Core process. However, extension of the utilities to manage multiple stacks would be straightforward.
10.1.2 General-Purpose FORTRAN Utility Abstracts

GENERAL-PURPOSE UTILITIES

C  ASCINT  ****************************************************
$control segment=dmaint
   LOGICAL FUNCTION asciit(string,len)
      integer len
      character*(len) string
C  *** ABSTRACT ***
C PURPOSE Checks a string to be sure it contains only numbers.
C AUDIT HISTORY
C  MScarey  17-mar-83 AUTHOR
C FORMAL PARAMETERS
Cin   len   length of input string
Cin   string string to be checked
C COMMON BLOCKS
C  none
C CALLER various
C METHOD
C Makes sure each byte is within proper octal range. Allows a
C trailing blank for strings of odd length, ad a leading "-".
C LOCAL VARIABLES
C  buffer word-aligned version of input string
C
**GENERAL-PURPOSE UTILITIES**

```c
C ASCPRN

$control segment=dmaint

LOGICAL FUNCTION ascprn(string, len)
integer len
character*(len) string

*** ABSTRACT ***

**PURPOSE** Checks a string to be sure it contains only printing chars

**AUDIT HISTORY**
MSCarey 17-mar-83 AUTHOR

**FORMAL PARAMETERS**
Cin len length of input string
Cin string string to be checked

**COMMON BLOCKS**
C none

**CALLER** various

**METHOD**
C Makes sure that each byte is within the proper octal range.

**LOCAL VARIABLES**
C buffer word-aligned version of string
```
C      ASCREL  ***************************************************************************

#define segment=dmaint

LOGICAL FUNCTION ascrel(string,len)
  integer len
  character*(len) string

C*      *** ABSTRACT ***
C*PURPOSE Checks a string to be sure it contains only numbers or .
CAUDIT HISTORY
C    MSCarey     17-mar-83 AUTHOR
C*FORMAL PARAMETERS
Cin    len     length of input string
Cin    string  string to be checked
C*COMMON BLOCKS
C    none
C*CALLER various
C*METHOD
C    Makes sure each byte is within allowed octal range, or is a "."
C    or a "-".
C*LOCAL VARIABLES
C    buffer   word-aligned version of string
C**
C BETWN

LOGICAL FUNCTION betwn(i,low,high)

*** FORMAL PARAMETER DECLARATIONS ***

integer i, low, high

*** ABSTRACT ***

C\*PURPOSE betwn = low <= i <= high

C\*AUDIT HISTORY

C Densmore 04-Feb-83 AUTHOR

C\*TYPE misc. utility

C\*FORMAL PARAMETERS

Cin i, low, high  *2 integers for function
SUBROUTINE ccat2(s1,len1,s2,len2,sr,lenr,nlenr)

INTEGER len1,len2,lenr,nlenr
CHARACTER sl(len1), s2(len2), sr(nlenr)

** ABSTRACT **

** PURPOSE ** concatenates s1 to s2 and returns result in sr

** AUDIT HISTORY **

C Densmore 15-Dec-82 AUTHOR

** TYPE ** string manipulation utility

** FORMAL PARAMETERS **

Cin s1 first string, may be DTS
Cin len1 length in characters of s1
Cin s2 second string, may be DTS
Cin len2 length of s2
Cout sr returned string: may have same address as s1 or s2
Cout lenr length of sr
Cin nlenr maximum length allowable for sr

** METHOD **

DTS refers to a Delimited Text String, in which the length
is determined by delimiters, one before and one following the
intended string. The delimiter character is the first in the string
which is nonblank: 'abcdef' -> 'abcdef'. A string is assumed
to be DTS if and only if the length associated with it is ZERO.

** LOCAL VARIABLES **

C b? beginning position of (possibly delimited) string
C e? ending position of (possibly delimited) string
C l? length of (possibly delimited) string
C  CCAT3**********************************
$CONTROL check=2
SUBROUTINE ccat3(s1,len1,s2,len2,s3,len3,sr,lenr,mlenr)
C*
  *** FORMAL PARAMETER DECLARATIONS ***
  INTEGER len1,len2,len3,len4,lenr,mlenr
  CHARACTER*255 s1,s2,s3,s4,sr
C*
  *** ABSTRACT ***
C*PURPOSE performs sr := s1 :: s2 :: s3
C*AUDIT HISTORY
C  Denne more  15-Dec-82 AUTHOR
C*TYPE string manipulation utility
C*FORMAL PARAMETERS
Cin  sM  strings
Cin  lenM  character lengths
Cout  sr  returned string
Cout  lenr  its length
Cin  mlenr  maximum length allowable for sr
C**
C CONTROL check=3
INTEGER*4 FUNCTION cdtodd(datstr)
         character*10 datstr
C* *** ABSTRACT ***
C PURPOSE Character Date TO Relate Date. Converts a character
C string of the form MM/DD/YYYY to a I*4 stored as
C in RELATE format. See the data type text in TDDATE.INCL
C AUDIT HISTORY
C   MSCarey    16-mar-83 AUTHOR
C   Gensmore   26-Apr-83 Moved from OMUTIL to UTILA
C FORMAL PARAMETERS
C in   datstr   date string
C COMMON BLOCKS
C none
C CALLER various
C METHOD
C Parses the string, converts its parts, and places it in storage.
C**
GENERAL-PURPOSE UTILITIES

$CONTROL check=2
LOGICAL FUNCTION ceq(str1,len1,str2,len2)
    INTEGER len1,len2
    CHARACTER str1*(len1), str2*(len2)

C*                  *** ABSTRACT ***
C*PURPOSE compares two strings, padding on the right with blanks
C* since HP's Fortran does string comparisons differently
C*AUDIT HISTORY
C    MSCarey      2-Feb-83 AUTHOR
C>Type character utility
C#FORMAL PARAMETERS
CIn str1 left character string in comparison
CIn len1 length of str1
CIn str2 right character string
CIn len2 length of str2
C#COMMON BLOCKS
C    none
C#METHOD
C    one line routine. necessary because HP will not consider
C    two identical strings of different length to be equal.
C    Assumes that both strings are left-justified.
C##
SUBROUTINE cfinit

* FORMAL PARAMETER DECLARATIONS *

* ABSTRACT *

C#PURPOSE Command Files INITIALize. Initializes comcfl common block switches to false.

C#AUDIT HISTORY

MSCarey 14-FEB-83 AUTHOR

C#FORMAL PARAMETERS

C NONE

C#COMMON BLOCKS

Cout comcfl command file facility switches and io assignments

C#CALLER mnurun

C#METHOD

C Assignment statements.

C#LOCAL VARIABLES

C none

C#
SUBROUTINE chash(a,len,kmax,nrec,k,amin,amax,ih,nh)

** FORMAL PARAMETER DECLARATIONS **

INTEGER len,kmax,nrec,k,nh, ih(nh)
CHARACTER*(len) a(kmax,nrec), amin, amax

** ABSTRACT **

PURPOSE Returns the sorted order of the char records A based on row k

AUDIT HISTORY

Densmore  19-Jun-83 AUTHOR

TYPE Sort utility

PARAMETERS

a the array of nrec records, each of length kmax
len the number of chars in each element of each record
kmax the number of elements in each record
nrec the number of records
k the element of each record on which to sort
amin a lower bound on the values a(k,*)
amax an upper bound on the values a(k,*)

ih the sorted order of the records contained in a, based on the element k in each record. That is, ih(1) contains the number of the record which appears first when they are given in order; ih(2) contains the number of the second record, etc.

nh the length of the array ih. This number must obviously be >= nrec; ih is actually used as a work area and should be at least 2*nrec, preferably 3*nrec.

METHOD

Assumes that the records are approximately linearly distributed. Takes the value of each record's key and uses it to estimate its sequence number, placing that record's index number in the ih element corresponding to that sequence number. This is repeated for each record, resolving collisions as required. If only a few collisions need to be resolved this is a nearly linear ordering algorithm. At the end the nonzero (unfilled) elements of the ih array are removed and the filled elements left shifted so that the first nrec elements of ih give the ordering information.
C GENERAL-PURPOSE UTILITIES

C $CONTROL check=2
    REAL FUNCTION chashv(s,m)
C
C* FORMAL PARAMETER DECLARATIONS *
C
    INTEGER m
    CHARACTER*1 s(m)
C
C* ABSTRACT **

C* PURPOSE Returns hash value from short char string for chash
C* AUDIT HISTORY
C  Densmore 20-Jun-83 AUTHOR
C* TYPE sort utility
C* FORMAL PARAMETERS
C  Cin  s  the character string
C  Cin  m  its length
C* CALLER chash
C* METHOD
C  Pretends each character is a base-128 digit; assumes blank
C  padding exists in s and that m has the same positive value for
C  every call from a given invocation of chash.
C**
C  CHNALS******************************************************
$CONTROL check=3
   INTEGER FUNCTION chnalo(chn)
   PARAMETER nhd=2
   INTEGER chn(nhd)
C*PURPOSE allocates from chain system. See chnini.
C**

C  CHNDEA******************************************************
$CONTROL check=3
   SUBROUTINE chndeal(chn,item)
   PARAMETER nhd=2
   INTEGER chn(nhd),item
C*PURPOSE deallocates item back into available area of chain
   system. See chnini.
C**

C  CHNFRE******************************************************
$CONTROL check=3
   LOGICAL FUNCTION chnfre(chn,item)
   PARAMETER nhd=2
   INTEGER chn(nhd),item
C*PURPOSE checks availability of item in chain. See chnini.
C**
SUBROUTINE chnini(chn,size,nitems)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER size,nitems,chn(size)

*** ABSTRACT ***

PURPOSE initializes chain allocation system

HISTORY

Densmore 09-Dec-82 AUTHOR

TYPE chain utility

FORMAL PARAMETERS

chn      chain array of length nitems+2; the first word
         contains a copy of nitems, the second contains
         the address of the first available item value.
         The next is initialized to the first item value.

size     the length of the chn array

nitems   the number of items to be chained

METHOD

For a call CALL chnini(chn,size,nitems), this routine initializes

the chn array so that it contains (nitems,1,2,3,4,...,nitems)

Accompanying this routine are three others: chnalo to allocate

new item values, chndeal to deallocate item values by name, and

chnfcre to verify that an item value is free (not normally needed

by the "outside world". Once an item value is allocated, the

idea is that the item value is never again returned by chnalo

unless at some future time that item value is deallocated. These

routines can be used in conjunction with a doubly dimensioned

array which the item values may then serve as indexes.

Chain initialization: CHNINI(chn,size,nitems)

Chain allocation: CHNALO(chn) integer fcn returns item value

Chain deallocation: CHNDEA(chn,item)

Chain free item test: CHNFRE(chn,item) logical fcn

returns true if item is available for allocation.
C CIF

$CONTROL check=2
LOGICAL FUNCTION cif(str1, len1, str2, len2)

INTEGER len1, len2
CHARACTER str1*(len1), str2*(len2)

*** ABSTRACT ***
C PURPOSE performs complete blank strip on both strings; then compares
C AUDIT HISTORY
C Densmore 04-Feb-83 AUTHOR
C TYPE character utility
C FORMAL PARAMETERS
Cin str? the strings to compare
Cin len? the lengths of each string
C LOCAL VARIABLES
C lft? leftmost nonblank
C lgth? length between leftmost and rightmost nonblank
C

10-38
C CKDATE******************************************************************************
$CONTROL check=2
   LOGICAL FUNCTION ckdate(string,length)
      integer length
      character*(length) string
C
*** ABSTRACT ***
C#PURPOSE Checks that string is valid character representation of a date
C#AUDIT HISTORY
C MSCarey 17-mar-83 AUTHOR
C Densmore 28-Apr-83 Extensive mod to check date
C Densmore 14-Oct-83 change lmonth data to Fn ckdati
C#FORMAL PARAMETERS
Cin length length of input string
Cin string string to be checked
C#CALLER various
C#METHOD
C First checks that all characters in the string are blanks,
C digits, or slashes; further that there are exactly two slashes.
C Then the respective numbers are checked for validity.
C**
LOGICAL FUNCTION ckdati(month, day, year)

INTEGER month, day, year

C#PURPOSE Finishes work of ckdata: separate entry point in
C case the work needs to be done from the intermediate
C step of (mm,dd,yy)
C#AUTHOR Densmore 14-Oct-1983
C#FORMAL PARAMETERS

Cin  month, day, year  - INTEGER input date to check
CFunction ckdati  - LOGICAL true if date is a valid one
C"
SUBROUTINE cntinu

*** ABSTRACT ***

** PURPOSE Pauses execution until user hits return.
C Main use is ensuring that error messages stay on screen.
C USES READLN, requires cfinit call before usage.

** AUDIT HISTORY **
C MSCarey 14-dec-83 AUTHOR

** FORMAL PARAMETERS **
C none

** COMMON BLOCKS **
C cin loc system io units

** CALLER **
C various

** METHOD **
C fortran write; call to readln for read

** LOCAL VARIABLES **
C line readln argument
C eof readln argument

**
REAL FUNCTION cputim(dummy)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER dummy

*** ABSTRACT ***

PURPOSE Returns a CPU second value directly from the internal clock
(value MAY NOT BE NORMALIZED to any zero)

AUDIT HISTORY

Densmore 27-Oct-82 AUTHOR

TYPE Simple function

FORMAL PARAMETERS

COMMON BLOCKS

CIN dummy not used...just allows the function call

FUNCTION cputim output real value...must be normalized
to be of use by saving the value returned

by the first call.

METHOD

Calls the PROCTIME intrinsic, which gives a doubleword in millisecs

```
SUBROUTINE crngi(i, low, high, text)
INTEGER i, low, high
CHARACTER*255 text

PURPOSE checks that low<=i<=high for INTEGER*2 variables

HISTORY
Densmore 28-Oct-82 AUTHOR

PARAMETERS

  i  value
  low  lowest possible value
  high  highest possible value
  text  delimited text string giving caller, etc.
SUBROUTINE crngi4(i, low, high, text)
INTEGER*4 i, low, high
CHARACTER*255 text
C*PURPOSE checks that low <= i <= high for INTEGER*2 variables
C#AUDIT HISTORY
C Densmore 28-Oct-82 AUTHOR
C#FORMAL PARAMETERS
Cin i value
Cin low lowest possible value
Cin high highest possible value
Cin text delimited text string giving caller, etc.
SUBROUTINE csinit

C PURPOSE command stack initializer
C the routines documented here handle a command stack used to implement
C recursive commands.
C AUTHOR Kerchner December 14, 1981
C TYPE Command Stack Utility
C FORMAL PARAMETERS
C
C SUBROUTINE csinit
C SUBROUTINE cspop(n)
C SUBROUTINE cspopr(n,A)
C SUBROUTINE cspsh(a)
C SUBROUTINE cspsh2(a2)
C SUBROUTINE cspshi(n,A)
C
C n: int*2 length a: typeless*4 item A: typeless*4 array
C
C SUBJECT csinit - initializes command stack system
C SUBROUTINE cspop(n) - pops n items off stack (w/ no read)
C SUBROUTINE cspop(n,A) - pops n items off stack into array A
C SUBROUTINE cspsh(a) - pushes the item a onto the stack
C SUBROUTINE cspsh2(a2) - pushes int*2 item a2 onto stack
C SUBROUTINE cspshl(n,A) - pushes the n items in A onto stack
C INTEGER*4 FUNCTION csrd(n) - returns the n'th item on the stack; 
C n=1 yields the top of the stack
C INTEGER*2 FUNCTION csrd2(n) - returns n'th item as a *2 integer
C SUBROUTINE csrdl(n,A) - returns to array A the top n items
C
C cspsh, cspsh2, and cspshl will overflow if stack array lacks room.
C cspop, csrd, csrd2, and csrdl will underflow if too many items read.
C Subroutine cspop never underflows.
SUBROUTINE cspop(n)
INTEGER n
C#PURPOSE pops n items off stack; see csinit for complete documentation
C#AUTHOR Kerchner December 14, 1981
C#TYPE Command Stack Utility
C##

SUBROUTINE cspopr(ni,array)
INTEGER ni
INTEGER*4 array(ni)
C#PURPOSE pops ni items off stack into array. complete doc in csinit
C#AUTHOR Kerchner December 14, 1981
C#TYPE Command Stack Utility
C##

SUBROUTINE cspsh(item)
INTEGER*4 item
C#PURPOSE pushes item onto top of stack. complete doc in csinit
C#AUTHOR Kerchner December 14, 1981
C#TYPE Command Stack Utility
C##

SUBROUTINE cspsh2(item2)
INTEGER*2 item2
C#PURPOSE pushes item2 onto top of stack. complete doc in csinit
C#AUTHOR Densmore 3 Feb 1983
C#TYPE Command Stack Utility
C##
GENERAL-PURPOSE UTILITIES

C CSPLH------------------------------
$CONTROL check=2
SUBROUTINE cspshl(ni,array)
INTEGER ni
INTEGER*4 array(ni)
C#PURPOSE pushes ni items in array onto stack. complete doc in csinit
C#AUTHOR Kerchner December 14, 1981
C#TYPE Command Stack Utility
C##

C CSRD-------------------------------
$CONTROL check=0
INTEGER*4 FUNCTION csrd(n)
INTEGER n
C#PURPOSE returns n'th item on stack;
see csinit for complete documentation.
C#AUTHOR Kerchner December 14, 1981
C#TYPE Command Stack Utility
C##

C CSRD2------------------------------
INTEGER*2 FUNCTION csrd2(n)
INTEGER n
C#PURPOSE returns n'th item on stack, assuming it to be a *2 int;
see csinit for complete documentation.
C#AUTHOR Densmore 3 Feb 1983
C#TYPE Command Stack Utility
C##

C CSRL-------------------------------
$CONTROL check=2
SUBROUTINE csrdl(ni,array)
INTEGER ni
INTEGER*4 array(ni)
C#PURPOSE reads top ni items on stack, nondestructively.
Complete doc in csinit.
C#AUTHOR Kerchner December 14, 1981
C#TYPE Command Stack Utility
C##
SUBROUTINE CVTDAT (INSTR, IM, ID, IY, ERR)

** FORMAL PARAMETER DECLARATIONS **

CHARACTER INSTR *LLINE
LOGICAL ERR
INTEGER IM, ID, IY

ABSTRACT

`PURPOSE` decodes string of form 'month/day/year', where year is specified as 1982 for example, into IM=month, ID=day, IY=year.

`AUDIT HISTORY`
MEMutchler 17 JAN 83 AUTHOR
MEMutchler 7 FEB 83 TESTER (program tstdat)

`TYPE` mnurun utility

`FORMAL PARAMETERS`

`IN` instr date string of form mm/dd/yyyy
`OUT` im month from 1 to 12
`OUT` id day from 1 to 32
`OUT` iy year as in 1983
`OUT` err true iff string not of correct form

`COMMON BLOCKS`

`IN` incpar global parameter statements

`CALLER` dpmenu

`METHOD` split 'string' into month, day, and year pieces by keying on '/', the delimiter, and convert them to their integer values.

`LOCAL VARIABLES`

string local of instr
lenstr length of 'string' in non-blank characters
delim character representation of '/'
piece character string holding piece to be converted to integer.

*** INCLUDES and LOCAL DECLARATIONS ***

INTEGER LENSTR, I
CHARACTER PIECE*4, DELIM*1, STRING*LLINE
DATA DELIM('/')/
SUBROUTINE datemk(imark, month, day, year)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER*4 imark
INTEGER month, day, year

*** ABSTRACT ***

PURPOSE Given a mark day (mrkday), returns date (mrkday's inverse)

AUDIT HISTORY
Densmore 31-May-83 AUTHOR

TYPE date utility

FORMAL PARAMETERS

imark input mark day -- number of days since 31-Dec-1600
month month corresponding to mark day (1-12)
day day corresponding to mark day (1-31)
year year corresponding to mark day

CONSTANTS

mark for 31-Dec-1600 is defined to be zero
mark for 31-Dec-1999 is mk1999=145731 -- year 2000 is the only century leap year
mark for 31-Dec-2000 is mk2000=146097 -- same comment
number of days from 31-Dec-NN00 to 31-Dec-(NN+1)00 is nd100=36524
number of days in a 4-year period is nd4=1461


SUBROUTINE datepl(m,d,y)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER m,d,y

*** ABSTRACT ***

PURPOSE adds one day to the date

AUDIT HISTORY

Denimore 01-Jun-83 AUTHOR

TYPE date utility

FORMAL PARAMETERS

IN/OUT m,d,y month (1..12), day(1..31), year incremented

10-50
SUBROUTINE datstr(string)
    character*8 string

*** ABSTRACT ***

PURPOSE returns current date as MM/DD/YY

AUDIT HISTORY
MSCarey 02-feb-83 AUTHOR

FORMAL PARAMETERS
Cout string see purpose

COMMON BLOCKS
none

CALLER various

METHOD
Calls system intrinsic dateline and decode char month to num

LOCAL VARIABLES
bytara argument for intrinsic call
month holds strings for comparison against bytara
mnum number of the month returned by dateline
$CONTROL check=3
INTEGER*4 FUNCTION dclrfy(rawdat)
C
*** FORMAL PARAMETER DECLARATIONS ***
INTEGER*4 rawdat
C
*** ABSTRACT ***
C#PURPOSE Clarifies a raw RELATE DDATE. See explanation
given in the data type include file TDDATE
C#AUDIT HISTORY
Densmore 26-Apr-83 AUTHOR
C#TYPE Date utility
C#FORMAL PARAMETERS
Cin rawdat raw date in RELATE format
C#METHOD various
C zeros the unused bits: L0, L13-15, R9-15
INTEGER*4 FUNCTION ddate(dum)
    integer dum

*** ABSTRACT ***

C* PURPOSE Returns the current date in RELATE I*4 format, which
C is year in bits 1-12 of left word, month in 0-3 of right word,
C day in 4-8 of right word, and all other bits unused.
C See the documentation on the data type DDATE in TDDATE.INCL.

C* HISTORY
C MSCarey 28-Feb-83 AUTHOR
C Densmore 26-Apr-83 Moved from DMUTIL to UTILA
C and fixed LEAP YEAR part
C Densmore 14-Oct-83 Changed lmonth data to function

C* FORMAL PARAMETERS
C dummy
C none
C various

C* METHOD
C Calls calendar intrinsic, converts to month-day-year, and packs
C output variable.

C* LOCAL VARIABLES
C date as returned by intrinsic
C year
C day in year
C a running total of days
C month of year
C day of month

C
/* GENERAL-PURPOSE UTILITIES */

C** FUNCTION ddtocd(ddate) **
C** INTEGER*4 ddate **

** ABSTRACT **
** PURPOSE ** Converts a DDATE into a string MM/DD/YYYY
** AUDIT HISTORY **
Densmore 26-Apr-83 AUTHOR
** TYPE ** Date utility
** FORMAL PARAMETERS **
Cin ddate a DDATE as described in TDDATE.INCL --
C it is in RELATE format: Bits 1-12 = year
C Bits 0-3 = Month, and Bits 4-8 = Day
** CALLER ** various
** METHOD ** unpacks ddate and encodes string

10-54
SUBROUTINE ddtoid(ddate,month,day,year)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER*4 ddate
INTEGER month, day, year

*** ABSTRACT ***

C#PURPOSE Converts a DDATE into a string MM/DD/YYYY
C See the include file TDDATE.INCL describing data type
C#AUDIT HISTORY
C Densmore 26-Apr-83 AUTHOR
C#TYPE Date utility
C#FORMAL PARAMETERS
Cin ddate a DDATE as described in TDDATE.INCL --
C it is in RELATE format: Bits L1-12=year
C Bits R0-3=Month, and Bits R4-8=Day
Cout month the integer month given in ddate [1..12]
Cout day the integer day [1..31]
Cout year the integer year (ie. 1983)
C#CALLER various
C##
**GENERAL-PURPOSE UTILITIES**

C DEARLY -------------------------------

$CONTROL check=3
LOGICAL FUNCTION DEARLY (FRSTDAT,LASTDAT)
C *** FORMAL PARAMETER DECLARATIONS ***
INTEGER*4 FRSTDAT,LASTDAT
C *** ABSTRACT ***
C PURPOSE True if first date earlier than last date. Clarifies.
C AUDIT HISTORY
C
C
C
C
C
C
C CALLER FLREPT AND BGREPT
C METHOD
C DOES A CLARIFY AND THEN USES TDDATE ROUTINES
C LOCAL VARIABLES
C datfrst flag
C

10-56
C DEBUG ***********************************************
$CONTROL check=3,SEGMENT=utlr
LOGICAL FUNCTION debug(idum)
C
    integer idum
C
*** FORMAL PARAMETER DECLARATIONS ***
C
*** ABSTRACT ***
C PURPOSE Checks lprnton job control word to see if in debug mode.
C AUDIT HISTORY
C MSCarey 28-jul-84 AUTHOR
C TYPE utility
C FORMAL PARAMETERS
Cin idum dummy parameter to meet HP calling standards
C COMMON BLOCKS
C none
C CALLER various
C METHOD
C Use the findjcw system intrinsic to read the lprnton job
C control word. If it is 1, debug is true.
C LOCAL VARIABLES
C jcwname name of jcw to read
C
10-57
SUBROUTINE delim(string, first, last, length)

PARAMETER m=1024
CHARACTER*1 string(m)
INTEGER first, last, length

*** ABSTRACT ***

C#PURPOSE to discover the extent of a delimited text string
C#AUDIT HISTORY
C Densmore 27-Oct-82 AUTHOR
C#TYPE Simple subroutine
C#FORMAL PARAMETERS
Cin string delimited text string -- the text is delimited
C first location after first nonblank char in string
C last location before second occurrence of delimiter
C length the number of characters in the enclosed string
C
C#METHOD
C Uses the HP Fortran byte addressing capability to locate
C the first nonblank character, then the second occurrence
C of the delimiter. Note that the input string may be any
C length up to m characters.
C##
C INDEX

$control check=2

INTEGER FUNCTION dindex(string,lstr,substr,lsub)

character*1 string(255),substr(255)

integer lstr,lsub

*** ABSTRACT ***

C$PURPOSE A version of HP function 'index' for delimited search string

C$AUDIT HISTORY

C MSCarey 09.may.83 AUTHOR

C$FORMAL PARAMETERS

Cin string possibly delimited string to look for substring in

Cin substr substring to look for in string

Cin lstr length of string

Cin lsub length of substring

C$COMMON BLOCKS

C none

C$CALLER various

C$METHOD

C Loop over string, looking for a match

C$
GENERAL-PURPOSE UTILITIES

C     DPAUSE*********************************************************************
$CONTROL check=3
SUBROUTINE dpause(wait)
C*
*** FORMAL PARAMETER DECLARATIONS ***
C*
real wait
C*
*** ABSTRACT ***
C#PURPOSE Causes process to pause wait seconds.
C#AUDIT HISTORY
C  MSCarey 10-dec-83 AUTHOR
C#FORMAL PARAMETERS
Cin wait number of seconds to pause
C#COMMON BLOCKS
C#CALLER various
C#METHOD
C  Calls system intrinsic pause.
C#LOCAL VARIABLES
C  none
C#
SUBROUTINE eject(unit)
    INTEGER unit

*** ABSTRACT ***
PURPOSE sends a page eject down to specified unit
AUDIT HISTORY
Densmore 21-Mar-83 AUTHOR
TYPE screen utility
FORMAL PARAMETERS
    unit logical unit number
COMMON BLOCKS
    tty terminal parameters
SUBROUTINE ELIMBL (INSTR,LSTR,OUTSTR,LOSTR)

CHARACTER OUTSTR*(LSTR),INSTR*(LSTR)
INTEGER LSTR,LOSTR

PURPOSE eliminate leading and trailing blanks from a string
and give its non-padded length

AUDIT HISTORY
MEMutchler 18 JAN 83 AUTHOR

TYPE character string utility

FORMAL PARAMETERS
CIN instr text to be stripped of leading and trailing blanks
CIN lstr maximum length of strings
COUT outstr text stripped of leading and trailing blanks
COUT lostr length of outstr

COMMON BLOCKS
CIN incpar global parameter statement

METHOD
loop through string and count

LOCAL VARIABLES
C string temporary storage of text
C blank ' '

**
SUBROUTINE ERRMSG (DELSTR)

*** FORMAL PARAMETER DECLARATIONS ***

%INCLUDE INCPAR
CHARACTER DELSTR, LLINE

*** ABSTRACT ***

CSPURPOSE Takes a delimited text string error message held at delstr and outputs it to iout

C#AUDIT HISTORY
C MEMutchler 18 JAN 83 AUTHCR
CSTYPE utility
C#FORMAL PARAMETERS
Cin delstr delimited text string to be output
C#COMMON BLOCKS
Cin incpar global parameter statement
Cin ioc i/o file assignments
C#METHOD Undelimit text string, get its length and write it.
C#LOCAL VARIABLES
C output undelimited text string
C lenout length of 'output' in non-blank characters
C   FDDATE**************************************************************
$CONTROL segment=asgn,check=3
   INTEGER*4 FUNCTION fddate(ddatel,idurat)
C*
   INTEGER idurat
   INTEGER*4 ddate1
C*
   *** FORMAL PARAMETER DECLARATIONS ***
C*
C#PURPOSE Returns date of the first date in the period
C#AUDIT HISTORY
C  Densmore      17-Jun-83  AUTHOR
C#TYPE date utility
C#FORMAL PARAMETERS
Cin  ddate1 date within the first period
Cin  idurat duration (of each period) index
C  1=Year 2=Quarter 3=Month 4=Week 5=Day
C#COMMON BLOCKS
Cin  tddate ddate data type block
C#METHOD
C  Convert, push back to period's start, and convert back.
C##
LOGICAL FUNCTION fexist(filnam,len)
  integer len
  character*(len) filnam

*** ABSTRACT ***

PURPOSE Returns true if the given file already exists as a
permanent file.

AUDIT HISTORY
MSCarey 28-feb-83 AUTHOR

FORMAL PARAMETERS
  Cin len length of string
  Cin filnam name of file to check for, with extents if any

COMMON BLOCKS
  none

CALLER various

METHOD
Calls intrinsic fopen specifying file as old. If error indicating
it doesn't exist then fexist is false. Close file if open succeeds.
Test is on both job temporary file domain and perm file domain.

LOCAL VARIABLES
  foptions bit-map argument for fopen
  aoptions "ifoptions"
  iaoptions"
  filnum MPE file number returned by fopen
  ercode error code returned by fcheck
  disp file disposition argument for fclose
  string warning message buffer

10-65
C SUBROUTINE filcls(unit,ok,param)
    integer unit
    character*255 param
   logical ok
C**** ABSTRACT ****
C** PURPOSE Closes files opened by filopn.
C** AUDIT HISTORY
C MSCarey 27-JAN-83 AUTHOR
C** FORMAL PARAMETERS
  Cin unit fortran logical unit to be closed
  Cin param delimited string holding control arguments separated
    by commas. Options are limited to SAVE,DELETE,MEONLY
  Cout ok true if close successful
C** COMMON BLOCKS
  Cio untref cross ref of MPE file number with fortran unit nums
C** CALLER various
C** METHOD
  C Calls system intrinsic fclose.
C** LOCAL VARIABLES
  C dispos transfers file disposal status
  C secode transfers file security status
  C mesg holds an error message
  C fnum MPE file number
  C ercode error code returned by fcheck
C**
abstract

SUBROUTINE filopn(unit,ok,param)
   integer unit
   logical ok
   character*255 param

C*  ABSTRACT ***

C* PURPOSE Opens HP files programmatically.
C* AUDIT HISTORY
C  MSCarey  30-JAN-83 AUTHOR
C* FORMAL PARAMETERS
Cin     unit  fortran logical unit number
Cout    ok    flag set to true if open successful; if false,
           likely cause is someone else having lock or
           sole access to desired file. More serious errors
           cause abort calls from this routine.
Cin     param delimited character string containing legal
           arguments separated by commas, as in
           "NAME=JUNK,NEW,ASC,FIXL,SEQL,RECL=128,NREC=1000;"
C* COMMON BLOCKS
Cout    untref cross ref of MPE file nums & fortran logical units
C* CALLER various
C* METHOD
C Decodes arguments, checks for consistency, and calls MPE intrinsic
C FOPEN and fortran library routine FSET. See intrinsics manual,
C Fortran manual section 8 for more on these. String argument is
C decoded into two arrays, one holding params and the other values
C attached to the parameters where applicable. For each, a list of
C legal parameters is searched for a match. The index of the match
C is used as a reference by a computed goto to code setting parameters
C for the fopen call. Error checking is done after the FOPEN by a call
C to intrinsic fcheck to identify conditions mandating an abort.
C* LOCAL VARIABLES
C nparms number of string parameters decoded
C fooption bit-mapped word for passage to FOPEN
C loption logical of this word
C aoption similar
C laoption "
C tooption "
C ltoption "
C mesg   error message
C filz   file size as a double integer, required by FOPEN
C fnum   MPE system file number
C ercode error code returned by fcheck
C block   number of records per block
C arg    array holding decoded alpha parameters
C value   array holding values corresponding to args
C option array initialized to legal arg values
C name   name of file to be opened

10-67
C group group user is currently logged onto
C filsz maximum size of file in records (block if fixl)
C nparms number of character parameters found
C recsiz size of record in bytes
C recpbl number of records per block
C igo computed goto index
C argnum number of argument being processed by goto

10-68
C FINMEM **********************************************
$CONTROL segment=seg'
   SUBROUTINE finmem(id,code)
C*  *** FORMAL PARAMETER DECLARATIONS ***
      integer id,code
C*  *** ABSTRACT ***
C#PURPOSE  De-allocates an extended memory buffer.
C#AUDIT HISTORY
C  MSCarey  11-aug-83 AUTHOR
C#FORMAL PARAMETERS
Cin   id   operating system id code
Cin   code  id code supplied by user to inimem
C#COMMON BLOCKS
C#CALLER various
C#METHOD
C  Calls freedseg
C##
C     SUBROUTINE FLT ( BUFFER, LENBUF, ERROR, NUMBER )
C* *** FORMAL PARAMETER DECLARATIONS ***
INTEGER LENBUF
CHARACTER BUFFER *( LENBUF )
REAL NUMBER
LOGICAL ERROR
C* *** ABSTRACT ***
C* PURPOSE set number<-rnum(buffer), if possible pointed to by IPTR
C* AUDIT HISTORY
C    MEMutchler 7 FEB 83 AUTHOR
C    MEMutchler 7 FEB TESTER
C* TYPE convert string to corresponding real value if possible
C* FORMAL PARAMETERS
Cin buffer string containing character version of real
Cin lenbuf non-blank length of buffer
Cout error true iff buffer doesn't contain a real
Cout number real number found in buffer
C* COMMON BLOCKS NONE
C* METHOD determine if real number in string, else err = true
C* LOCAL VARIABLES
C none
C#70
GENERAL-PURPOSE UTILITIES

C GDATEP******************************************************

$CONTROL segment=asgnd,check=3
INTEGER*4 FUNCTION gdatep(pern,idurat,fddate)
C* *** FORMAL PARAMETER DECLARATIONS ***
INTEGER pern,idurat
INTEGER*4 fddate
C* *** ABSTRACT ***
C* PURPOSE Returns the date of the first day in the pern'th period
C* AUDIT HISTORY
C Densmore 17-Jun-83 AUTHOR
C TYPE date utility
C FORMAL PARAMETERS
Cin pern period number to be converted to a date
Cin idurat duration (of each period) index
C 1=Fyear 2=Cyear 3=quarter 4=month 5=week 6=day
Cin fddate date of first day of first period
C COMMON BLOCKS
Cin tddate data type for RELATE ddate
C METHOD
C convert, increment, convert
C*

10-71
SUBROUTINE GETGRP

*** FORMAL PARAMETER DECLARATIONS ***

Purpose: get group name and name's length in which runtime

menu system files should be located

Audit History

MEMutchler 10-MAR-83 AUTHOR

Type: menu system utility

Formal Parameters: none

Common Blocks:

Cut   envirn holds info about runtime environment

Caller: ppinit and inimnu

Method:

Inspect the proper Job Control Word flag's

value. JCW should be absent or 0 unless the user

has given the DEVELOP UDC command.

Set variables used to determine which group

system files and relations are to be found in.

Local Variables:

jcnam  job control word name

10-72
SUBROUTINE getmem(id, length, source, start)

**FORMAL PARAMETER DECLARATIONS**

- integer id, length, start
- logical source(1)

**ABSTRACT**

Swaps data from extended memory into source array.

**AUDIT HISTORY**

MSCarey 11-aug-83 AUTHOR

**FORMAL PARAMETERS**

- `id`: operating system id code for area
- `length`: number of *2 words to swap*
- `source`: target array for words
- `start`: starting position in extended mem to grab from

**COMMON BLOCKS**

- none

**CALLER**

- various

**METHOD**

- Calls dmovin.

**LOCAL VARIABLES**

- `lid`: segment id
GENERAL-PURPOSE UTILITIES

C

GPERN

$CONTROL segment=asgn, check=3

INTEGER FUNCTION gperr(ddatel,numper, idurat, fddate)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER numper, idurat
INTEGER*4 ddatel, fddate

*** ABSTRACT ***

C* PURPOSE Returns period number given a RELATE ddate
C* AUDIT HISTORY
C Densmore 17-Jun-83 AUTHOR
C* TYPE date utility
C* FORMAL PARAMETERS
C* Cin ddatel the date to be converted
C* Cin numper maximum number of periods
C* Cin idurat duration (of each period) index
C 1=Year 2=Cyear 3=quarter 4=month 5=week 6=day
C* Cin fddate date of the first day in the first period
C* function gperr returned period number, in [0..numper];
C 0 is returned if ddatel is outside the range
C* of valid period numbers
C* COMMON BLOCKS
C* Cin tddate RELATE ddate data type block
C* LOCAL VARIABLES
C C Convert dates, take difference. SEE SIMILAR CODE IN ASNLBS.
C* LOCAL VARIABLES
C C fm, fd, fy month/day/year of first period's first day
C C dm, dd, dy month/day/year corresponding to ddatel
C C fidate/lidate first/last indexed period in absolute time
C##
C IAND*********************************************************************** *
$CONTROL check=3
INTEGER FUNCTION iand(m,n)
INTEGER m,n
C bitwise ...; uses HP-FTN 16-bit exprs...Densmore 28 July 1983
INTEGER jm,jn
LOGICAL lm,ln
EQUIVALENCE (jm,lm), (jn,ln)
C*ENDDEC
jm = m
jn = n
ln = lm.AND.ln
iand = jn
RETURN
C
ENTRY lor(m,n)
jm = m
jn = n
ln = lm.OR.ln
iand = jn
RETURN
END
C IDAYS*******************************************************************************
$CONTROL check=3
  INTEGER FUNCTION idays(ml,dl,yl,m2,d2,y2)
C*  *** FORMAL PARAMETER DECLARATIONS ***
C*  INTEGER ml,dl,yl,m2,d2,y2
C*  *** ABSTRACT ***
C*PURPOSE Returns the number of days between two dates.
C#AUDIT HISTORY
C  Densmore  04-May-83 AUTHOR
C#TYPE date utility
C#FORMAL PARAMETERS
Cin  ml,dl,yl month/day/year of first date
Cin  m2,d2,y2 month/day/year of second date
C#METHOD Subtracts mrkduys (which checks date validity);
C  See Function JDAYS for an INTEGER*4 version.
C##
SUBROUTINE idtodd(ddate, month, day, year)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER*4 ddate
INTEGER month, day, year

*** ABSTRACT ***

PURPOSE: Converts to RELATE internal DDATE format; see TDDATE.INCL

AUDIT HISTORY
Densmore 11-May-83 AUTHOR

TYPE: Date Utility

FORMAL PARAMETERS
Cout  ddate  output date in RELATE format
Cin   month, day, year  input date in integer form

C#
C INIOC
$CONTROL SEGMENT-MENU
   SUBROUTINE INIOC
C*  *** ABSTRACT ***
C*PURPOSE initializes file assignments of i/o files found in
C  common /ioc/
C*AUDIT HISTORY
C  MEMutchler 18 JAN 83 AUTHOR
C  MEMutchler  8 FEB 83 TESTER
C*TYPE mnugen and mnurun utility
C*COMMON BLOCKS
Cout  ioc  i/o file assignments
C*
**INIMEM**

`SUBROUTINE inimem(id,len,code,unique)`

*** FORMAL PARAMETER DECLARATIONS ***

```
integer id,code,len
logical unique
```

*** ABSTRACT ***

**PURPOSE** Allocates an extended memory buffer. On HP 3000 this means getting an extra data segment.

**AUDIT HISTORY**

MSCarey 11-aug-83 AUTHOR

**FORMAL PARAMETERS**

- **Cout id** (id number returned for use by putmem and getmem)
- **Cin code** (id code provided by user to make unique segment)
- **Cin len** (length of buffer desired in 2-byte word)
- **Cout unique** (true if buffer requested did not already exist)

**COMMON BLOCKS**

- **CALLER** various
- **METHOD**

Call to getdseg.

**END**
*CONTROL segment=seg'
  SUBROUTINE iniPrc

*** FORMAL PARAMETER DECLARATIONS ***

*** ABSTRACT ***

PURPOSE  INITIALIZE module PROCess. Does all necessary
           initialization for a module son process.

AUDIT HISTORY

MSCarey  29-jun-83 AUTHOR

FORMAL PARAMETERS

none

COMMON BLOCKS

name pvalue menu system parameter values

CALLER various

METHOD

Calls to other initialization routines. Swaps in /pvalues/
/scenar/, /lprnts/, /icc/, /io/ from an extra data segment

LOCAL VARIABLES

name file name icoref.mnurel/makmenu
SUBROUTINE initio

C Initializes the most necessary I/O unit numbers
C Note that this routine does NOT use include directives
C so that the utility library need not be compiled with INCL
C
C IXSUM************************************************************
$CONTROL check=3
   INTEGER FUNCTION ixsum(n,v)
C* *** FORMAL PARAMETER DECLARATIONS ***
   INTEGER n
   INTEGER v(n)
C* *** ABSTRACT ***
PURPOSE Sums the cross section vector V -- integer sum
AUDIT HISTORY
Densmore 17-Mar-83 AUTHOR
TYPE manual assigner routine
FORMAL PARAMETERS
  n length of v
  V vector of integers whose elements to sum
function ixsum
     / i=1 n 
COMMON BLOCKS
none
C

10-82
*CONTROL check=3

INTEGER*4 FUNCTION jdays(ml,d1,y1,m2,d2,y2)

INTEGER ml,d1,y1,m2,d2,y2

*** ABSTRACT ***

C#PURPOSE Returns *4 number of days between two dates.

C#AUDIT HISTORY

C#TYPE date utility

C#FORMAL PARAMETERS

Cin ml,d1,y1 month/day/year of first date

Cin m2,d2,y2 month/day/year of second date

C#METHOD Subtracts mrkdays (which checks validity);

C See Function IDAYS for an INTEGER*2 version.
C SUBROUTINE jhash(a,kmax,nrec,k,amin,amax,ih,nh)
C* *** FORMAL PARAMETER DECLARATIONS ***
INTEGER kmax,nrec,k,nh, ih(nh)
INTEGER*4 a(kmax,nrec), amin, amax
C* *** ABSTRACT ***
C* PURPOSE Returns the sorted order of the I*4 records A, based on row k
C AUDIT HISTORY
C Densmore 16-Jun-83 AUTHOR
C TYPE Sort utility
C FORMAL PARAMETERS
Cin a the array of nrec records, each of length kmax
Cin kmax the length of each record
Cin nrec the number of records
Cin k the element of each record on which to sort
Cin amin a lower bound on the values a(k,*)
Cin amax an upper bound on the values a(k,*)
Cout ih the sorted order of the records contained in a,
C based on the element k in each record. That is, ih(1) contains
C the number of the record which appears first when they are given in order; ih(2)
C contains the number of the second record, etc.
C nh the length of the array ih. This number must
C obviously be >= nrec; ih is actually used as a
C work area and should be at least 2*nrec, preferably
C 3*nrec.
C METHOD
C Assumes that the records are approximately linearly distributed.
C Takes the value of each record's key and uses it to estimate its
C sequence number, placing that record's index number in the ih
C element corresponding to that sequence number. This is repeated
C for each record, resolving collisions as required. If only a
C few collisions need to be resolved this is a nearly linear order-
C ing algorithm. At the end the nonzero (unfilled) elements of the
C ih array are removed and the filled elements left shifted so that
C the first nrec elements of ih give the ordering information.

10-84
SUBROUTINE KFIX ( BUFFER, LENBUF, ERROR, NUMBER )

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER LENBUF
CHARACTER BUFFER *(LENBUF)
INTEGER NUMBER
LOGICAL ERROR

*** ABSTRACT ***

PURPOSE
set number = inum(buffer), if possible
pointed to by IPTR

AUDIT HISTORY
MEMutchler 7 FEB 83 AUTHOR
MEMutchler 7 FEB TESTER

TYPE
convert string to corresponding integer value if possible

FORMAL PARAMETERS

cin buffer string containing character version of integer
clenbuf non-blank length of buffer
cout error true iff buffer doesn't contain a integer
number integer number found in buffer

COMMON BLOCKS
NONE

METHOD
determine if integer number in string, else err = true

LOCAL VARIABLES
none
SUBROUTINE LABORT (INTVAR, STRING)

%INCLUDE INC PAR
CHARACTER STRING*LLINE
INTEGER INTVAR

** ABSTRACT **

** PURPOSE causes a program abort and writes a diagnostic message **

** AUDIT HISTORY **
MEMutchler 18 JAN 83 AUTHOR

** TYPE **
abort and message

** FORMAL PARAMETERS **

** COMMON BLOCKS **

** LOCAL VARIABLES **

** METHOD **
concatenate to get dts string to output

** LOCAL VARIABLES **


SUBROUTINE LATDAT (DATE)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER*4 DATE

*** ABSTRACT ***

C#PURPOSE get maximum date value
C#AUDIT HISTORY
C MEMutchler 31 may 83 AUTHOR
C#TYPE relate date utility
C#FORMAL PARAMETERS
C: date maximum date value
C#METHOD
C set date to greatest *4 value and clarify
C#
**LOGICAL FUNCTION**

```c
lbit(word,pos)
```

**FORMAL PARAMETER DECLARATIONS**

```c
INTEGER word,pos
```

**ABSTRACT**

returns whether bit "pos" of "word" is set

**PURPOSE**

**FORMAL PARAMETERS**

- `word` a sixteen-bit word, of which only first 15 are used
- `pos` bit position: [1..15]

**METHOD**

Note that the bits are numbered from right to left.
LOGICAL FUNCTION LETNUM(string, len)
integer len
character*255 string

PURPOSE checks a string for characters other than letters or numbers. True if no such characters.

MSCarey 28-feb-83 AUTHOR

C#FORMAL PARAMETERS
Cin len length of input string
Cin string string to check

C#COMMON BLOCKS
C none

C#METHOD
C Looks for characters outside permitted octal code ranges

10-89
**LETONL**

*control segment=dmaint

LOGICAL FUNCTION letonl(string,len)

integer len

character*(len) string

*** ABSTRACT ***

**PURPOSE** Checks a string to be sure it contains only letters or ".".

**AUDIT HISTORY**

MSCarey 17-mar-83 AUTHOR

**FORMAL PARAMETERS**

Cin len length of input string

Cin string string to be checked

**COMMON LOCKS**

C none

**CALLER** various

**METHOD**

C Checks to make sure each byte is within the proper octal range.

**"**
**GENERAL-PURPOSE UTILITIES**

```c
C     LISTON ****************************************************
*CONTROL segment-seg'
SUBROUTINE liston(scenar,mnumam,list,lchars,mxnlst,numon,tmany)
    character*(lchars) list(mxnlst)
    character*12 scenar,mnumam*8
    integer lchars,mxnlst,numon
    logical tmany
C*     *** ABSTRACT ***
C*PURPOSE  Reads a list menu relation and returns a list
C     of candidates which are "on".
C*AUDIT HISTORY
C    MSCarey     02-Jun-83 AUTHOR
C    Denamore    10-Jun-93 Added neglected 'tmany' formal
C*FORMAL PARAMETERS
Cin    scenar   current scenario name
Cin    mnumam  list menu for which list is desired
Cin    lchars  max chars in a list candidate
Cin    mxnlst  max number of candidates returnable
Cout   list    list of candidates which are "on"
Cout   numon   number of candidates returned
Cout   tmany   more found on than allowed by mxnlst
C*COMMON BLOCKS
Cin    prmers  permanently open cursor indexes
Cin    envrn   group name for list relations
Cin    rcrd@1  buffer for list retrievals
C*CALLER various
C*METHOD
C    Look in the cross reference relation for the relation name
C    holding candidate statuses for the given menu. Open that
C    relation.
C    Calc to the first tuple for the given scenario
C    Read sequentially until all tuples for that scenario are
C    found, placing the candidate field for each on the list
C    if its status is "on". Close the relation and return.
C*LOCAL VARIABLES
C    cand    candidate name
C    stat    candidate status
C*          
```

10-91
**GENERAL-PURPOSE UTILITIES**

```c
INTEGER FUNCTION Imonth(month,year)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER month,year

*** ABSTRACT ***

C& PURPOSE Returns number of days in given month

C& AUDIT HISTORY

C Densmore 11-Oct-83 AUTHOR

C& TYPE Date utility

C& FORMAL PARAMETERS

Cin month integer representation of month

Cin year integer year (e.g. 1983)

C& METHOD

C uses array indexed by month; feb is special case.
C&
```

10-92
SUBROUTINE lpsend(unit)

integer unit

Closes printer spool file, causing output to commence.

MSCarey 20-sep-83 AUTHOR

utility

unit number lp file is open on

various

Closes file unless it is $stdlist (terminal).

10-93
C    LPSET************************************************************
$CONTROL segment=devctrl,check=3
    SUBROUTINE lpset(unit)
C*    *** FORMAL PARAMETER DECLARATIONS ***
C*    INTEGER unit
C*    *** ABSTRACT ***
C* PURPOSE Determines Line Printer UNIT number
C and opens spool file if appropriate
C AUDIT HISTORY
C    Densmore    25-May-83    AUTHOR
C    MSCarey     29-Jun-83    'undummied' to use PVALUE
C TYPE    I/O Utility
C FORMAL PARAMETERS
C COMMON BLOCKS
C unit unit to use for printer output
C ioc    also changes lp unit in this common
C CALLER    anyone who wants to write to a line printer
C**
SUBROUTINE lstrng(sin,lin0,sout,lout,mlout)
*FORMAL PARAMETER DECLARATIONS *
INTEGER lin0,lout,mlout
CHARACTER*1 sin(1024), sout(mlout)
*** ABSTRACT ***
C* PURPOSE moves possibly delimited string sin to sout
C*AUDIT HISTORY
C Densmore 15-Dec-82 AUTHOR
CTYPE string manipulation utility
C$FORMAL PARAMETERS
Cin sin input string
Cin lin0 length of sin; if lin0=0, then sin is a OTS
Cout sout output string
Cout lout actual length of sout
Cin mlout maximum allowable length of sout
C$METHOD
C Determines first, last, and length; uses HP Fortran character
C assignment with substring operators. Note that sout may share
C addresses with sin, since the assignment operations are
C buffered.
C
C

LTRIM

*CONTROL check=2

INTEGER FUNCTION ltrim(string,len)

* *** FORMAL PARAMETER DECLARATIONS ***

  INTEGER len
  CHARACTER*1 string(len)

* *** ABSTRACT ***

C* PURPOSE returns position of first nonblank character in string

C* AUDIT HISTORY

C  Densmore    20-Jan-83  AUTHOR

C* TYPE    character utility

C* FORMAL PARAMETERS

Cin    string  character string

Cin    len     length of string

C**
SUBROUTINE LWARN (INTVAR, STRING)

SOFTWARE DECLARATIONS

INTEGER INTVAR
CHARACTER STRING*1000

ABSTRACT

PURPOSE
causes a program warning and writes a diagnostic message
of "intvar: string".

C\$AUDIT HISTORY
MEMutchler 18 JAN 83 AUTHOR

C\$TYPE mnuong utility

C\$FORMAL PARAMETERS

Cin intvar integer to go into diagnostic message
Cin string char. string to go into diagnostic

C\$COMMON BLOCKS

Cin ioc i/o assignments

C\$LOCAL VARIABLES

C

nstring unelimited input string
SUBROUTINE mabort(text)

PARAMETER m=255
CHARACTER text*m

*** ABSTRACT ***

PURPOSE Print message to whatever units are appropriate, then abort

AUDIT HISTORY

Densmore 27-Oct-82 AUTHOR

TYPE Simple subroutine (no output)

FORMAL PARAMETERS

text delimited text string giving caller and an indication of the error that occurred.
C

MATCH2------------------------------------------

$CONTROL check=2

INTEGER FUNCTION match2(list,length,ientry)

C*  *** FORMAL PARAMETER DECLARATIONS ***

INTEGER length,ientry
INTEGER list(length)

C*  *** ABSTRACT ***

C#PURPOSE Makes first match of entry to list; returns index
C#AUDIT HISTORY
C  Densmore 08-Jun-83 AUTHOR
C#TYPE data checking utility
C#FORMAL PARAMETERS
Cin  list  list of integer*2 items
Cin  length length of list
Cin  ientry item to check against list
C#METHOD
C  Simple do-loop
C##
C MATCHC******************************************************************************
C*control check=2
INTEGER FUNCTION MATCHC(CHARAR,LENCH,LENARR,MATCH)
C*************************************************
C*** FORMAL PARAMETER DECLARATIONS ***
C*******************************************************************************************************
INTEGER LENCH,LENARR
CHARACTER*(LENCH) MATCH,CHARAR(LENARR)
C*******************************************************************************************************
C*** ABSTRACT ***
PURPOSE get index of match in charar array of character strings

AUDIT HISTORY
MEMutchler 28-may-83 AUTHOR
Oensmore 29-Jun-83 Moved to CUTILS

TYPE character utility

FORMAL PARAMETERS
Charar array of strings to match into
lench length of character strings
lenarr length of array
match string to match

METHOD
find match to array, match = position, 0 if not found

C MODCOR**********************************************************************
$CONTROL check=3
    INTEGER FUNCTION modcor(number,base)
    INTEGER number,base
C*PURPOSE Provide correct modulo function which is always
C  positive (0..base-1) instead of negative when number is.
C*AUTHOR Densmore
C**
SUBROUTINE moncom(comand, succes)

PARAMETER m=255, n=160

C

CHARACTER comand*m
logical succes

C

*** ABSTRACT ***

C

PURPOSE to execute the string "comand" as a monitor command.
C

AUDIT HISTORY
C

Densmore 27-Oct-82 AUTHOR
C

TYPE Simple subroutine
C

FORMAL PARAMETERS
C

comand delimited text string giving command to
C

be executed
C

succes logical variable indicating if the command
C

was successfully executed.
C

METHOD
C

Calls delim to determine the extent of the actual command
C

text, then places it in a buffer (maximum length of a command
C

is n-1 = 159 characters). A carriage return is appended
C

using the % construct valid for HP Fortran, then the command
C

intrinsic is called. Success is given by the Condition Code
C

contract .CC. and the ierr value, which is zero if okay.
C

**
SUBROUTINE mpdcod(param, arg, value, nparms, maxprm, lenprm)
character*255 param
character*(lenprm) arg(maxprm), value(maxprm)
integer maxprm, nparms

*** ABSTRACT ***

PURPOSE Takes a delimited string of parameters separated by commas and decodes it into individual parameters, also decoding individual parameters into left and right sides of any embedded equal signs.

HISTORY MSCarey 30-JAN-83 AUTHOR

PARAMETERS
Cin param delimited string to be decoded
Cout arg individual parameters, left side of equal sign
Cout value individual parameters, right side of equal sign
Cout nparms number of parameters found in decoding
Cin maxprm maximum number of parameters to decode
Cin lenprm maximum length of an arg or value after decoding

BLOCKS none

CALLER filopn, filcls

METHOD
C Force to uppercase and un-delimit. Then loop over number of commas found, searching also for equal signs. Blank the work array as search moves to the right.

LOCAL VARIABLES
1, find, lind, len, iword, icom, leq, lenv, lenp: char position or loop indexes
work storage for decode of character parameter
C

MRKDAY

$CONTROL check=3

INTEGER*4 FUNCTION mrkday(imonth,iday,iyear)

INTEGER imonth,iday,iyear

C*** ABSTRACT ***

CPURPOSE Marks the day; returns #dys since a given date (e.g. 31-December-1600). This routine is only required to return relative values; the above date need not be used, but subtracting two mark-days should yield the number of days between the two corresponding dates.

C

AUDIT HISTORY

Densmore 04-May-83 AUTHOR

TYPE Date utility

FORMAL PARAMETERS

Cin imonth integer representation of month [1..12]
Cin iday day [1..31]
Cin iyear year [1601..2399]

CALLER various

RELATED ROUTINES

Other (self-contained) functions may depend on the actual date used (31-Dec-1600) to return other information. For example, NUMDAY returns [1..7] (ie. [Sun..Sat]) given a date; this depends on the fact that 31-Dec-1600 was a Sunday. Such routines, if they exist on this library, are NUMDAY and DATEMK (which is the inverse of MRKAY).

METHOD

Checks that date is valid. Determines number of full year days. Determines number of full month days. Adds leap year days. Conditionally subtracts this leap year day. Conditionally subtracts Century non-leap year days. Conditionally adds the year 2000 leap year day.

LOCAL VARIABLES

idcum number of days in a year to that month

10-104
**GENERAL-PURPOSE UTILITIES**

```c
* CONTROL check=3
  INTEGER FUNCTION mchuc(clist,nchar,len,ientry)
* *** FORMAL PARAMETER DECLARATIONS ***
  INTEGER nchar,len
  CHARACTER*(nchar) clist(len), ientry
* *** ABSTRACT ***
C* PURPOSE Tries to match entry to a list element, but for ordered list
C* (use matchc for unordered list)
C*AUDIT HISTORY
C  Densmore  10-Jun-83 AUTHOR
C*TYPE character match utility
C*FORMAL PARAMETERS
C in  clist  character array of items to match against
C in  nchar  number of characters in each clist item
C in  len    number of clist items
C in  ientry item against which to match
C*CALLER utility
C*METHOD
C Binary search...returns 0 if no match exists
C**
```
GENERAL-PURPOSE UTILITIES

C  INTEGER FUNCTION ncfw(nwords)
C  *** FORMAL PARAMETER DECLARATIONS ***
C  INTEGER nwords, nchars
C  *** ABSTRACT ***
C#PURPOSE convert from word sizes to character sizes & vice versa
C#AUDIT HISTORY
C  Densmore  15-Dec-82 AUTHOR
C#TYPE character manipulation utility
C#FORMAL PARAMETERS
Cin  nwords number of words
Cin  nchars number of characters
C#MNEMONICS
C  N = number       C = Characters
C  F = From         W = Words
C  SW = ShortWords (*2) LW = LongWords (*8)
C#ENTRIES
C  ncfw (nwds4) := 4*nwds4  *4 words to characters
C  ncfsw(nwds2) := 2*nwds2  *2 words to characters
C  nclw(nwds8) := 8*nwds8  *8 words to characters
C  nwfc (nchars) := (nchars+3)/4 characters to *4 words
C  nswfc(nchars) := (nchars+1)/2 characters to *2 words
C  nlwfc(nchars) := (nchars+7)/8 characters to *8 words
C**
SUBROUTINE numask(number, nchar, cmask, cout)

INTEGER number, nchar
CHARACTER*(nchar) cmask, cout

*** ABSTRACT ***

PURPOSE Uses cmask as a mask over which significant digits in number are placed.

AUDIT HISTORY
Densmore 17-Mar-83 AUTHOR

TYPE manual assigner routine

FORMAL PARAMETERS

number the value to use in overwriting mask
nchar length of cmask and cout (result)

cmask the character mask
cout result

COMMON BLOCKS

c/out asgn assigner data block

METHOD

Examples using notation making numask look like char*(*) function:

'00234' = numask( 234,5,"00000") ! " " = numask( 0,4," ")

'13' = numask( 3,2,"!"") ! "###" = numask( 0,4,"#####")

"*****" = numask(100000,5,"abcde") ! "0-12" = numask(-12,4,"0000")

---

10-107
FUNCTION NUMSFX

TFUNCTION NUMSFX(number,ncaps)

CHARACTER*2 function numsfx(number,ncaps)

INTEGER number,ncaps

C$PURPOSE returns a number suffix, like "th" -- the "5th" item...

C$AUDIT HISTORY
C Densmore 30-Mar-83 AUTHOR
C$TYPE utility
C$FORMAL PARAMETERS
Cin number number for which suffix is to be provided
Cin ncaps set to 1 for lowercase, 2 for UPPERCASE
C$LOCAL VARIABLES
C tenprt "ten part" -- 10*(tens-digit) + (ones-digit)
C oneprt "one part" -- value of (ones-digit)
C$
**GENERAL-PURPOSE UTILITIES**

```plaintext
C NWDATE

$CONTROL check=3
INTEGER*4 FUNCTION nwdate(oldate, ndays)

C* *** FORMAL PARAMETER DECLARATIONS ***

INTEGER*4 oldate
INTEGER ndays

C* *** ABSTRACT ***

C$PURPOSE Returns the date ndays away from oldate
C$AUDIT HISTORY
C Densmore 31-May-83 AUTHOR
C$TYPE date utility
C$FORMAL PARAMETERS
C in oldate old date...in RELATE format (see /TODATE/)
C in ndays number of days...may be positive or negative
C$CALLER utility
C$METHOD
Converts to mm/dd/yy representation and uses datemk/mrkday

C* *** INCLUDES and LOCAL DECLARATIONS ***

C\$
```

10-109
GENERAL-PURPOSE UTILITIES

C NWDATU**************************************************************************
$CONTROL check=2
   INTEGER*4 FUNCTION nwdatu(ddate,nper,pertyp)
C*           *** FORMAL PARAMETER DECLARATIONS ***
   INTEGER*4 ddate
   INTEGER nper
   CHARACTER*4 pertyp
C*           *** ABSTRACT ***
C*PURPOSE Adds given number periods to date
C#AUDIT HISTORY
C Densmore  12-Oct-83 AUTHOR
C#TYPE Date utility
C#FORMAL PARAMETERS
Cin    ddate  a RELATE date (not necessarily clarified)
Cin    nper   number of periods to add (+ or -)
Cin    pertyp period type; may be 'DAY' 'WEE' 'MON' 'QUA' 'YEA'
C#METHOD
C looks for which type; performs addition; checks that the
C resulting date is still valid.
C#LOCAL VARIABLES
C    type   3-char version of pertyp
C    m,d,y  new date
C    inper  internal version of nper
C    movm,movy amounts to change month and year
C**
SUBROUTINE nwidat(om, od, oy, ndays, nm, nd, ny)

**FORMAL PARAMETER DECLARATIONS**

INTEGER om, od, oy, ndays, nm, nd, ny

**ABSTRACT**

NWEST Purpose Returns mm/dd/yy ndays away from input date

Densmore 31-May-83 AUTHOR

C SETUP

Ccin old month/day/year
Ccin ndays number of days separating old from new
Ccout new month/day/year output

METHOD

Uses mkday and datemk routines

C

10-111
C      PGINIT  *------------------------------------------------------------------
$CONTROL segment=pgprnt
   SUBROUTINE pginit
C*       *** FORMAL PARAMETER DECLARATIONS ***
C*       *** ABSTRACT ***
C**PURPOSE  INITIALizes the PaGe printing subsystem.
C**AUDIT HISTORY
C   MSCAREY   05-sep-83 AUTHOR
C**FORMAL PARAMETERS
C   none
C**COMMON BLOCKS
Cout   pgsys  page printing utility control info
C**CALLER utility
C**METHOD
C   Opens the buffer file for the printer and does an intitial
C   reset of the buffer to empty.
C**LOCAL VARIABLES
C   ok     fileopn flag
C**
**SUBROUTINE pgrset(unit, linlen, paglen, mode, fmode, quit, qchar)**

* *** FORMAL PARAMETER DECLARATIONS *** *

logical quit
integer unit, linlen, paglen, mode, fmode
character*1 qchar

* *** ABSTRACT *** *

C*PURPOSE  Reset Page printing utility.
C*AUDIT HISTORY
C  MSCarey  05-sep-83 AUTHOR
C*FORMAL PARAMETERS
Cin  unit  unit number to send output to now
Cin  linlen  length of output lines
Cin  paglen  number of lines on a page now
Cin  mode  operating mode (see /pgsys/)
Cin  fmode  page feed mode
Cin  quit  true if user wants pg to prompt for quit
Cin  qchar  character to accept as quit signal
C*COMMON BLOCKS
Cloc  pgsys  page printer globals
C*CALLER  various
C*METHOD
C  Set up the common block variables according to the arguments
C*LOCAL VARIABLES
C  none
C**
SUBROUTINE pgsend(header, nchdr, hlines, line, eoblock, eopage, quit)

* FORMAL PARAMETER DECLARATIONS *

%include pgsys

integer nchdr, hlines
character*(nchdr) header(hlines), line
logical eoblock, eopage, quit

** ABSTRACT **

PURPOSE Accepts a line of output and places it in an output buffer for eventual full-page printing. Optionally, controls the full-page output event and prompts the user for his desire to page next.

AUDIT HISTORY

MSCarey 05-sep-83 AUTHOR

FORMAL PARAMETERS

 Cin header page heading text
 Cin nchdr number of characters in a header line
 Cin hlines number of lines of heading text
 Cin line line to be output
 Cin eoblock true if line is end of a block which must fit on
 Cout eopage true if page is now full

COMMON BLOCKS

Cin pgsys page printing utility globals
 Cin ic global ic units

CALLER various

METHOD

 Send the text to the output buffer (a file).
 Jump to the code handling the current mode.
 Prompt and/or call pgwr to do the output and/or set eopage.

LOCAL VARIABLES

 prompt prompt string
GENRAL-PURPOSE UTILITIES

C PGWRT ****************************************************
C* \$CONTROL segment=pgprnt
C* SUBROUTINE pgwrit(header,nchdr,hlines,page)
C* *** FORMAL PARAMETER DECLARATIONS ***
C* ** include pgsys
C* integer nchdr,hlines
C* character header*(nchdr)(hlines)
C* logical page
C* *** ABSTRACT ***
C* PURPOSE    Writes out a page or line from the buffer and does
C* some buffer housekeeping
C* AUDIT HISTORY
C*  MScarey      05-sep-83  AUTHOR
C* FORMAL PARAMETERS
C* Cin  header  text of page header
C* Cin  nchdr   number of chars in header line
C* Cin  hlines  number of lines in header
C* Cin  page    true if in header to be written on each call
C* C         false, header written only on first call
C* COMMON BLOCKS
C* Cin  pgsys   page utility globals
C* CALLER various, mostly pgsend
C* METHOD
C*  Write from pgtop to pglast; reset pgtop to record after pglast
C* or to 0 if this is > pglin. Write header according to mode.
C* LOCAL VARIABLES
C* Cin  line    line buffer for transfer from buffer file to output
C* Cin  device
C*
PLURAL

$CONTROL check=3
CHARACTER*1 FUNCTION plural(number,case)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER number,case

*** ABSTRACT ***

PURPOSE Returns "S" if number <> 1, returns blank if =1

AUDIT HISTORY

Densmore 21-Apr-83 AUTHOR

TYPE Format utility

FORMAL PARAMETERS

number value

case 1=lower case, 2=upper case
SUBROUTINE prthlp(name,found,in,out)

C* FORMAL PARAMETER DECLARATIONS ***

CHARACTER*8 name
LOGICAL found
INTEGER in,out

C* ABSTRACT ***
C* PURPOSE prints the text in file UNIT associated with category NAME
C* AUDIT HISTORY
C Densmore 29-Mar-83 AUTHOR
C* TYPE utility for use with assigner
C* FORMAL PARAMETERS

Cin name char*8 name of category
Cout found .TRUE. if category name was found and printed
Cin in unit number for file on which text is located
Cin out display unit number

C* METHOD
C The file associated with unit number IN is expected to have leader
C lines associated with each category it contains. These leader
C lines are of the form:
C %BEGIN CAT-NAME
C where the first seven characters are "%BEGIN ", and the next eight
C (8) characters are the category name. Remaining characters on
C these leader lines are ignored and may be used for comments.
C When the names (ignoring case) match, the corresponding text
C is printed until another %BEGIN, or End-Of-File, is encountered.
C If the text found contains lines whose first seven characters are
C "%BREAK ", at each such point the process is halted and the file
C is queried for a carriage return to continue, except when
C OUT is not file 5.
C**
SUBROUTINE putmem(id,length,source,start)

integer id,length,start
logical source(1)

C#PURPOSE Swaps data in source into extended memory area.
C#AUDIT HISTORY
C MSCarey 11-aug-83 AUTHOR
C#FORMAL PARAMETERS
Cin id operating system id code for area
Cin length number of *2 words to swap
Cin source source array for words
Cin start starting position in extended mem to send to
C#COMMON BLOCKS
C none
C#CALLER various
C#METHOD
C Calls dmovout.
C##

$CONTROL check=2,segment=seg'

C PUTMEM ****************************************************

** FORMAL PARAMETER DECLARATIONS **

*** ABSTRACT ***

C**PURPOSE Swaps data in source into extended memory area.
C**AUDIT HISTORY
C MSCarey 11-aug-83 AUTHOR
C**FORMAL PARAMETERS
Cin id operating system id code for area
Cin length number of *2 words to swap
Cin source source array for words
Cin start starting position in extended mem to send to
C**COMMON BLOCKS
C none
C**CALLER various
C**METHOD
C Calls dmovout.
SUBROUTINE qsortc(a,n,c,s,l)

FORMAL PARAMETER DECLARATIONS

INTEGER n,c,s,l

CHARACTER*(c) a(n)

ABSTRACT

Quick SORTing method for Character arrays; uses HEAPSORT

HISTORY

Oensmore 17-May-83 AUTHOR

TYPE

Sort utility

FORMAL PARAMETERS

a array to be sorted (in place)

n number of elements in the array a

c number of characters in each element of a

s starting character of the key for each element

l length of the substring comprising the key

METHOD

Uses HEAPSORT...See Knuth Volume 3 pp. 146-147

Heapsort is guaranteed to be an N*Log(N) algorithm even in worst cases. Records considered are those between i and j at any one point in the algorithm. If left>1, then a "Heap" is being formed, such that a(floor(j/2))[s:l] > a(j)[s:l] for all j; j such that 1 < floor(j/2) < j <= n. Once left=1, a(1) has the largest remaining key, and in this manner the records are sifted into a sorted order, in place.

Oensmore 17-May-83 AUTHOR

C#TYPE Sort utility

C#FORMAL PARAMETERS

Cin/out a array to be sorted (in place)

Cin n number of elements in the array a

Cin c number of characters in each element of a

Cin s starting character of the key for each element

Cin l length of the substring comprising the key

C#METHOD

C Uses HEAPSORT...See Knuth Volume 3 pp. 146-147

C Heapsort is guaranteed to be an N*Log(N) algorithm even in worst cases. Records considered are those between i and j at any one point in the algorithm. If left>1, then a "Heap" is being formed, such that a(floor(j/2))[s:l] > a(j)[s:l] for all j; j such that 1 < floor(j/2) < j <= n. Once left=1, a(1) has the largest remaining key, and in this manner the records are sifted into a sorted order, in place.

C#
C QUERY

$CONTROL check=2
LOGICAL FUNCTION query(text)

*** FORMAL PARAMETER DECLARATIONS ***

CHARACTER*255 text

*** ABSTRACT ***

C PURPOSE prints query and calls yesno

C AUDIT HISTORY

C Densmore 10-Feb-83 AUTHOR

C TYPE I/O utility

C FORMAL PARAMETERS

C in text query text -- string is delimited

C$
$CONTROL check=3

REAL FUNCTION ranf(iseq)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER iseq
PARAMETER lenl=11
INTEGER*4 jseed(lenl),jseed1

*** ABSTRACT ***

PURPOSE Generates uniform random numbers over the range (0,1)

AUDIT HISTORY
Densmore 16-Jun-83 AUTHOR

TYPE statistics utility

FORMAL PARAMETERS

iseq sequence number (0..lenl-1 -- if not in this range
then the 0 sequence is used)

jseed the seeds for each sequence

jseed1 a single seed which is used to init all seeds

METHOD
ranf returns a uniform random number on sequence iseq over (0,1)
ranset initializes all lenl sequences independently
ranstl initializes all seeds from a single input seed
ranget retrieves all lenl seeds for storage
ALL entries are functions because ranf is; only ranf uses the
function return.

10-121
REAL FUNCTION rdate(dum)
    integer dum

*** ABSTRACT ***

Purpose: Returns the current date as YYMMDD in a real variable.

Author: MSCarey 28-feb-83

Formal Parameters:
    dum dummy

Common Blocks:
    none

Caller: various

Method: Calls calendar intrinsic, converts to month-day-year, and packs output variable.

Variables:
    date date as returned by intrinsic
    year year
    days day in year
    td a running total of days
    month month of year
    dayom day of month

10-122
FUNCTION rdstr(realdt)
  real realdt

*** ABSTRACT ***

C Purpose: Relate real Date Format to STRING format conversion.
C Converts dates stored in real variables as YYMMDD to a string
C format of "MM/DD/YY".

C* Audit History

MSCarey 26-feb-83 AUTHOR

C Formal Parameters

Cin realdt date stored in RELATE real variable format

C Common Blocks

C none

C Caller

various

C Method

Break out the three 2-integer fields and convert them to strings.

C Local Variables

C string character buffer for date

C
SUBROUTINE RDLN (IUNIT, LINE, EOF)

C* INCLUDE INCPAR
C* INCLUDE I0C
 INTEGER IUNIT
 LOGICAL EOF
 CHARACTER LINE*, LLINE, BUFFER*, LLINE

C* PURPOSE read a line from IUNIT, without uppercasing
C* AUDIT HISTORY
C MEMutchler 17 JAN 83 AUTHOR
C MEMutchler  8 FEB 83 TESTER (program treadl)
C MSCarey  10 FEB 83 Reads 80 from S, else 72 col
C MSCarey  1 MAR 83 Echoes input if cfecho true
C MSCAREY  5 Mar 83 Handles com file termination
C*TYPE nnurun utility
C*FORMAL PARAMETERS
 Cin iunit  unit number from which to read
 COUT line  line that was read
 COUT eof   true iff eof was read
C*COMMON BLOCKS
 Cin incpar global parameter statement
 Cin ccomfl holds command file info.
C*METHOD An unformatted read is done from unit =
 C iunit.  EOF = false unless an end of file is read
 C in which case EOF = true.  If command file building
 C is in use, LINE is echoed to unit = icomfile.
 C  JUST LIKE READLN WITHOUT UPPERC
C*LOCAL VARIABLES none
C**
SUBROUTINE RDLNC (IUNIT, LINE, EOF)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER IUNIT

CHARACTER LINE*LLINE

*** ABSTRACT ***

PURPOSE read from file IN and keep track of lines read

without uppercasing, especially for reading text files

AUDIT HISTORY

MEMutchler 17 JAN 83 AUTHOR

MEMutchler 8 FEB 83 TESTER (program treads)

TYPE mnugen utility

FORMAL PARAMETERS

Cin  IUNIT file number from which to read

Cout LINE input line read

Cout EOF true iff EOF read from IUNIT

COMMON BLOCKS

Incpar global parameter statements

Read holds LINE

METHOD. An unformatted read is done from unit =

IUNIT. EOF = false unless an end of file is read

in which case EOF = true. If command file building

is in use, LINE is echoed to unit = icomfile.

Icount is incremented.

LOCAL VARIABLES

Recch 'I' recognition character for comment card

***
PROGRAM ROLNCU

**CONTROL SEGMENT**

SUBROUTINE ROLNCU (IUNIT,LINE,EOF)

**FORMAL PARAMETER DECLARATIONS**

INTEGER IUNIT
CHARACTER LINE, LLINE

**ABSTRACT**

**PURPOSE**
read from file IN and keep track of lines read

**HISTORY**
MEMutchler 17 JAN 83 AUTHOR
MEMutchler 8 FEB 83 TESTER (Program thread)

**TYPE**
 munogen utility

**PARAMETERS**

Cin file number from which to read
Cout input line read
Cout EOF true iff EOF read from iunit

**COMMON BLOCKS**

Cin reads holds line

**METHOD**
An unformatted read is done from unit =
Cin. EOF = false unless an end of file is read
in which case EOF = true. If command file building
is in use, LINE is echoed to unit = icomfile.
Lcount is incremented.

**LOCAL VARIABLES**

C recch 'r' recognition character for comment card

**
SUBROUTINE READLN (IUNIT, LINE, EOF)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER IUNIT
LOGICAL EOF
CHARACTER LINE*LLINE, BUFFER*LLINE

*** ABSTRACT ***

C PURPOSE: read a line from IUNIT

C AUDIT HISTORY
C
C MEMutchler 17 JAN 83 AUTHOR
C MEMutchler 8 FEB 83 TESTER (program treadl)
C MSCarey 10 FEB 83 Reads 80 from 5, else 72 col
C MSCarey 1 MAR 83 Echoes input if cfecho true
C MSCAREY 5 MAR 83 Handles com file termination

C TYPE: mnurun utility

C FORMAL PARAMETERS

C IN
IUNIT unit number from which to read

C OUT
LINE line that was read
EOF true iff EOF was read

C COMMON BLOCKS

C IN
INCPAR global parameter statement
COMCFL holds command file info.

C METHOD
An unformatted read is done from unit = IUNIT. EOF = false unless an end of file is read in which case EOF = true. If command file building is in use, LINE is echoed to unit = icomfile.

C LOCAL VARIABLES
none

C
C RTRIM**************************************************************************
$CONTROL check=2
   INTEGER FUNCTION rtrim(string,length)
C* *** FORMAL PARAMETER DECLARATIONS ***
   INTEGER length
   CHARACTER*1 string(length)
C* *** ABSTRACT ***
C#PURPOSE Finds length of string, NOT including any trailing blanks.
C#AUDIT HISTORY
C  Densmore  28-Oct-82 AUTHOR
C#TYPE Simple function
C#FORMAL PARAMETERS
Cin  string  the character string
Cin  length  length of string
Cfunction rtrim  length of string without trailing blanks
C#METHOD
C  Uses HP Fortran substring operator to locate last nonblank.
C  If all blank, rtrim is returned zero.
C**
SUBROUTINE SCLEAR

*** FORMAL PARAMETER DECLARATIONS ***

*** ABSTRACT ***

PURPOSE clear terminal screen

AUDIT HISTORY

MEMutchler 11-mar-83 AUTHOR

TYPE mnurun utility

FORMAL PARAMETERS none

COMMON BLOCKS

Cin envirn holds clear screen control characters

Cin ioc i/o file assignments

CALLER display menu routines

METHOD write control characters to unit iout

**
C       SETCCL******************************************************************************

*CONTROL SEGMENT=devctrl
SUBROUTINE SETCCL

C*                                          *** FORMAL PARAMETER DECLARATIONS ***
C*                                          *** ABSTRACT ***

C#PURPOSE Set clear screen control characters
C#AUDIT HISTORY
C#TYPE murun utility
C#FORMAL PARAMETERS none
C#COMMON BLOCKS
Cin   pvalue  holds runtime parameter values
Cin   pvdecl holds declarations for parameter names
Cin   pvequiv equivalence statements between pvdecl and pvalue
Cio   envirn holds info about runtime environment
C#CALLER sclear
C#METHOD
C  set correct characters according to terminal type
C#LOCAL VARIABLES
C  none
C#

10-130
GENERAL-PURPOSE UTILITIES

C SETTTY****************************************************
$CONTROL SEGMENT=devctrl
    SUBROUTINE SETTTY
C*    *** FORMAL PARAMETER DECLARATIONS ***
C*    *** ABSTRACT ***
C*PURPOSE Determine user's terminal type so that control
C  characters can be set accordingly.
C*AUDIT HISTORY
C  MEMutchler   12-MAR-83 AUTHOR
C*TYPE   mnurun utility
C*FORMAL PARAMETERS  none
C*COMMON BLOCKS
Cin   pvalue holds runtime parameter values
Cin   pvdecl holds declarations for parameter names
Cin   pveqiv equivalence statements between pvdecl and pvalue
C*CALLER  inimnu
C*METHOD
C  determine terminal type by checking terminal line number. If
C  it is 51, the terminal is an hp, if not set it to the most
C  commonly used terminal at OSA
C*LOCAL VARIABLES
C    filnum  file number assigned to logical unit 6
C    linum  terminal line number
C**

10-131
GENERAL-PURPOSE UTILITIES

C SLPRNT

$CONTROL check=3

SUBROUTINE slprnt(nprnt,vprnt)

C* *** FORMAL PARAMETER DECLARATIONS ***

INTEGER nptnr
LOGICAL vprnt

C* *** ABSTRACT ***

C* PURPOSE alters specific LPRNTS values
C* AUDIT HISTORY
C Densmore 10-Feb-83 AUTHOR
C* TYPE miscellaneous utility
C* FORMAL PARAMETERS
C1 nprnt lprnts index
C1 vprnt lprnts value (true or false)
C* COMMON BLOCKS
Cout lprnts diagnostic flags
C**
**SUBROUTINE stopcf**

---

**ABSTRACT**

Takes care of housekeeping on eof in current command file.

**HISTORY**

MSCarey 27-FEB-83 AUTHOR

**PARAMETERS**

- `comcfl` command file usage status info
- `ioc` io unit assignments

**METHOD**

Reduces execution nesting level by one. If it reaches zero, sets *inuse* to false. Resets io unit numbers.

**LOCAL VARIABLES**

- `none`
GENERAL-PURPOSE UTILITIES

C STRN..............................................................

*CONTROL SEGMENT-MENU
CHARACTER*72 FUNCTION STRN( INUM,LEN)
%INCLUDE INCPAR
   INTEGER LEN, INUM
C*    *** FORMAL PARAMETER DECLARATIONS ***
C*        CHARACTER STRING*LLINE
C*    *** ABSTRACT ***
C*PURPOSE like intrinsic function str, returns len necessary
C*AUDIT HISTORY
C MEMutchler                  1 FEB 83 AUTHOR

10-134
SUBROUTINE trecol(list,nchar,len,unit)
*** FORMAL PARAMETER DECLARATIONS ***
INTEGER nchar,len,unit
CHARACTER*(nchar) list(len)
*** ABSTRACT ***
C* PURPOSE Prints LIST in three columns on unit UNIT
C* AUDIT HISTORY
C Densmore 10-Jun-83 AUTHOR
C*TYPE I/O utility
C*FORMAL PARAMETERS
Cin list list of strings to print
Cin nchar length of each string
Cin len number of strings
Cin unit Logical Unit Number on which to print strings
C*METHOD simple write statement
C**
SUBROUTINE ttyini

*** ABSTRACT ***

PURPOSE initializes /tty/

AUTHOR
Densmore  24-Mar-83

COMMON BLOCKS
Cin/out tty terminal parameters

10-136
SUBROUTINE upperc(text,n)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER n
CHARACTER*l text(n)

*** ABSTRACT ***

C* PURPOSE Convert all lower (upper) case characters to upper (lower)

C* AUDIT HISTORY
C Densmore 27-Oct-82 AUTHOR
C* TYPE Inout Subroutine

C* FORMAL PARAMETERS
Cin/out text text string to be modified
Cin n length (characters) of text string

C* METHOD
C Uses HP's byte addressing construct to move through the
C string and locate any in the appropriate range. Since
C in ASCII the difference between any lowercase letter and
C the corresponding uppercase letter is a constant value
C (decimal 32), this value is merely added or subtracted
C from the integer representation of each character to be
C altered to opposite case.

C**
SUBROUTINE uread(unit, plist, length)

C** FORMAL PARAMETER DECLARATIONS ***

INTEGER unit, length, plist(length)

C** ABSTRACT ***

C#PURPOSE Allows record structure unformatted reads and writes

C#AUDIT HISTORY

C Densmore 04-Apr-83 AUTHOR

C#TYPE I/O Utility

C#FORMAL PARAMETERS

Cin  unit  logical unit number for transfer file

Cin/out  plist  parameter list of 4 words (in=write, out=read)

Cin  length  length of plist

C##
SUBROUTINE usrinf(uname,ugroup,uacct,uhome)
character*8 uname,ugroup,uacct,uhome

*** ABSTRACT ***

PURPOSE retrieves user name/directory info

AUDIT HISTORY
Densmore 13-jan-83 AUTHOR

FORMAL PARAMETERS

uname user name
ugroup user's log-on group
uacct user's log-on account
uhome user's home group, if any

COMMON BLOCKS

CALLER various

METHOD
See fortran manual appendix A for discussion of calls to intrinsics
See MPE Intrinsics manual page 2-195 for "WHO"

LOCAL VARIABLES

none


SUBROUTINE vsumni(n, head, tail, result)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER n
INTEGER head(n), tail(n), result(n)

*** ABSTRACT ***

PURPOSE: vector sum/difference for integers

AUDIT HISTORY

CSTYPE: manual assigner routine

FORMAL PARAMETERS

Cin  n  length of vectors
Cin  head  first vector
Cin  tail  second vector
Cout result  head+tail or head-tail
SUBROUTINE xmit (length, source, target)
INTEGER length
INTEGER*4 source(l), target(l)
"B" entry allows right-shifts by "DO"ing backwards.

C#PURPOSE fills target array with source via copy
C#HISTORY Densmore 28-Oct-82
C#FORMAL PARAMETERS
Cin   length size of move...if <0, only source(1) is used
Cin   source source array
Cout  target target array
C SUBROUTINE xmit2(length, source, target)
   INTEGER length
   INTEGER*2 source(1), target(1)
C PURPOSE fills target array with source via copy
C "B" entry allows right-shifts by "DO"ing backwards.
C AUDIT HISTORY Densmore 28-Oct-82
C FORMAL PARAMETERS
Cin    length size of move...if <0, only source(1) is used
Cin    source source array
Cout   target target array
C=}
SUBROUTINE xmit4(length, source, target)
INTEGER*4 length
INTEGER*4 source(1), target(1)

PURPOSE
fills target array with source via copy

'B' entry allows right-shifts by 'DOing backwards.

AUDIT HISTORY  Densmore  28-Oct-82

FORMAL PARAMETERS

length  size of move...if <0, only source(1) is used
source  source array
target  target array
SUBROUTINE xmitc(length, source, target)
INTEGER length
CHARACTER source(length), target(length)
C#PURPOSE fills target array with source via copy
C "B" entry allows right-shifts by "DO"ing backwards.
C#AUDIT HISTORY Densmore 28-Oct-82
C#FORMAL PARAMETERS
Cin length size of move...if <0, only source(1) is used
Cin source source array
Cout target target array
C##
logical function yesno(in,iout)
   integer in,iout

*** ABSTRACT ***

PURPOSE Prompts for an answer, true if yes.

AUDIT HISTORY
MSCarey 03-feb-83 AUTHOR

TYPE I/O utility

FORMAL PARAMETERS
   in unit number to read from
   in unit number to write to

COMMON BLOCKS

CALLER various

METHOD
Prompts, accepting only "y","n","yes", or "no"

LOCAL VARIABLES
   answr input buffer
   answer user answer
   len string size

10-145
SUBROUTINE zabort

*** ABSTRACT ***
PURPOSE Aborts when an error occurs...often called by MABORT.

AUDIT HISTORY
Densmore 27-Oct-82 AUTHOR

TYPE Simple subroutine
10.2 FORTRAN UTILITIES FOR DBMS USAGE

The routines in the Data Base management system Interface library (DBIF; source in dbifa.src, dbifdm.src, dbifl.src, dbifrv.src; principal include files strngs.incl, cursrs.incl; object code in dbif.obj) were created for two reasons:

1) A move of ALIAS software to a computer other than the HP 3000 was declared to be a possibility at the outset of ALIAS development. Given that RELATE runs only on the HP 3000, and that ALIAS routines would be making very heavy use of RELATE, it seemed prudent to buffer all requests for DBMS services through a set of interface routines. At conversion it should be possible to change only the internals of the interface to work with a new DBMS, making it possible to avoid major changes to the applications programs.

2) The RELATE Host Language Interface routines are rather difficult and finicky to work with directly. A more programmer-friendly means of accessing the data base was desired.

10.2.1 DBIF Organization

Although the DBIF can be used to issue any RELATE, CREATE, or GRAF command, it is primarily designed to make use of the routines of RELATE's Host Language Interface. These routines provide a record-level method of data base access (as opposed to the set-level method of interactive RELATE); that is, operations are performed on data base files one tuple at a time. In addition to the obvious read, add, delete, and update capabilities, the HLI also provides a "point" routine which allows the programmer to jump to the location on an index whose field values match the values of the target he specifies. Also, a query routine will return information about relations and the state of the DBMS, and an error routine can be used to learn more about errors after they have occurred.

The DBIF can be divided into high-level and low-level routines, the high-level routines being those called by programmers. An annotated listing of the high-level routines is presented in Table 10-5. Low level routines are listed in Table
Table 10-5. DBIFA High-Level Routines

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKWPRV</td>
<td>Security utility which module authors can call during their initialization code to see if the user is going to have write access to all the relations he will require. If not, graceful termination can be engineered. This routine duplicates the logic of rvscen, which is called by DBIF routines about to do a DB write to check privilege. Rvscen invokes a ZABORT if privileges are insufficient, which is why explicit testing is nice.</td>
</tr>
<tr>
<td>RCINIT</td>
<td>Initializes DBIF. Must be called before any other DBIF routine.</td>
</tr>
<tr>
<td>RELCOM</td>
<td>Call this to execute any RELATE, CREATE, or GRAF command programmatically. Allows simulation of interactive use of RELATE.</td>
</tr>
<tr>
<td>RTPADD</td>
<td>Adds a single record to the path specified.</td>
</tr>
<tr>
<td>RTPCAL</td>
<td>Requires a buffer of target values for the fields on the current index. Performs a &quot;point&quot; operation which locates the record with fields matching these values; then returns the contents of the record into a second data buffer. Much more efficient than SELECT for many types of searching. Returns no data if point fails.</td>
</tr>
<tr>
<td>RTPDEL</td>
<td>Deletes the current record on the path specified.</td>
</tr>
<tr>
<td>RTPKIL</td>
<td>Similar to rtpcal, except for record deletion. Finds the record matching the specified index value via a point, then deletes it.</td>
</tr>
<tr>
<td>RTPNEW</td>
<td>Attempts to add a record to the current path; this routine is an integer function which returns a status code value, where 0=success, 1=failure due to unary key violation, and 2=failure due to relation full. It's a good idea to use this rather than rtpadd and to place error handling logic in your code.</td>
</tr>
<tr>
<td>RTPNFD</td>
<td>Like rtpcal except expects NOT to find the record pointed to. Reads and returns whatever record the point left the record pointer at.</td>
</tr>
<tr>
<td>RTPNXT</td>
<td>Reads and returns the next record on the current path.</td>
</tr>
<tr>
<td>RTPREP</td>
<td>Like rtpcal except for update. Points to the record specified, then updates it with the specified values.</td>
</tr>
</tbody>
</table>
Table 10-5. DBIFA High-Level Routines

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTPUPD</td>
<td>Updates the current record on the specified path with given values.</td>
</tr>
<tr>
<td>RVCLCOS</td>
<td>Closes a path and de-allocates its cursor.</td>
</tr>
<tr>
<td>RVCFIL</td>
<td>Creates a new relation with the specified structure, returning the index of its cursor/path.</td>
</tr>
<tr>
<td>RVCKIL</td>
<td>Deletes the relation open on the specified path.</td>
</tr>
<tr>
<td>RVCPTH</td>
<td>Creates a relation and opens it with an alternative path name. Very similar to rvcfil.</td>
</tr>
<tr>
<td>RVCREL</td>
<td>Opens a relation, returning the index of its cursor/path.</td>
</tr>
<tr>
<td>RVCRWD</td>
<td>Rewinds the specified path.</td>
</tr>
<tr>
<td>RVCSLC</td>
<td>Does a SELECT and returns the index of its cursor/path. The files the select draws on must already be open on other cursors. Their names must be included as part of normal SELECT syntax somewhere in the field list or WHERE clause (e.g. field list of &quot;+filea.fld1, filea.fld2, fileb.fld9+&quot;).</td>
</tr>
<tr>
<td>RVCSRT</td>
<td>Opens a relation and sets to a specified index. Note that if points will be desired using only a subset of the index fields (e.g. only SCENARIO of an index on SCENARIO,CLASS) then the last desired field should be followed by a &quot;</td>
</tr>
<tr>
<td>RVCSST</td>
<td>Opens a relation and sets to a specified index, using an alternative path name. Like rvcsrt.</td>
</tr>
<tr>
<td>RVCSYN</td>
<td>Opens a relation on an alternative path name. Similar to rvcrel. Note that the alternative path name routines are seldom useful. Since all files are opened on separate cursors, and the same file may be open more than once under the same name if the opens are done on separate cursors, then there is no particular reason to use path synonyms.</td>
</tr>
</tbody>
</table>

10-149
Section 10.2.7 (subroutine abstracts) gives the detailed calling requirements for each routine.

10.2.2 Using the Routines

The reader will notice that the routines named in Table 10-5 can be divided into four categories by their names:

1) rcinit
2) relcom
3) All routines whose names begin with "rvc".
4) All routines whose names begin with "rtp".

Rcinit initializes the DBIF. Relcom can be used to give any interactive RELATE, CREATE, or GRAF command programmatically. The "rvc" (Relate Virtual Cursor) routines are used to set up a retrieval path; they open and close files, choose indexes, and give selects. The "rtp" (Relate TuPle) routines find and/or manipulate individual records in data relations.

A typical calling sequence would include rcinit, rvcsrt to open a relation and set to a particular index, and a combination of rtpcal and rtpnxt calls to jump to a location on the index and retrieve records starting there.

There are four basic choices for setting up a retrieval path:

1) The equivalent of a regular OPEN FILE (rvcrel).
2) The equivalent of an OPEN FILE followed by a SET INDEX (rvcsrt).
3) The equivalent of a SELECT (rvcs1c); this presumes that the relations the SELECT wants have already been opened by other "rvc" calls.
4) The equivalent of a CREATE FILE (rvcfil).
<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCGIDX</td>
<td>Does a SET INDEX and related data structure set-up.</td>
</tr>
<tr>
<td>DCGTUP</td>
<td>Does a record update for the current tuple.</td>
</tr>
<tr>
<td>DCINIT</td>
<td>Initializes the cursor and string chain subsystems.</td>
</tr>
<tr>
<td>DCKCRS</td>
<td>Error management routine which prints status information which can be extracted from the cursor the problem occurred on and from the DBIF data structure.</td>
</tr>
<tr>
<td>DCKERR</td>
<td>Checks to see if an error happened on the last call involving the given cursor, and causes the associated RELATE error message to be printed to the terminal if one did.</td>
</tr>
<tr>
<td>DCSLCT</td>
<td>CLOSES all files open on a cursor a select was given on and releases the cursor.</td>
</tr>
<tr>
<td>DCSPTH</td>
<td>CLOSES the file open on a regular (non-select) cursor and releases the cursor.</td>
</tr>
<tr>
<td>DELCRS</td>
<td>Routine which actually releases cursors, both in DBIF data structure and in RELATE son process.</td>
</tr>
<tr>
<td>DELIDX</td>
<td>Deletes an index from the current relation. This routine is non-function (i.e. it will abort) as long as the current convention of opening all relations with MODE-SHARED is in place.</td>
</tr>
<tr>
<td>DELREL</td>
<td>Deletes the relation open on the given cursor.</td>
</tr>
<tr>
<td>DELTUP</td>
<td>Deletes the current record in the relation open on the given cursor.</td>
</tr>
<tr>
<td>DMKCRS</td>
<td>Allocates a cursor in the DBIF (/cursrs/) data structure and call rdbinit to initialize it.</td>
</tr>
<tr>
<td>DMKIDX</td>
<td>Attempts to create a new index. Will always abort at present since all relations are opened with MODE-SHARED.</td>
</tr>
<tr>
<td>DMKREL</td>
<td>Creates a new relation.</td>
</tr>
<tr>
<td>DMKTUP</td>
<td>Adds a record to the specified path.</td>
</tr>
<tr>
<td>DOPPTH</td>
<td>Opens the given relation on the given cursor.</td>
</tr>
<tr>
<td>DPCMD</td>
<td>Diagnostic print utilities.</td>
</tr>
<tr>
<td>ROUTINE</td>
<td>PURPOSE</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>DPCMD1</td>
<td></td>
</tr>
<tr>
<td>DREWND</td>
<td>Rewinds the file open on the given cursor.</td>
</tr>
<tr>
<td>DSOPEN</td>
<td>Service routine for rvcslc, opens all the files requested as part of the select on the select's cursor. Note that since only the path parts of the file names are specified in the select syntax the group names must be extracted from the DBIF data structure and by rdbinfo calls.</td>
</tr>
<tr>
<td>DSELECT</td>
<td>Does a select.</td>
</tr>
<tr>
<td>DTCALC</td>
<td>Does a point on the given cursor.</td>
</tr>
<tr>
<td>DTNEXT</td>
<td>Reads a record from the given cursor's path.</td>
</tr>
<tr>
<td>LENIDX</td>
<td>Figures out the number of words in the specified index and stores it.</td>
</tr>
<tr>
<td>RCKPRV</td>
<td>Checks to see if the user has write privileges on the given file. Both sysusr.sysro and scenario system checks implicitly involved. Called before each DBIF write operation as a last-ditch defense.</td>
</tr>
<tr>
<td>RSTIDX</td>
<td>Execute for index-setting.</td>
</tr>
<tr>
<td>RVSCEN</td>
<td>Security check and flag-setting routine called whenever a path is set up by one of the &quot;rvc&quot; routines. Also sets scenario key field value in /scenar/.</td>
</tr>
<tr>
<td>SNRLSN</td>
<td>Utilities used to access the scenario system's extra data segment. Truly part of the scenario system, as is rvscen, but present here as part of scenario system &quot;presence&quot; in DBIF.</td>
</tr>
<tr>
<td>SNRLNM</td>
<td></td>
</tr>
</tbody>
</table>
The remaining utilities perform similar actions but using alternative path names (rvcsyn, rvcsts, and rvcpth respectively), close files, delete files, and "rewind" record pointers to top-of-file.

Use of the relcom routine should be avoided except to give commands not provided for in the other utilities.

Developers are likely to find the rtpcal routine particularly useful. Functionally similar to a combination of a BUILDER RECORD POINT and RECORD READ given in sequence, this routine can locate and return the contents of a particular record in a relation (by key/index value) much faster than an equivalent select can. Benchmarks have shown that rtpcal requires approximately 250 milliseconds (single-user) regardless of the size of the relation or the number of fields in the index.

Note that if it is necessary for the implicit point to operate on only a subset of an index (e.g. you want to point only to YARD on an index of YARD, DATADATE, ENTRY_DATE) this can be done by specifying the index with a "|" rather than a "," following the last field of point-interest in the "rvc" call (e.g. YARD|DATADATE, ENTRY_DATE)

All of the routines require a single-word integer argument called "cursor". More about this in the next section.

In addition to a cursor index, the "rvc" routines often require one or more delimited text strings which specify the name of the relation to be opened, fields in the index to be set to, clauses to include in the select statement, etc. The only unusual requirement is by rvcslc, which requires that the name of each relation to selection is to draw from be mentioned at least once in the field list or by clause argument, in the form "relation.fieldname,relation.fieldname,...".

10-153
The "rtp" routines will typically require at least one data buffer as an argument, and perhaps a delimited list of fields to be returned, updated, etc. It is VERY IMPORTANT that the DATA BUFFER BE WORD ALIGNED, i.e. that it not be a character string. If the data is of type character, equivalence the character variable to an integer array and pass the integer form as an argument. RELATE will abort nastily if it received a non-word-aligned buffer.

To be most usable, data buffers should consist of a series of variables, one per corresponding field in the relation (or on the index), and of identical length and type in comparison to the fields. Thus a buffer for the fields SCENARIO, CLASS, HULL would consist of a character*12,character*10,integer*2 series of variables, all next to each other in process data memory. The best way to ensure that the variables are actually sequential in memory is to declare them sequentially in the same common block (character and numeric data may be mixed in HP FORTRAN common blocks). Equivalences may also be used, but require more coding. Note in the example given that the "scenario" variable needs to be equivalenced to an integer array to word-align the common block.

When using the rtpcal routine, it may seem that there should be two field lists as arguments, to accompany the two data buffers required: one to specify the fields in the target to be pointed to, and one to specify the fields to be returned. The target field list is implicit, being defined by the current index. Note that rtpcal will ALWAYS return notfnd=.true. if the target (key) data buffer is not of the same length as the index, or if values are improperly positioned within the buffer. Note especially that since RELATE left-justifies strings, they should be left-justified in the target buffer (but non-justified strings can be placed in the relation using the DBIF).
10.2.3 Cursors and the DBIF Data Structure

The DBIF manipulates three global data structures. The first is a string buffer, managed by the CHN general purpose utilities, which is used for handling field lists, file names, etc. This buffer is of no particular interest to users; the chain strategy was used since field lists can exceed 255 characters and in order to conserve memory.

The second data structure is the cursen array in the /scenar/ common block. When a file is opened using any of the "rvc" routines, the proper scenario key field value for that file is retrieved and placed in the location in cursen indexed by the cursor index to be returned to the "rvc" routine's caller. The corresponding location in the wrtprv array is also set. This activity actually is the portion of the scenario system which resides in the DBIF.

The "cursor" data structure is the third. Remember from Section 8.4 that HLI routines require that a 50-word integer array be provided with each call as a communication area and a repository for certain data the HLI needs to have global. These arrays are called "partitions" in BUILDER; they are called "cursors" in the HLI section of the RELATE manual.

The DBIF has the capacity to work with 20 cursors. The DBIF is designed so that each retrieval path will have its own cursor; except for paths set up by an rvcslc (select) call, a SHOW PATH command given on any of these cursors would reveal only a single file open. Since paths set up by rvcslc may only use files open on another cursor, this means that the DBIF may work with no more than 20 files simultaneously.

The 50-word integer arrays are managed internally to the DBIF. The "rvc" routines are all integer functions which return a single word integer with a value between 1 and 20—a cursor
index. Calls to other DBIF routines supply this index to indicate which file they want to work with; the index is then used to pick out a particular element of the DBIF's 20x50 cursor array.

Thus, the information returned by the "rvc" routines in response to a path-creation call is useable only in queries and updates made through the "rtp" (and relcom) routines.

This design makes it unnecessary for application routines to create and manage large cursor data structures, and also makes intensive work with a few relations easier since the file name and index need be specified only once; after that only an integer variable is required in calling code.

This intensive use of a few relations is the most common form of programmatic data base access.

The design is limiting in that only 20 files can be open simultaneously, but remember that a single RELATE process can handle a maximum of about 25 open files before aborting with a memory overflow. The rdbinitx means of using multiple RELATE sons was not available at the time the DBIF was implemented.

Giving the DBIF a multiple-son handling capability would require paging of the cursor data structure as well as substitution of rdbinitx calls for rdbinit calls (and logic to detect when to use a new process as opposed to an old one). Otherwise the cursor data structure would begin to take up too much process memory in the Core.

10.2.4 DBIF Internals

Many high-level DBIF routines just call low-level routines which in turn call functionall similar HLI routines. For example, rtpupd calls dcgtup which calls rdbupdate. Given an understanding of the HLI, the structure of the DBIF is thus
fairly clear. However, string handling, error handling, and index management require some exposition.

10.2.4 String Handling

As noted above, the DBIF uses the string chain (CHN__) general purpose utilities to manage a string buffer. This buffer, called str, is 3K bytes long, and stored in the /strings/ block. A typical DBIF routine will receive a field list in the form a delimited string in a character variable. The field list must be left-justified in a word-aligned array for passage to RELATE, and must be uppercased. The routine will move the list into a (word-aligned) area of str via a call to the lstrng un-delimit utility, will uppercase the entire area, and will then pass name of the integer array equivalenced to str to the given HLI routine.

10.2.4.2 Error Handling

After every call to an HLI routine the DBIF uses dckerr to check to see if an error occurred during HLI execution. If one did (indicated by a non-zero value of the first word of the appropriate cursor), then the HLI routine rdberror is called with a request to print the RELATE error message corresponding to the problem which occurred, and dckcrs is called to print the status of some DBIF variables.

This error handling is one of the greatest benefits of using the DBIF, since any errors which occur are guaranteed detection and an at least moderately explicable error message.

Note that the DBIF uses lprnts 2 and 3, and that quite extensive running diagnostics of DBIF operations are generated if these are both set to .true.

10.2.4.3 Index Management

When a user specifies an index in an rvcsrc, rvcsts, or rvcslc call several things must happen, all of which are managed
by the rstidx routine. First, an attempt is made to do a SET
INDEX via a call to dgcidx. The method used is to query RELATE
for the indexes on the open file, doing the SET for the first one
which has at least fields matching the keys requested. Note that
if the request is for SCENARIO, CLASS, and the two indexes on the
file are SCENARIO, CLASS, HULL and SCENARIO, CLASS, the first index
will be the one chosen.

If this fails, the routine will attempt to create an index.
This creation will always fail, since all relations accessed
through the DBIF are opened with MODE=SHARED, and indexes can
only be created when the user has exclusive access. Thus, a
permanent index must exist which matches the request.

After a successful set, the length of the index fields in
words is determined via a call to lenidx. This will be needed if
rtpcal is ever called on the given relation, because the number
of words in the index to use is an argument to the rdbpoint
routine.

There are two cases in which this length will not just be
the length of the actual index used. The first case occurs when,
as in the example above, the number of words in the requested
index is less than the number in the index used, because there
are "superfluous" fields in the actual index. It is very
important that the argument to rdbpoint have the number of words
implicitly requested in this case: since the programmer has no
idea which index will be chosen by the DBIF, he will have
contructed his target buffer for rtpcal to be of length matching
only those fields in his index request. If the actual index
length were used, his points would always fail.

The second case occurs when the programmer wants to point
on only a subset of the index requested. He can do this by
replacing the comma following the last field he want included in
the point with a "|" in the index request he makes to rvcsrt,
rvcsts, or rvcslc. The number of words stored for use in
rdbpoint calls must in this case match the size of the fields
named before the "|". For example, a request for index
SCENARIO,CLASS|HULL would yield a word count of 11, not 12.

10.2.5 Security and the Scenario System "Presence"
In addition to the mechanics of data base access, the DBIF
is also concerned with security enforcement. In particular, it
is the last line of defense against unauthorized programmatic
data base changes (before RELATE security). Every DBIF routine
which modifies the contents of relations calls the rckprv utility
to check the user's privileges before doing so.

Changes may be disallowed (leading to a ZABORT) for two
reasons: the user does not have basic DB change privileges, as
specified by the ALTDB flag in the sysusr.sysro relation; or the
scenario the user is currently working with is using the given
relation's data indirectly, an access method which forbids
changes.

Developers should call the ckwprv logical function when
they open a relation to see if the user will have write
privileges, and abort gracefully if not.

The scenario system is also supported by calls to the
rvscen routine by all the "rvc" routines when they open a rela-
tion. Rvscen retrieves the proper scenario field key value for
the current scenario for the given relation from the scenario
system's extra data segment and places it in the appropriate
(cursor_index) location in the cursen array of the /scenar/
common block for referencing by application routines.

10.2.6 DBIF Modification
If it should be necessary to modify an recompile any DBIF
routines, be sure to re-create the dbif.obj file when compilation
is complete. Typically, simple compilation of any portion of the

10-159
DBIF will not result in changes to dbif.obj (e.g. compiling dbifa.src with the normal utilities will create or update dbifa.obj). Two re-create dbif.obj from the four constituent DBIF source libraries, use the command "GLUE dbif" at the MPE level.

Note that any new routines should always be assigned to segment dbif.
10.2.7 DBIF Subroutine Abstracts

DBIF UTILITY ROUTINES

C CKWPRV *****************************************
$CONTROL check=3,segm=2d1f
  LOGICAL FUNCTION ckwpv(modnam,filnam)
C*  *** FORMAL PARAMETER DECLARATIONS ***
    character*20 modnam,filnam
C*  *** ABSTRACT ***
C$PURPOSE Checks to see if User has write privilege for the
C file named for the current scenario. Useful at top of
C module initialization. Duplicates logic of rvchek.
C$AUDIT HISTORY
C  MSCarey 20-sep-83 AUTHOR
C$FORMAL PARAMETERS
C  modnam delimited name of module test being performed
C  for. If of nonzero length, ckwpv writes warning on
C  lack of write privilege.
C  filnam name of OB file to test privilege for
C$COMMON BLOCKS
C  uzrprv user privilege levels
C  scenar scenario status info
C  snref scenario set-up info and function declarations
C$CALLER various
C$METHOD
C  Uses logic similar to rvscen. Find name of file in list of
C  known DB files, then check to see if scenario field value for
C  that file matches current scenario overall name. Also check
C  overall user privilege levels.
C$
SUBROUTINE dcgdmn(relatn, flist, fmtlst)

CHARACTER*255 relatn, flist, fmtlst

*** FORMAL PARAMETER DECLARATIONS ***

CHARACTER*255 relatn, flist, fmtlst

*** ABSTRACT ***

PURPOSE change domain of a relation

AUDIT HISTORY

Densmore  12-Dec-82  AUTHOR

TYPE Database low-level interface utility

PARAMETERS

relatn  DTS relation name
flist   DTS field list
fmtlst DTS format list

cursrs  cursor buffers

METHOD

not currently allowed

10-162
OBIF UTILITY ROUTINES

C DGIDX***************************************************************
$CONTROL segment=dbif,check=3
LOGICAL FUNCTION dgidx(cursor,flist)
C*
*** FORMAL PARAMETER DECLARATIONS ***
CHARACTER*255 flist
INTEGER cursor
C*
*** ABSTRACT ***
C*PURPOSE sets or ChanGes InDeXes to an already open file
C*AUDIT HISTORY
C Densmore 12-Dec-82 AUTHOR
C*TYPE Database low-level interface utility
C*FORMAL PARAMETERS
Cin cursor cursor index to an open path
Cin flist DTS field list defining desired index
Cfunction dgidx .TRUE. if the desired index is found
C*COMMON BLOCKS
Cin/out cursrs cursor buffers
Cin/out strngs string buffers
C*METH00D
C Carey 07-Dec-83 Now uses process id-specific
C rdbinitx RELATE init routine
C perfo rms SET INDEX <fieldlist>
C**
**DBIF UTILITY ROUTINES**

```
C
DCGTUP..............................................................
$CONTROL segment=dbif,check=3
    SUBROUTINE dcgtup(cursor,flist,source)
    *** FORMAL PARAMETER DECLARATIONS ***
    INTEGER cursor,source(1)
    CHARACTER*255 flist
    *** ABSTRACT ***

C#PURPOSE Changes TUPLE (modifies)
C#AUDIT HISTORY
C    Densmore  12-Dec-82 AUTHOR
C#TYPE Database low-level interface utility
C#FORMAL PARAMETERS
Cin    cursor cursor index
Cin    flist DTS names of fields to be updated
Cin    source new values for each of these fields
C#COMMON BLOCKS
Cin/out cursrs cursor buffers
Cin/out strs string buffers
C#METHOD
C    calls rdbupdate
C##
```
C SUBROUTINE dcinit

*** ABSTRACT ***

C@PURPOSE Database Cursor INITialization
C@AUDIT HISTORY
C Densmore 15-Dec-82 AUTHOR
C@TYPE Database low-level interface utility
C@FORMAL PARAMETERS
C none
C@COMMON BLOCKS
Cout cursrs cursor buffers
Cout stnngs string buffers
C@METHOD
C initializes all chained buffer systems
C##
DBIF UTILITY ROUTINES

C

DCKCRS***************

SUBROUTINE dckcrs(cursor,out)

C*  *** FORMAL PARAMETER DECLARATIONS ***

INTEGER cursor,out

C*  *** ABSTRACT ***

C#PURPOSE Check Cursor...prints locally kept cursor information

C#AUDIT HISTORY

C  Densmore  26-Dec-82  AUTHOR

C#TYPE Database low-level interface utility

C#FORMAL PARAMETERS

Cin  cursor index to cursor

Cin  out  output logical unit number

C#COMMON BLOCKS

Cin  cursrs cursor buffers

Cin  indexs index buffers

C##
C  DCKERR************************************************************
$CONTROL segment=dbif,check=3
   LOGICAL FUNCTION dckerr(cursor)
C*     *** FORMAL PARAMETER DECLARATIONS ***
C*  INTEGER cursor
C*     *** ABSTRACT ***
C#PURPOSE checks for any errors in relate processing
C#AUDIT HISTORY
C  Densmore 12-Dec-82 AUTHOR
C#TYPE Database low-level interface utility
C#FORMAL PARAMETERS
Cin  cursor  cursor index
Cfunction dckerr returns true if an error exists
C#COMMON BLOCKS
Cin  cursrs  cursor buffers
C#METHOD
C uses rdberror; prints error information on ROBOUT ::= $STDLIST
C no error exists if first word in cursor buffer is zero
C#
SUBROUTINE dcs1ct(cursor)

C** FORMAL PARAMETER DECLARATIONS ***

    INTEGER cursor

C** ABSTRACT ***

C$PURPOSE Database...Closes current SeLeCT virtual cursor

C$AUDIT HISTORY

C Densmore 15-Dec-82 AUTHOR

C$TYPE Database low-level interface utility

C$FORMAL PARAMETERS

Cin/out cursor cursor index of cursor associated with the

C cursor to be closed

C$COMMON BLOCKS

Cin/out cursrs cursor buffers

C$METHOD

C see SELECT command.

C**
SUBROUTINE dcspth(cursorpthnam)

C** FORMAL PARAMETER DECLARATIONS ***

INTEGER cursor
CHARACTER*25S pthnam

C*** ABSTRACT ***

C* PURPOSE closes a previously open path and its cursor
C* AUDIT HISTORY
C Densmore 15-Dec-82 AUTHOR
C TYPE Database low-level interface utility
C FORMAL PARAMETERS
Cin/out cursor cursor pointer index
Cin pthnam DTS pathname to be closed
C COMMON BLOCKS
Cin/out cursrs cursor buffers
Cin/out strngs string buffers
C METHOD
C calls CLOSE PATH relate command
C**
SUBROUTINE delcrs(cursor)

C** FORMAL PARAMETER DECLARATIONS ***

INTEGER cursor

C** ABSTRACT ***

C#PURPOSE closes and DELeTES CuRSor
C#AUDIT HISTORY
C  Densmore  12-Dec-82 AUTHOR
C#TYPE Database low-level interface utility
C#FORMAL PARAMETERS
Cin/out  cursor  cursor index... set to zero indicating deallocation
C#COMMON BLOCKS
Cin/out  cursrs  cursor buffers
C#METHOD
C  closes cursor; deallocates cursor index
C##
DBIF UTILITY Routines

SUBROUTINE delidx(cursor, relatn, flist)

C* *** FORMAL PARAMETER DECLARATIONS ***

CHARACTER*255 relatn, flist
INTEGER cursor

C* *** ABSTRACT ***

C#PURPOSE deletes an index from an already open relation
C#AUDIT HISTORY
C  Densmore    12-Dec-82 AUTHOR
C#TYPE Database low-level interface utility
C#FORMAL PARAMETERS
Cin  cursor   index to cursor opened under the pathname RELATN
Cin  relatn   OTS relation name
Cin  flist    OTS field-list defining index
C#COMMON BLOCKS
Cin/out cursrs  cursor buffers
Cin/out strings string buffers
C#METHOD
C  call set-index using flist; then calls
C  rdbinfo to get index number so it can be deleted.
C**
SUBROUTINE delrel(cursor, relatn)

*** FORMAL PARAMETER DECLARATIONS ***

CHARACTER*20 relatn
INTEGER cursor

*** ABSTRACT ***

PURPOSE deletes a relation; assumes that the relation is open

AUDIT HISTORY

Densmore 12-Dec-82 AUTHOR

TYPE Database low-level interface utility

PARAMETERS

Cin cursor index to cursor opened under the pathname RELATN
Cr reltn DTS relation name to delete

COMMON BLOCKS

cursrs cursor buffers

METHOD

Calls for a PURGE FILE command...also deallocates cursor resources
SUBROUTINE deltup(cursor)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER cursor

*** ABSTRACT ***

PURPOSE deletes the current tuple

AUDIT HISTORY

Dr. Densmore 12-Dec-82 AUTHOR

TYPE Database low-level interface utility

FORMAL PARAMETERS

IN cursor cursor index

COMMON BLOCKS

IN/OUT cursrs cursor buffers

METHOD

calls rdbdelete

***
FUNCTION dmkcrs(dummy)

INTEGER dummy

*** ABSTRACT ***

PURPOSE retrieves a unique cursor index from chain data type and initializes the cursor...Database Make Cursor.

Densmore 12-Dec-82 AUTHOR

TYPE database low-level interface utility

FORMAL PARAMETERS

dummy dummy variable

dmkcrs a unique index taken from the crschn chain data type which indexes a cursor in the array crs of RELATE cursors. This integer is used throughout the RELATE utilities to represent a cursor.

COMMON BLOCKS

cursors cursor buffers

METHOD

calls chnalo...assumes that chain is initialized; then initializes the indexed cursor via RDBINIT.

10-174
SUBROUTINE dmkidx(cursor, relatn, flist, unary)

INTEGER cursor
CHARACTER*255 relatn, flist
LOGICAL unary

C*** ABSTRACT ***

C#PURPOSE Database create (MaKe) InDeX for relate system;
C the relation named by relatn must be open.
C#AUDIT HISTORY
C Densmore 12-Dec-82 AUTHOR
C#TYPE Database Low-level interface utility
C#FORMAL PARAMETERS
Cin cursor index to cursor opened under the pathname RELATN
Cin relatn DTS name of relation
Cin flist DTS field list for indexing purposes
Cin unary logical: .TRUE. if no key may be duplicated or
C allowed in index
C#COMMON BLOCKS
Cin/out cursrs cursor buffers
Cin/out strngs string buffers
C#METHOD
C forms command string, calls relate.
C"
INTEGER FUNCTION dmkrel(relatn, pthnam, struct)

CHARACTER*255 relatn, pthnam
CHARACTER*1 struct(600)

*** ABSTRACT ***

PURPOSE Database, Makes RELation; returns cursor

AUDIT HISTORY
Densmore 22-Feb-83 Deleted USEPTH arg so that high level routines easily interface
Carey 10-Feb-83 Made struct an array to accommodate big field lists.

AUTHOR
Densmore 12-Dec-82

TYPE Database low-level interface utility

PARAMETERS
relatn DTS relation name
pthnam DTS path name, if not same as relation name
struct DTS field name list specifying structure of relation

cursor index to the new cursor constructed.

COMMON BLOCKS
currs cursor buffers
strngs string buffers

METHOD
creates new cursor; creates command; calls relate.

SUBROUTINE dmktup(cursor, list, source)

INTEGER cursor, source(l)

CHARACTER*255 list

FUNCTION dmktup

*** FORMAL PARAMETER DECLARATIONS ***

C** PURPOSE: Database Make Tuple -- adds tuple to relation pointed to by cursor.

C** AUDIT HISTORY

C** Densmore 14-DEC-82 AUTHOR

C** TYPE: Database low-level interface utility

C** FORMAL PARAMETERS

Cin  cursor  index to a cursor from the pool

Cin  list   DTS list of fields in tuple to be added

Cin  source array of data referenced by list to be added as the new tuple

Cf  function dmktup returns success index... 0 means successful;

C    1 means unary violation; 2 means file full (EOF)

C** COMMON BLOCKS

Cin/out cursrs cursor buffers

Cin/out strngs string buffer variables

C** METHOD

C word aligns list, then calls rdadd
C**
DOPPTH

INTEGER FUNCTION doppth(relatn,pthnam)

CHARACTER*255 relatn,pthnam

*** ABSTRACT ***

PURPOSE opens a path to the named relation

C#AUIDIT HISTORY

Densmore 15-Dec-82 AUTHOR

Carey 5-may-83 open all files in shared mode

C#TYPE Database low-level interface utility

C#FORMAL PARAMETERS

Cin relatn DTS relation name

Cin pthnam DTS path name

C#COMMON BLOCKS

Cin/out cursrs cursor buffers

Cin/out strings string buffers

C#METHOD

calls OPEN PATH relate command

C##
SUBROUTINE dpcmd(cursor, routin, kstr, len)

**FORMAL PARAMETER DECLARATIONS**

```
CHARACTER*6 routin
INTEGER len, cursor

INTEGER kstr((len+1)/2)
```

**ABSTRACT**

For use when LPRNT 3 is on to print RELATE commands issued

**AUIDT HISTORY**

Densmore 14-Feb-83 AUTHOR

**TYPE** diagnostic

**FORMAL PARAMETERS**

- `cursor` cursor index
- `routin` character*6 routine name
- `kstr` integer array containing characters of cmd
- `len` number of characters in kstr

**COMMON BLOCKS**

- `lprnt` diagnostic flags and ioutp

**CALLER** all D... relate routines

---

**DBIF UTILITY ROUTINES**

---

10-179
SUBROUTINE dpcmdl(cursor,routin)

CHARACTER*6 routin
INTEGER cursor

*** ABSTRACT ***

C* PURPOSE Like OPCMD, except for zero length character strings
C* AUDIT HISTORY
C* TYPE diagnostic
C* FORMAL PARAMETERS
Cin cursor cursor index
Cin routin routine name
C* COMMON BLOCKS
Cin lprnts diagnostic flags and ioutp
C* CALLFP all O... routines
C**  

C

$CONTROL segment=dbif,check=3
SUBROUTINE dpcmdl(cursor,routin)

C* *** FORMAL PARAMETER DECLARATIONS ***

C*
C DREWND*****************************************************************************************
$CONTROL segment=dbif,check=3
SUBROUTINE drewnd(cursor)

C* *** FORMAL PARAMETER DECLARATIONS ***

C* INTEGER cursor

C* *** ABSTRACT ***

C* PURPOSE rewinds relation corresponding to cursor
C* AUDIT HISTORY
C Densmore 02-Feb-83 AUTHOR
C* TYPE low-level relate database utility
C* FORMAL PARAMETERS
C* Cin cursor cursor to be rewound
C* CALLER rvcrwd
C* METHOD
C Calls RDBPOINT with rewind flag set.
C**
C DSOPEN

$CONTROL segment=dbif,check=3
INTEGER FUNCTION dsopen(crss,ncrss)

*** FORMAL PARAMETER DECLARATIONS ***
INTEGER ncrss,crss(ncrss)

*** ABSTRACT ***

C* PURPOSE Inits a new cursor and opens files associated with each crs
C* AUDIT HISTORY
C Densmore 23-Mar-83 AUTHOR
C*TYPE low level RELATE database utility
C*FORMAL PARAMETERS
Cin crss cursor index for each cursor associate with a file
Cin ncrss length of crss
C*COMMON BLOCKS
Cin/out crsr s cursor buffers
C*CALLER rvcslc
C*METHOD
C loops over cursors getting current db number; retrieves full
C filename using rdbinfo again; opens each file on the new cursor
C*LOCAL VARIABLES
C icrs do index
C cursor each crss value
C ndb database number
C len length of dbname
C icmd length of command
C info info array for rdbinfo (dbname)
C dbname full file name for database (info)
C comand full RELATE command (icmd)
C icmd integer version of (comand)
C##
SUBROUTINE dslect(cursor, tgtlst, unique, keylst, cond)

FORMAL PARAMETER DECLARATIONS

LOGICAL unique
INTEGER cursor
CHARACTER*255 tgtlst, keylst, cond

PURPOSE implements RELATE's select command

AUDIT HISTORY

Densmore 15-Dec-82 AUTHOR

TYPE Database low-level interface utility

FORMAL PARAMETERS

cursor cursor index for cursor on which select is to be done

tgtlst DTS target list, indicating what fields should be returned and the values they should assume; in the form name1[=expr][,name2[=expr]]...

unique LOGICAL indicates that selection results unique values in the key list

keylst DTS names of fields on which selection is sorted optional unless unique is TRUE; avoid specification via the DTS ';'

cond DTS condition which created virtual tuples should be returned.

COMMON BLOCKS

csr cursor buffers
strngs string buffers

METHOD

performs RELATE select command


SUBROUTINE dtcalc(cursor,keyval,notfnd)
C** *** FORMAL PARAMETER DECLARATIONS ***
INTEGER cursor,keyval(1)
LOGICAL notfnd
C** *** ABSTRACT ***
PURPOSE Calculates position of next tuple

Densmore 21-Feb-83 Add RDBINFO to retrieve key
Densmore 18-Feb-83 Remove FLIST,TUPLE arguments
Densmore 15-Dec-82 AUTHOR

TYPE Database low-level interface utility

FORMAL PARAMETERS
Cin  cursor  cursor index
Cin  keyval  key value to search for
Cout notfnd  not-found flag -- .TRUE. if tuple not found

COMMON BLOCKS
Cin/out cursrs  cursor buffers
Cin/out strngs  string buffers

METHOD
C retrieves length of KEYVAL from crsxl, then calls RDBPOINT
# DBIF UTILITY ROUTINES

C

**dtnext**

*CONTROL segment-dbif,check=3*

SUBROUTINE dtnext(cursor,flist,tuple,eof)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER cursor,tuple(l)
CHARACTER*25S flist
LOGICAL eof

*** ABSTRACT ***

PURPOSE Returns next tuple associated with cursor

AUDIT HISTORY

Oensmore 15-Dec-82 AUTHOR

TYPE Database low-level interface utility

FORMAL PARAMETERS

Cin/out cursor cursor index
Cin flist DTS field list
Cout tuple destination for next tuple
Cout eof returns TRUE if no next tuple, FALSE otherwise

COMMON BLOCKS

Cin/out cursrs cursor buffers
Cin/out strsngs string buffers

METHOD

Calls rdbread

C#
C SUBROUTINE lenidx(cursor, indx, newidx)

C*** FORMAL PARAMETER DECLARATIONS ***

INTEGER cursor
CHARACTER*1 indx(1)
LOGICAL newidx

C*** ABSTRACT ***

C* PURPOSE Sets the word length of an index in a cursor

C* AUDIT HISTORY

C Densmore 22-Mar-83 AUTHOR

C* TYPE low-level RELATE database utility

C* FORMAL PARAMETERS

Cin cursor cursor index
Cin indx DTS string describing index...not used
C if the cursor has no current index
Cin newidx .TRUE., if this index was just created
C and therefore must have length=maxlen

C* COMMON BLOCKS

Cin cursrs cursor buffers

C* CALLER rvc... with sort requests

C* METHOD

C Lots of ROBINFO calls.

C Assumes that the index in question is the current index
C and that indx describes it.
C
C First, the filenumber and indexnumber are retrieved by
C an info call using the cursor (current path). If there is no
C index then it is assumed crsxl is to be set to the number
C of words in a tuple. Otherwise, set crsxl to the number
C of words in the index described by the fields in indx.
C
C Now, the current index has been set using indx, but
C RELATE is such that there may be MORE fields in the current
C index. This occurs whenever an index already exists whose
C first N fields match the N fields given in indx. In
C this case, only the sum of the number of words in the first
C N fields of the index should be used in setting crsxl.

C* LOCAL VARIABLES

C info returned info from ROBINFO (except field numbers)
C fieldn returned field numbers from ROBINFO ([1]=quantity)
C indexn index number for this path
C filen file number for this path
C maxlen maximum possible length (words) for index
C excess number of extra words given as index length
C presumably to include the line number field
C length length for this index
C count number of commas plus one, in indx -- i.e.
C the number of fields in the index actually used
OBIF UTILITY ROUTINES

first first significant character in DTS indx
last last
len last-first+1
char DO index from first to last
i DO index
nfield fieldn(1) = number of fields in index
SUBROUTINE rcinit

*** ABSTRACT ***
PURPOSE Initialize RELATE Cursor system for database interfacing

AUDIT HISTORY
Densmore 17-Dec-82 AUTHOR

TYPE RELATE Database High-Level Interface Utility

COMMON BLOCKS
Cout  indexs  index buffers
C**
DBIF UTILITY ROUTINES

C  RCKPRV

*CONTROL segment=dbif,check=3
  LOGICAL FUNCTION rckprv(cursor,path)

C*  *** FORMAL PARAMETER DECLARATIONS ***
  integer cursor
  character*20 path
C*  *** ABSTRACT ***
C$PURPOSE Checks user write privilege on call to a relate utility routine which will change a relation's contents
C$AUDIT HISTORY
C  MSCarey  10-sep-83  AUTHOR
C$FORMAL PARAMETERS
Cin  cursor  relate cursor index
Cin  path  name of the path for this cursor
C$COMMON BLOCKS
Cin  scenar  current scenario information
C$CALLER high-level relate utilities
C$METHOD
C  Write a message if no write privilege and return.
C$
C RELCOM **************************************************
$CONTROL check=2,segment=dbif
   SUBROUTINE relcom(incurs,comand)
C* *** FORMAL PARAMETER DECLARATIONS ***
   integer incurs
   character comand(1020)
C* *** ABSTRACT ***
C* PURPOSE    Executes a RELATE command.
C* AUDIT HISTORY
C MSCAREY     09-aug-83 AUTHOR
C* FORMAL PARAMETERS
C in incurs  cursor usage code: 1-mccrs indicates use
C              specified cursor; >mccrs indicates use any
C              open cursor.
C* COMMON BLOCKS
C in cursrs  relate cursors
C in lprnts  debug switches
C* CALLER various
C* METHOD
C    Check the cursor code and set the cursor to use.
C    Extract the command from the delimited string.
C    Make the call to RELATE, and check for errors.
C* LOCAL VARIABLES
C
C##
SUBROUTINE rstidx(rtn, cursor, relatn, flist)

C* *****************************************************************
C* FORMAL PARAMETER DECLARATIONS ***
C* INTEGER cursor
C* CHARACTER rtn*6, relatn*25S, flist*255
C* ** ABSTRACT ***
C* PURPOSE For RVC sort open routines, Sets up Indexes
C* AUDIT HISTORY
C* Oensmore 15-Jun-83 AUTHOR
C* TYPE RELATE Database utility for High-Level routines
C* FORMAL PARAMETERS
Cin rtn name of calling routine
Cin cursor the cursor just opened by calling routine
Cin relatn DTS relation name for the cursor
Cin flist DTS field list for the (possibly new) index
C* COMMON BLOCKS
Cin/out cursrs cursor buffers
Cin/out indexes index buffers
Cin lprnts diagnostic block
C* CALLER rvcart, rvcsts
C* METHOD
C Calls dcgidx with full fieldlist. If dcgidx fails, then the
C desired index is known not to exist; it is created, and the
C corresponding fieldlist is stored in the idx array. When
C dcgidx succeeds it means that the index will not be destroyed
C when rvclos is called to close the cursor.
C
C When the fieldlist includes a vertical bar (|) in place of
C exactly one of the commas (,) delimiting the field names, it
C means that the caller desires the index to be opened as before,
C but that only the fields up to the bar are to be used when any
C calcs are performed. In this way, one may allow calcs to
C certain fields, and then guarantee ordered sequential reads
C for the following fields even though the latter fields are not
C included in the calc. This is implemented by searching for the
C character and using a different index length (lenidx).
C* LOCAL VARIABLES
C i|bar location of vertical bar: 0 if none
C indx place where idx buffer is located in idx array
C*
**OBIF UTILITY ROUTINES**

```plaintext
C RTPADD**************************************************************************
$CONTROL segment=dbif,check=2
   SUBROUTINE rtpadd(cursor,flist,source)

C* *** FORMAL PARAMETER DECLARATIONS ***
   INTEGER cursor,source(1)
   CHARACTER*255 flist

C* *** ABSTRACT ***
C* PURPOSE Relate Tuple ADDition: adds tuple to current relation
C* AUDIT HISTORY
C   Densmore 17-Dec-82 AUTHOR
C* TYPE RELATE Database High-Level Interface Utility
C* FORMAL PARAMETERS
Cin    cursor  cursor index for current relation
Cin    flist   field list for tuple
Cin    source source for new data
C* METHOD
C   Calls DMKTUP
C**
```
SUBROUTINE rtpcal(cursor,keyval,flist,dest,notfnd)

INTEGER cursor,keyval(l),dest(l)
CHARACTER*255 flist
LOGICAL notfnd

C* ABSTRACT C*
C* PURPOSE Relate TuPle CALculate: calculates by key-value the next tuple
C* desired from the current relation.
C* AUIDT HISTORY
C* Densmore 17-Dec-82 AUTHOR
C* TYPE RELATE Database High-Level Interface Utility
C* FORMAL PARAMETERS
Cin cursor cursor index for the current relation
Cin keyval value of the key for the tuple desired; the current
C* relation must be indexed by this key
Cin flist field list for tuple
Cout dest output tuple (DESTination)
Cout notfnd Logical indicating if the tuple was NOT FouND
C*METHOD
C Calls dtcalc
C*
SUBROUTINE rtpdel(cursor)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER cursor

*** ABSTRACT ***

C* PURPOSE Relate TuPle DELeate: deletes current tuple in current relation

C* AUDIT HISTORY

C TYPE RELATE Database High-Level Interface Utility

C FORMAL PARAMETERS

Cin cursor cursor index for the current relation

C COMMON BLOCKS

Cin/out cursrs cursor buffers

C METHOD

C Calls deltup

C**
C RTPKIL******************************************************************************
$CONTROL segment-dbif,check=2
SUBROUTINE rtpkil(cursor,keyval,notfnd)
C* *** FORMAL PARAMETER DECLARATIONS ***
INTEGER cursor,keyval(1)
LOGICAL notfnd
C* *** ABSTRACT ***
C#PURPOSE finds and deletes (KILls) the tuple whose key is keyval
C#AUDIT HISTORY
C Denimore 18-Feb-83 AUTHOR
C#TYPE high-level relate utility
C#FORMAL PARAMETERS
Cin cursor cursor index
Cin keyval key value -- must correspond to current index
Cin notfnd true if NOT FOuND
C++
C RTPNEW

$CONTROL segment=OBIF,check=2
   INTEGER FUNCTION rtpnew(cursor,flist,source)
C*   *** FORMAL PARAMETER DECLARATIONS ***
      INTEGER cursor,source(1)
      CHARACTER*255 flist
C*   *** ABSTRACT ***
C#PURPOSE Relate Tuple NEW--add NEW tuple to current relation; returns mode
C#AUDIT HISTORY
C Densmore 23-Mar-83 AUTHOR
C#TYPE high level relate DB utility
C#FORMAL PARAMETERS
Cin  cursor relate cursor
Cin  flist field list
Cin  source source data making up tuple
Cfunction rtpnew 0=succesful add  1=unary violation
C   2=EOF--no room to add tuple in file
C##

10-196
SUBROUTINE rtpnfd(cursor,keyval,flist,dest,fnd,eof)
C* ** FORMAL PARAMETER DECLARATIONS ***
INTEGER cursor,keyval(l),dest(1)
CHARACTER*ZS5 flist
LOGICAL fnd,eof
C* ** ABSTRACT ***
C* PURPOSE Relate TuPle Not Found: calculates by key-value the next tuple
C desired from the current relation, expecting NOT to match
C the key value. Then reads and returns the record
C which the failed point left us at, i.e. the next
C greatest value of the key. Similar to RTPCAL.
C*AUDIT HISTORY
C MSCarey 31-may-83 AUTHOR
C#TYPE RELATE Database High-Level Interface Utility
C#FORMAL PARAMETERS
Cin cursor cursor index for the current relation
Cin keyval value of the key for the tuple desired: the current
C relation must be indexed by this key
Cin flist field list for tuple
Cout dest output tuple (DESTination)
Cout fnd Logical indicating that an exact match on the
C key value was found, which is an error here.
Cout eof True if the point left us at the end of the
C relation, with no tuple to return.
C#METHOD
C Calls dtcalc
C**
SUBROUTINE rtpnxt(cursor, flist, dest, eof)

C* *** FORMAL PARAMETER DECLARATIONS ***
INTEGER cursor, dest(l)
CHARACTER*255 flist
LOGICAL eof

C* *** ABSTRACT ***
C* PURPOSE Relate TuPe NeXT: obtain next tuple in sequence from
C* current relation.
C* AUDIT HISTORY
C Densmore 17-Dec-82 AUTHOR
C TYPE RELATE Database High-Level Interface Utility
C FORMAL PARAMETERS
Cin cursor cursor index for current relation
Cin flist field list for tuple
Cout dest output tuple (DESTination)
Cout eof Logical indicating if no more tuples are available
C COMMON BLOCKS
Cin/out cursrs cursor buffers
C METHOD
C Calls dtnext
C**
SUBROUTINE rtprep(cursor,keyval,flist,source,notfnd)

C***

FORMAL PARAMETER DECLARATIONS ***

INTEGER cursor,keyval(1),source(1)
CHARACTER*255 flist
LOGICAL notfnd

C*** ABSTRACT ***

PURPOSE finds tuple whose key value is keyval; replaces it w/ source

AUDIT HISTORY

Densmore       18-Feb-83     AUTHOR

TYPE high-level relate utility

FORMAL PARAMETERS

cin   cursor      cursor index
cin   keyval     key value -- corresponds to current index
cin   flist      field list to which tuple source data corresponds
cin   source     source data for the tuple to be updated
cout  notfnd    True if tuple was NOT FouND

c##
**SUBROUTINE rtpupd(cursor, flist, source)**

C*  *** FORMAL PARAMETER DECLARATIONS ***

INTEGER cursor, source(1)
CHARACTER*255 flist

C*  *** ABSTRACT ***

C#PURPOSE Relate Tuple UPDATE: modify the value of the current tuple
C in the current relation.
C
C#AUDIT HISTORY
C  Densmore  17-Dec-82 AUTHOR
C#TYPE RELATE Database High-Level Interface Utility
C#FORMAL PARAMETERS
Cin  cursor  cursor index for the current relation
Cin  flist  field list for tuple
Cin  source  source for new tuple data
C#COMMON BLOCKS
Cin/out cursrs  cursor buffers
C#METHOD
C  Calls dcgtup
C**
SUBROUTINE rvclos(cursor)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER cursor

*** ABSTRACT ***

C* PURPOSE Relate Virtual Cursor CLOSE: close the current relation
C* AUDIT HISTORY
C Densmore 17-Dec-82 AUTHOR
C* TYPE RELATE Database High-Level Interface Utility
C* FORMAL PARAMETERS
Cin/out cursor cursor index for current relation; set to zero
C* COMMON BLOCKS
Cin/out cursrs cursor buffers
Cin/out indexs index buffers
C* METHOD
C Depending on cursor type, calls dcsapth or dcsalct.
C If an index was created during opening, it is purged via delidx.
C**
INTEGER FUNCTION rvcfil(relatn, struct)

CHARACTER*255 relatn, struct

*** ABSTRACT ***

PURPOSE to CREATE a new relation with structure STRUCT

AUDIT HISTORY

Densmore 22-Feb-83 AUTHOR

TYPE high-level relate utility

FORMAL PARAMETERS

relatn relation name -- DTS
struct the relation structure -- DTS (see RELATE manual)

COMMON BLOCKS

cursor cursor buffers


SUBROUTINE rvckil(cursor, relatn)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER cursor
CHARACTER*255 relatn

*** ABSTRACT ***

C* PURPOSE PURGES the relation named and all its indexes
C* AUDIT HISTORY
C Densmore 23-Feb-83 AUTHOR
C TYPE high-level relate utility
C FORMAL PARAMETERS
Cin  cursor  cursor index
Cin  relatn  relation name (file name)
C COMMON BLOCKS
Cin/out cursrs  cursor buffers
Cin/out indexs  index buffers
Cin  lprnts  diagnostics
C METHOD
C  cursor must not be from a selection; deallocates index buffers;
C  calls delrel to delete relation and deallocate cursor buffers.
C**
C RVCPTH

$CONTROL segment-dbif,check=2
  INTEGER FUNCTION rvcpth(relatn,synym,struct)
C
  *** FORMAL PARAMETER DECLARATIONS ***
  CHARACTER*255 relatn,synym,struct
C
  *** ABSTRACT ***
C PURPOSE Relate Virtual Cursor PaTH; CREATES a relation under a
C synonymous name.
C AUDIT HISTORY
C Densmore 22-Feb-83 AUTHOR
C TYPE RELATE Database High-Level Interface Utility
C FORMAL PARAMETERS
Cin relatn Delimited Text String giving relation name
Cin synym Delimited Text String giving desired synonym
Cin struct OTS structure spec as in RVCFIL
C COMMON BLOCKS
Cin/out cursrs cursor buffers
C METHOD
C Calls dmkrel and cursor type is noted
C

10-204
C RVCREL

$CONTROL segment=dbif,check=3

INTEGER FUNCTION rvcrel(relatn)

C*** FORMAL PARAMETER DECLARATIONS ***

CHARACTER*20 relatn

C*** ABSTRACT ***

C* PURPOSE Relate Virtual Cursor open RELation: opens a relation

C* AUDIT HISTORY

C Densmore 17-Dec-82 AUTHOR

C* TYPE RELATE Database High-Level Interface Utility

C* FORMAL PARAMETERS

C* IN relatn Delimited Text String specifying name of relation

C* FUNCTION rvcrel a cursor index to the new relation

C* COMMON BLOCKS

C* IN/OUT cursrs cursor buffers

C* METHOD

C Calls doprel and notes cursor type.

C**
SUBROUTINE rvcrwd(cursor)

INTEGRER cursor

PURPOSE Relate Virtual Cursor ReWinD: rewind current relation

HISTORY 17-Dec-82 AUTHOR

TYPE RELATE Database High-Level Interface Utility

PARAMETERS
cursor cursor index for current relation

BLOCKS
cursor buffers

METHOD
C calls drewnd.

10-206
DBIF UTILITY ROUTINES

C RVCSLC

$CONTROL segment=dbif,check=3

INTEGER FUNCTION rvcslc(tgtin,unique,keyin,condin)

C* *** FORMAL PARAMETER DECLARATIONS ***

CHARACTER*255 tgtin,keyin,condin
LOGICAL unique

C* *** ABSTRACT ***

C* PURPOSE Relate Virtual Cursor Selection: performs a SELECT operation

C* AUDIT HISTORY

Densmore 17-Dec-82 AUTHOR

C* TYPE RELATE Database High-Level Interface Utility

C* FORMAL PARAMETERS

Cin tgtin Delimited Text String indicating what fields should be returned, and the values they should assume;

Format: name1[=expr1],name2[=expr2]...

Cin unique Logical indicating that selection should result in unique values in the key list keylist; forces the specification of keylist

Cin keyin Delimited Text String names of fields on which the selection is to be sorted; optional unless unique is True; avoid specification via the DTS '::'

Cin condin Delimited Text String giving the condition under which any virtual tuples created by this select should be returned as part of the select

C function rvcslc virtual cursor index pointing to the cursor associated with the selection results

C* COMMON BLOCKS

Cin/out cursrs cursor buffers

C* METHOD

C First, figures out which cursors need to be associated with the new SELECT cursor. Currently, all open cursors are used.

C Then, dslect is called to perform the selection command.

C**
C

RVCSRT

$CONTROL segment=dbif,check=3

INTEGER FUNCTION rvcsrt(reltn,flist)

C

*** FORMAL PARAMETER DECLARATIONS ***

C

*** ABSTRACT ***

C

PURPOSE Relate Virtual Cursor via Sort: opens a new relation sorted via a specified key

C

AUDIT HISTORY

C

Densmore 17-Dec-82 AUTHOR

C

TYPE RELATE Database High-Level Interface Utility

C

FORMAL PARAMETERS

C

relatn Delimited Text String naming the relation

C

flist Delimited Text String naming the fields which form the key upon which to sort

C

The vertical bar (|) has significance when it appears in this argument as described in RSTIDX.

C

function rvcsrt cursor index to cursor associated with the named relation/index pair

C

COMMON BLOCKS

C

cursrs cursor buffers

C

indexs index buffers

C

METHOD

C

Calls dopth, then dcoidx. If dcoidx fails, the index is created via dmkidx, and this fact is noted.

C

10-208
C RVCSTS****************************************************
$CONTROL segment=dbif,check=3
  INTEGER FUNCTION rvcsts(relatn,synym,flist)
C*  *** FORMAL PARAMETER DECLARATIONS ***
C*  CHARACTER*255 relatn,synym,flist
C*  *** ABSTRACT ***
C#PURPOSE Relate Virtual Cursor via Sort; return Synonym: opens a new
C  relation sorted via a specified key and returns the cursor
C  associated with a synonym to that relation
C#AUDIT HISTORY
C  Densmore 17-Dec-82 AUTHOR
C#TYPE RELATE Database High-Level Interface Utility
C#FORMAL PARAMETERS
Cin relatn Delimited Text String naming the relation
Cin synym Delimited Text String naming synonym
Cin flist Delimited Text String naming the fields which form
C  the key upon which to sort
C  The vertical bar (|) character has meaning in
C  this arg as defined by RSTIDX
Cfunction rvcsts cursor index to cursor associated with the named
C  synonym/index pair
C#COMMON BLOCKS
Cin/out cursrs cursor buffers
Cin/out indexs index buffers
C#METHOD
C  Calls doppth, then dcgidx. If dcgidx fails, the index is
C  created via dmkidx, and this fact is noted.
C##
RVCSYN

$CONTROL segment=dbif,check=3
INTEGER FUNCTION rvcsyn(relatn,synym)

CHARACTER*255 relatn,synym

*** ABSTRACT ***
C* PURPOSE Relate Virtual Cursor SYNonym: opens a relation under a
C* synonymous name.
C* AUDIT HISTORY
C* Densmore    17-Dec-82 AUTHOR
C* TYPE RELATE Database High-Level Interface Utility
C* FORMAL PARAMETERS
Cin relatn   Delimited Text String giving relation name
Cin synym   Delimited Text String giving desired synonym
C* COMMON BLOCKS
Cin/out cursrs cursor buffers
C* METHOD
C* Calls doppth and cursor type is noted

C
DBIF UTILITY ROUTINES

C RVSCEN

$CONTROL segment=dbif,check=3

SUBROUTINE rvscen(cursor,type,file)

C* *** FORMAL PARAMETER DECLARATIONS ***
  integer cursor,type
  character*20 file

C* *** ABSTRACT ***

C#PURPOSE Checks/sets user access flags to the relation
being requested, and sets the scenario field key value
for this relation for the current scenario.

C#AUDIT HISTORY
C MSCarey 10-sep-63 AUTHOR

C#FORMAL PARAMETERS
Cin cursor utility system cursor index
Cin type cursor type (1, 2, or 3)
Cin file name of relation to be processed

C#COMMON BLOCKS
Cin senprm scenario system parameters
Cin uzrprv user privilege information
Cin snrref field key values for each relation
Cio scenar current scenario settings

C#CALLER relate utilities which open cursors

C#METHOD
C Set write privilege flag for this cursor according to
user privileges.
C Make sure the filename has a group suffix.
C Search snrlsn (extended memory) for a match of the filename
C On finding a match, set cursen(cursor) to snrlsn(match)

10-211
10.3 BUILDER-CALLABLE FORTRAN UTILITIES

This Section presents FORTRAN utility routines designed to serve BUILDER screens, with an emphasis on the routines likely to be of interest to developers of any BUILDER-based ALIAS module. All the routines were originally developed to serve the DBU.

Source code for the routines can be found in the slproc.src and sldate.src files. Object code processable by PREP is not maintained; run-time linkable object code is maintained in the account Segmented Library (sl.pub). Update of the contents of sl.pub is accomplished by compiling both source code files into $oldpass, and then running the Segmenter via the "GLUE addsl" command. Note that any new routines must be compiled into the "dsa" segment.

A Segmented Library is the source for program unsatisfied externals at run time. By calling the proper intrinsics, a program can even link routines after execution has begun. BUILDER must do this in order to implement the CALL PROCEDURE facility, since it has no idea what routines might be called until the moment it interprets a CALL PROCEDURE line. Note that routines in an SL may not contain common, data, read, or write statements. Global storage requirements must be met through use of BUILDER memory or extra data segments. For more information about SLs, see the Segmenter manual for the HP 3000.

Table 10-7 contains an annotated listing of all the routines which reside in the SL. The following sections will discuss selected routines by purpose. See Section 10.3.6 for abstracts of the SL routines, which contain detailed calling specifications.

10.3.1 BUILDER-FORTRAN Data Transfer

The BUILDER manual section on CALL PROCEDURE specifies that FORTRAN routines to be called must have three formal parameters: a 50-word integer array for the current cursor, a "table" of
<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABTRNS</td>
<td>aborts a transaction on all son RELATE son processes started up via the file management subsystem. See the RELATE reference manual for a discussion of what transactions are.</td>
</tr>
<tr>
<td>BGTRNS</td>
<td>Like ABTRNS, but a global BEGIN TRANSACTION.</td>
</tr>
<tr>
<td>CALCDATE</td>
<td>Specialty routine serving the PROJ_NC_SKED DBU screen. Allows quick recalculaton of ship schedule dates given a basis date and a set of planning factors (intervals between milestones).</td>
</tr>
<tr>
<td>CDTOOD</td>
<td>Character Date TO DDate. Same as utility of same name in RL. In SL only to allow use by BUILDER-called routines.</td>
</tr>
<tr>
<td>CURINI</td>
<td>These two routines form the FORTRAN part of the multiple-RELATE-son-process relation management system. This system is usable from any BUILDER module, not just the DBU. The only restriction is that each user of the system must specify a unique value of the SCREENSYS Job Control Word. See Section 8.4.3.2 for further detail on the system. CURINI initializes a file management system invocation, CURSWP swaps a cursor from the system's storage in an extra data segment into a BUILDER partition.</td>
</tr>
<tr>
<td>CURSWP</td>
<td></td>
</tr>
<tr>
<td>DDATEM</td>
<td>Date utility routines virtually identical to their counterparts of the same names in UTLR (see Section 10.1). In SL to allow usage by BUILDER-called routines.</td>
</tr>
<tr>
<td>DCLRFY</td>
<td>Similar to ltrim in UTLR. Integer function returning the leftmost non-blank character of a string.</td>
</tr>
<tr>
<td>DDTODCD</td>
<td>Similar to ABTRNS above, but does a global COMMIT TRANSACTION command.</td>
</tr>
<tr>
<td>DDTOID</td>
<td>Similar to rtrim in UTLR. Integer function returning the rightmost non-blank character in a string.</td>
</tr>
<tr>
<td>DLTRIM</td>
<td>These routines create son processes running the HP and TDP editors, respectively. They are obsolete now that BUILDER can transparently create son processes for you when you give :RUN commands.</td>
</tr>
<tr>
<td>DRUNED</td>
<td>Moves a string from a specified word address into an integer array. Useful when the CALL PROCEDURE needs to be read using the contents of the third formal parameter.</td>
</tr>
<tr>
<td>DSAFET</td>
<td></td>
</tr>
<tr>
<td>ROUTINE</td>
<td>PURPOSE</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DSAGETC</td>
<td>Same as dsafetch, but transfers string from a byte address into a character array.</td>
</tr>
<tr>
<td>DSAPUT</td>
<td>Transfers a character string value to a given address in BUILDER memory.</td>
</tr>
<tr>
<td>GETSCENV</td>
<td>This routine takes a relation name and a screen variable name from the CALL PROCEDURE line, looks up the relation and its associated current scenario key field value in the scenario system extra data segment, and puts this value into the given screen variable. The routine is necessary to enforce scenario security in any BUILDER-based module.</td>
</tr>
<tr>
<td>GETVAR</td>
<td>Retrieves the (ASCII) contents of a screen variable specified by name and places them in a character variable.</td>
</tr>
<tr>
<td>IDTODD</td>
<td>More date routines identical to their UTLR counterparts. These duplicates are here in the SL MODCOR so they can be called by the BUILDER-called routines.</td>
</tr>
<tr>
<td>LMONTH</td>
<td>See Section 10.1 for a description of each.</td>
</tr>
<tr>
<td>MODCOR</td>
<td></td>
</tr>
<tr>
<td>MRKDAY</td>
<td></td>
</tr>
<tr>
<td>NWDATU</td>
<td></td>
</tr>
<tr>
<td>NWDAT</td>
<td></td>
</tr>
<tr>
<td>PREPREPT</td>
<td>A preprocessor for RELATE EXECUTE files giving authors of such files the capability to enforce scenario security. Preprept opens and reads a file named on its CALL PROCEDURE line, echoing the file's records to a temporary file. It searches each record for instances of &quot;[relation.group]&quot;, looks up each such relation name found in the scenario system extra data segment, and substitutes the appropriate key value between (and including) the brackets. Thus selections can be given to limit the data returned to that of a particular scenario, without knowing which scenario in advance.</td>
</tr>
<tr>
<td>PUTVAR</td>
<td>Like getvar, but writes the contents of a FORTRAN character variable to the address of a screen variable specified by name.</td>
</tr>
<tr>
<td>SPSUSP</td>
<td>Suspends the current BUILDER process and activates its father. Useful for any module which it is desirable to put on &quot;hold&quot; (as opposed to termination by the BUILDER EXIT command) when the user returns to the command system, as the DBU is.</td>
</tr>
</tbody>
</table>
unspecified length which contains pointers into the BUILDER memory map, and an array of addresses and lengths which allow access to the interpreted text of the CALL PROCEDURE line.

In order for a BUILDER-called FORTRAN routine to be truly useful, there must be a means for passing data between the screen and the routine. The transfer can be done by file/relation i/o, but this is clumsy. Much more convenient is copying of data between BUILDER variables and variables local to the routine.

The PUTVAR and GETVAR utilities make use of the information in the second formal parameter to implement such a capability. The author of the FORTRAN routine need only know the name(s) of the screen variable(s) to/from which data is to be transferred. Note that BUILDER stores all data in an ASCII format, regardless of the type declaration in the screen, so type conversion will be necessary within the routine for numeric data. See the abstracts of these routines for specifics about the table of pointers into the BUILDER memory map if you are interested in that.

A more primitive but occasionally useful capability is provided by accessing the text of the CALL PROCEDURE line. This can be done by proper use of the DSARTRIM routine. DRTRIM and DLTRIM are useful in parsing the string extracted through use of dsartrim. See the code of the preprept or getsenv routines for examples of how this is done.

10.3.2 Scenario Security Enforcement Assistance

One of the most serious problems facing the designer of a BUILDER-based ALIAS module is the matter of scenario security. Like any other module, these must not access or change data for scenarios other than the user's current one, and no modifications can be allowed to data in relations that have only indirect access status for the given scenario.
In a FORTRAN module use of the DBIF to open and operate on relations automatically places the proper scenario field key value for each open relation in the /scenar/ common block, making it fairly easy to construct selections or point/read strategies which return only the proper data. However, none of these facilities are available from BUILDER.

The GETSCENV and PREPREPT routines solve these problems by extracting scenario field key values from the scenario system extra data segment in the same way that DBIF routines do.

GETSCENV takes a relation name and a screen variable name and returns the scenario field key value for that relation for the current scenario into the screen variable. This allows the screen designer to construct selects and point/read strategies in a way that maintains scenario security.

Designers of reports in the form of RELATE EXECUTE files, run from BUILDER, are able to enforce scenario security by use of the PREPREPT routine. The problem again is to construct a WHERE clause of the form WHERE SCENARIO="key_value", where the key_value is the one appropriate to the given relation and scenario. When preprept is available, this can be done reliably by substituting the phrase [relation.group] for key_value. Given the EXECUTE file name, preprept will read the file, echoing to a temporary named DBURTEMP. Whenever it encounters a relation name in brackets, it will search the scenario system extra data segment for the key_value currently appropriate for that relation, and will substitute that value for the bracketed expression. The screen can then EXECUTE DBURTEMP.

10.3.3 File Management

The DBU file management subsystem was discussed in detail in Section 8.4.3.2. It allows a BUILDER-based module to use a large number of relations simultaneously by operating multiple RELATE son processes. The subsystem was designed in a manner
that allows it to be used by several concurrently existing
BUILDER processes, the only restriction being that each use a
unique value for the SCREENSYS Job Control Word.

The subsystem consists of the CURINI and CURSWP routines
for relation/partition management, and the ABTRNS, BGTRNS, and
DOTRNS routines for global transaction management.

10.3.4 Process Handling

BUILDER-based modules which it is desirable to have exist
permanently (in a suspended state) when the user is exercising a
different part of the system may call the SPSUSP routine to
suspend themselves without terminating. Note that the process
creation/activation logic must also be properly arranged in the
mrump routine.

10.3.5 Other Capabilities

A quite DBU-specific utility, CALCDATE, was created to
perform schedule date recalculations for the PROJ_NC_SKED screen.
Although unlikely to be of use to other modules, a large number
of date utility routines were duplicated (from UTLR) to serve
calcddate. These may prove useful to screens with heavy date-
processing requirements.

10.3.6 Abstracts for SL Routines

Only abstracts for routines not duplicated from UTLR appear
in this section. See Section 10.1 for descriptions of the date
utilities (and (d)rtrim and (d)ltrisa) found in sl.pub.
**ABTRNS**

*CONTROLL SEGMENT=DSA*

**SUBROUTINE abtrns(cursor,table,pointr)**

**C**

*** FORMAL PARAMETER DECLARATIONS ***

integer cursor(50), table(1), pointr(4)

**C**

*** ABSTRACT ***

**C**PURPOSE  Does a relate ABORT TRANSACTION on all active
RELATE son processes EXCEPT that started up by the builder.

**C**AUDIT HISTORY

MSCarey  09-jan-83  AUTHOR

**C**FORMAL PARAMETERS

Cin  all arguments from builder CALL PROCEDURE facility

**C**COMMON BLOCKS

none

**C**CALLER DBU via CALL PROCEDURE

**C**METHOD

C  Retrieve the cursor storage data segment, which contains
ids of all active relate son processes in words 51-100, and
pointers to the locations of cursors open on each of these
processes in words 101-150.  In each case, a value of 0 terminates
the list of data for active sons.  Retrieves the info for each of
these cursors in turn and gives the ABORT TRANSACTION command.

**C**LOCAL VARIABLES

C  procs  son process ids (1-50), pointers to cursors (51-100)

**C**

**10-218**
**BUILD CALLABLE UTILITY ROUTINES IN THE SL**

```
C   BGETRNS  -----------------------------
*control segment=dsa
      SUBROUTINE bgtrns(cursor,table,pointr)
      *** FORMAL PARAMETER DECLARATIONS ***
      INTEGER cursor(50),table(1),pointr(4)
      *** ABSTRACT ***
C* PURPOSE Does a relate BEGIN TRANSACTION on all active
C   RELATE son processes EXCEPT that started up by the builder.
C* AUDIT HISTORY
C   MSCarey  09-jan-83  AUTHOR
C* FORMAL PARAMETERS
C   Cin    all arguments from builder CALL PROCEDURE facility
C* COMMON BLOCKS
C   none
C* CALLER DBU via CALL PROCEDURE
C* METHOD
C   Retrieve the cursor storage data segment, which contains
C   ids of all active relate son processes in words 51-100, and
C   pointers to the locations of cursors open on each of these
C   processes in words 101-150. In each case, a value of 0 terminates
C   the list of data for active sons. Retrieves the info for each of
C   these cursors in turn and gives the BEGIN TRANSACTION command.
C* LOCAL VARIABLES
C   proc   son process ids (1-50), pointers to cursors (51-100)
C**
```
**BUILD-CALLABLE UTILITY**

**ROUTINES IN THE SL**

C **CALC DATE** ************************************************************

*CONTROL segment-dsa

**SUBROUTINE** calcdate(cursor,table,pointr)

C*  *** FORMAL PARAMETER DECLARATIONS ***

integer cursor(50),table(41),pointr(4)

C*  *** ABSTRACT ***

C#PURPOSE  Implements the ESC R function for projected
C new construction schedules DBU screen; recalculates
C schedule dates using planning factors.

C#AUDIT HISTORY
C  MSCarey  02-apr-84 AUTHOR

C#FORMAL PARAMETERS
Cin  cursor  current DBU cursor
Cin  table  primary builder memory table
Cin  pointr  pointers to call procedure line text

C#COMMON BLOCKS
C  none

C#CALLER DBU, screen PROJ_NC_SKED

C#METHOD
C  Get the test from the call procedure line; first argument is
C  name of variable which is basis date; second argument is basis
C  date.
C  Load planning factors from builder memory.
C  Call ascdaysl for each date.
C  Place the new date in the proper variable in builder memory.

C#LOCAL VARIABLES
C  maxdat  max number of schedule dates processable
C  basind  index number of the basis date
C  basnam  name of the basis date
C  basdat  value of the basis date, ddate format
C  timunt  time units specification
C  dlist  list of dates to be processed, in ascending order
C  plist  list of planning factors, in ascending order
C  day  ddate representation of each new date
C  datnam name of each date to process

C**
CURINI

CONTROL segment=dsa,usinit

SUBROUTINE curini(cursor,table,pointr)
   integer cursor(S0),table(1),pointr(4)

*** ABSTRACT ***
PURPOSE
Initializes a set of cursors for use by the data entry system. Works in concert with CURSWP to. Allows use of multiple RELATE processes as sons of the builder.

Compiled code resides in SL.PUB

AUDIT HISTORY
MSCarey 30-sep-83 AUTHOR

PARAMETERS
Cin cursor cursor array used by screen system
Cin table global data storage table for screen system
Cin pointr pointers to argument from call

COMMON BLOCKS

nonedata segment format is:
location 0-9 index of cursor now in use by system by builder USE cursor id number
location 50-10000 by 50's: cursor data arrays

CALLER CRI builder application files

METHOD
Routines resident in an SL may not have global data declarations. The screen system multiple cursor facility simulates global storage for the cursors by using an extra data segment. This routine initializes that data segment.

The id of the data segment WAS taken from the argument supplied on the CALL PROCEDURE line in the application file. This argument must be numeric >0 and <32767.

A bug in the builder now prevents this. JOB CONTROL WORDS are currently used to communicate the id number of the cursor desired, the id number of the USE cursor to be swapped into, and the id number of the son process to use.

The routine does not actually initialize any cursors; this is done by CURSWP when it detects a 50-word data segment area which is not yet initialized. This routine writes codes into a word of each cursor area which tell CURSWP that no rcinitx call has yet been done. A 0 is placed in word 48, which RELATE uses to store son process id's in. This word will never be 0 once rcinitx has been called for a cursor.

LOCAL VARIABLES
numcur number of cursors usable by system
newcur,lcurs array of cursors to be initialized and stored
inuse cursor currently in use by system
iarg,carg,arg argument from call in various forms
remaining arguments are for intrinsic calls
BUILDER-CALLABLE UTILITY
ROUTINES IN THE SL

C CURSWP

$CONTROL segment=dsa

SUBROUTINE curswp(cursor, table, pointr)

C* *** FORMAL PR DECLARATIONS ***

  integer cursor(50), table(1), pointr(4)

C* *** ABSTRACT ***

C PURPOSE   Swaps the cursor currently in use by the screen
C system into cursor memory and brings in the cursor
C requested in the argument attached to the CALL PROCEDURE call
C to this routine (NOW READS JCW). See CURINI. Compiled code
C resides in SL.PUB

C Audit History
C MSCarey 30-sep-83 AUTHOR

C* FORMAL PARAMETERS
C  Currently in use by screen system
C
C  table  screen system global memory table
C
C  pointr  pointers to argument of CALL PROCEDURE

C* COMMON BLOCKS
C
C  none

C* CALLER CRI builder application files

C* METHOD
C
C  Parse the argument, whose format is D.C, where D is the
C  id of the data segment specified in a dcursorininit call,
C  and C is the index of the cursor which the application wishes
C  swapped in for its use.

C
C  Retrieve the SCREENSYS, NUMSWAP, CURSORNUM, AND CURSORPROC
C  Job Control Words, which specify the cursor memory data
C  segment id, the id of the builder USE cursor to be swapped,
C  the id number of the cursor to be swapped into 'numswap',
C  and the son process id code to be given to rdbinitx if
C 'cursonum' is not yet initialized.

C
C  Get the index of the data segment, swap out the current cursor,
C  and swap in the one desired.

C**
SUBROUTINE dotrns(cursor, table, pointr)
*** FORMAL PARAMETER DECLARATIONS ***
integer cursor(50), table(1), pointr(4)

*** ABSTRACT ***
PURPOSE  Does a relate COMMIT TRANSACTION on all active
RELATE son processes EXCEPT that started up by the builder.

AUDIT HISTORY
MSCarey       09-jan-83   AUTHOR

FORMAL PARAMETERS
all arguments from builder CALL PROCEDURE facility

COMMON BLOCKS
none

CALLER DBU via CALL PROCEDURE

METHOD
Retrieve the cursor storage data segment, which contains
ids of all active relate son processes in words 51-100, and
pointers to the locations of cursors open on each of these
processes in words 101-150. In each case, a value of 0 terminates
the list of data for active sons. Retrieves the info for each of
these cursors in turn and gives the COMMIT TRANSACTION command.

LOCAL VARIABLES
proc   son process ids (1-50), pointers to cursors (51-100)
SUBROUTINE druned(cursor,table,pointr)

*** FORMAL PARAMETER DECLARATIONS ***
integer cursor(50),table(1),pointr(4)

*** ABSTRACT ***

C#PURPOSE  Runs HP editor as a son of the screen system.

C#AUDIT HISTORY
MSCarey  25-nov-83  AUTHOR

C#FORMAL PARAMETERS
Cin  cursor  relate cursor in use at time of call
Cin  table  screen system io table
Cin  pointr  pointers to call parameter info

C#CALLER  BUILDER procedure

C#METHOD
Call to system intrinsic CREATE

C#LOCAL VARIABLES
C  pin  son process id number
C  flag  argument to create; value of 1 causes screen reactivation when son terminates.

10-225
$CONTROL segment=dsa
  SUBROUTINE druntdp(cursor,table,poıntr)
C*    *** FORMAL PARAMETER DECLARATIONS ***
C*        integer cursor(50),table(1),poıntr(4)
C*
C*    *** ABSTRACT ***
C*PURPOSE  Runs TOP editor as a son of the screen system.
C*AUDIT HISTORY
C  MSCarey         25-nov-83 AUTHOR
C*FORMAL PARAMETERS
Cin   cursor  relate cursor in use at time of call
Cin   table   screen system io table
Cin   poıntr   pointers to call parameter info
C*CALLER BUILDER procedure
C*METHOD
C  Call to system intrinsic CREATE
C*LOCAL VARIABLES
C    pin      son process id number
C    flag    argument to create: value of 1 causes screen
C    reactivatioın when son terminates.
SUBROUTINE dsafetch(data,datalen,address,length)

C*  FORMAL PARAMETER DECLARATIONS ***
C*  integer datalen,data(datalen),address(1),length
C*  ABSTRACT ***

C#PURPOSE  Converts the pointer information provided by the
C  CRI screen application builder CALL PROCEDURE facility
C  into an integer argument string usable by fortran.

C#AUDIT HISTORY
C  MSCarey  28-sep-83 AUTHOR

C#FORMAL PARAMETERS
Cout  data  argument string, integer form
Cin   datalen max length of string in words
Cin   address array mapped onto stack location where
C     argument string is stored
Cin   length length of argument string in bytes
C#COMMON BLOCKS
C  none
C#CALLER dsapoint,dsacursorinit,dsausecursor

C#METHOD
C  Calling routines provide a target array (data) which this
C  routine transfers the argument into.
C  Calling routines receive the word address in the stack of the
C  argument data in an integer word. By denoting this as a call-
C  by-value argument (syntax) in the calling routine,
C  while causing dsafetch to think it is a normal call-by-
C  reference, the address array in this routine is mapped onto
C  the proper location in the stack.

10-227
SUBROUTINE dsagetc(data,address,length)
   character*1 data(length),address(length)
   integer length

*** ABSTRACT ***

C# PURPOSE  Converts the pointer information provided by the
C# CRI screen application builder CALL PROCEDURE facility 
C# into a character argument string usable by fortran.
C# AUDIT HISTORY 
C# MSCarey 28-sep-83 AUTHOR
C# FORMAL PARAMETERS
Cout data argument string, character form
Cin address array mapped onto stack location where
C passed argument string is stored
Cin length length of argument string in bytes
C COMMON BLOCKS
C none
C CALLER dsapoint,dsacursorinit,dsausecursor
C METHOD
C Calling routines provide a target array (data) which this 
C routine transfers the argument into.
C Calling routines receive the byte address in the stack of the 
C argument data in a character word. By denoting this as a call-
C by-value argument (syntax ) in the calling routine, 
C while causing dsagetc to think it is a normal call-by-
C reference, the address array in this routine is mapped onto 
C the proper location in the stack.

10-228
SUBROUTINE dsaput(data,datalen,address,length)

*** FORMAL PARAMETER DECLARATIONS ***

integer datalen,data(datalen),address(datalen),length

*** ABSTRACT ***

C* PURPOSE  Writes 'data' to the given address in builder memory.

C* AUDIT HISTORY

C  MSCarey   28-sep-83 AUTHOR

C* FORMAL PARAMETERS

Cout data argument string, integer form
Cin datalen max length of string in words
Cin address array mapped onto stack location where
C  argument string is stored
Cin length length of argument string in bytes

C* COMMON BLOCKS

C  none

C* CALLER dsapoint,dsacursorinit,dsausecursor

C* METHOD

C  Calling routines provide a target array (data) and an address.
C  The call-by-value/check=0 trick is used to make this routine
C  see the address as a fortran array into which it can write.
C  See also routine dsafetch.
BUILDER-CALLABLE UTILITY
ROUTINES IN THE SL

C GETSCENV ********************************************
$CONTROL segment-dsa
SUBROUTINE getscenv(cursor,table,pointr)
C* *** FORMAL PARAMETER DECLARATIONS ***
   integer cursor(50),table(1),pointr(4)
C* *** ABSTRACT ***
C* PURPOSE Takes a DB relation name and a target variable
C name as input, finds the scenario field value for that
C relation for the current scenario, and places the value
C in the target variable.
C*AUDIT HISTORY
C MScarey 17-mar-84 AUTHOR
C*FORMAL PARAMETERS
Cin cursor current builder cursor
Cin table builder memory map
Cin args pointer to file name to process
C*COMMON BLOCKS
C none
C*CALLER builder procedures
C*METHOD
C Use daafetch to get the argument text.
C Get the file name from the argument string.
C Look through the scenario extra data segment and get the
C proper field value.
C Get the target variable name.
C Use putvar to place the value into the given variable in
C builder memory.
C*LOCAL VARIABLES
C filenam name of report template file, with group suffix
C com command/filename string
C rec record read from input file
C sname file name read from scenario data segment
C scen scenario field value for given file
C name name of target file, as parsed from input record
C filin,filout MPE file numbers for input and output files
C**
C GETVAR ****************************
$CONTROL segment=dsa
SUBROUTINE getvar(table,varnam,value,valdim,len)
C* *** FORMAL PARAMETER DECLARATIONS ***
    integer valdim,table(41),len
    character*16 varnam,value*(valdim)
C* *** ABSTRACT ***
C#PURPOSE  Retrieves the value of a variable in builder memory.
C#AUDIT HISTORY
C  MSCarey  02-apr-84 AUTHOR
C#FORMAL PARAMETERS
Cin  table  builder memory map start
Cin  varnam  name of variable to look for
Cout value  value of variable found
Cout len  length of variable's storage area
C#COMMON BLOCKS
C  none
C#CALLER builder procedures
C#METHOD
C  see routine putvar
C**
SUBROUTINE preprept(cursor, table, pointr)

C* *** FORMAL PARAMETER DECLARATIONS ***
  integer cursor(50), table(1), pointr(4)

C* *** ABSTRACT ***
C* PURPOSE Takes a file name, opens and reads the file,
C* substituting proper scenario field values for every
C* occurrence of {file.group}, and writes the result to
C* temporary file OBURTEMP, overwriting it if there.
C* Preprocessor for RELATE report generation files, making
C* them produce output for the proper scenario.
C* This version serves the BUILDER.

C*AUDIT HISTORY
C  MSCarey 17-mar-84 AUTHOR
C*FORMAL PARAMETERS
Cin cursor current builder cursor
Cin table builder memory map
Cin args pointer to file name to process
C*COMMON BLOCKS
C  none
C*CALLER builder procedures
C*METHOD
C  Use dsafetch to get the file name text. Do the file opens,
C  quitting if can't find input file, and then process.
C*LOCAL VARIABLES
C  filnam name of report template file, with group suffix
C  com command/filename string
C  rec record read from input file
C  sname file name read from scenario data segment
C  scen scenario field value for given file
C  name name of target file, as parsed from input record
C  filin,filout MPE file numbers for input and output files

10-232
C PUTVAR *******************************************************************************
$CONTROL segment=dsa
   SUBROUTINE putvar(table,varnam,value,len)
C*  *** FORMAL PARAMETER DECLARATIONS ***
   integer table(41),len
   character*16 varnam
   character*(len) value
C*  *** ABSTRACT ***
C#PURPOSE  Finds the given variable location in builder memory
   and writes the given value there.
C#AUDIT HISTORY
   MSCarey  19mar-84  AUTHOR
C#FORMAL PARAMETERS
Cin  table  map to builder memory
Cin  varnam  name of variable in builder application
Cin  value  new value for varnam
Cin  len  length of new value
C#COMMON BLOCKS
C  none
C#CALLER CRI builder
C#METHOD
C  table(41) is word address of start of variable table, which
C  is a linked list with 19-word elements. Elements of interest
C  are (1): word address next cell; (10): byte pointer to fieldname
C  (2): length of field name; (9) word address of data area;
C  (19) length of data area in bytes; (3) len data words
C
C  liberal use of dsafetch. map vartab onto variable table, then
C  run down the linked list looking for a match on the variable
C  provided. When found, map ival onto its data area and set
C  to new value.
C#LOCAL VARIABLES
C  vartab  a 19-word builder variable linked list cell
C  ifld,fldnam variable name as taken from builder memory
C  valbuf,valchr  word-aligned buffer for new value
C#  

10-233
SUBROUTINE spsusp(cursor, table, pointr)

**FORMAL PARAMETER DECLARATIONS**

integer cursor(50), table(1), pointr(4)

**ABSTRACT**

**PURPOSE** When called from a screen application file, it suspends operation of the application rather than terminating its execution. Thus next call to application avoids initialization work. When application is not a son process, merely causes an abort.

**AUDIT HISTORY**

MSCarey 30-sep-83 AUTHOR

**FORMAL PARAMETERS**

begin cursor current screen system cursor

begin table screen system memory

begin pointr pointers to arguments on CALL PROCEDURE line

**COMMON BLOCKS**

begin none

**CALLER** screen application files

**METHOD**

begin Activates father process and suspends this one.

```
10.4 BUILDER UTILITY SCREENS

This section will present a few BUILDER subroutine screens which can be thought of as utilities by the generality of their purpose and their limited data structure requirements. The screens are MPECOMMAND, RUNEDITOR, RUNTDP, and SEARCH. They are currently used solely by the DBU; their code resides in the dbusubr.screens file.

The selection of these few screens as utilities is somewhat arbitrary---there are many more screens in the DBU targeted toward performance of specific tasks, and extractable for use by other screen systems. Examples are the various command processing utilities and the comment screen support utilities handling text data. Developers of large screen systems should peruse Section 8.4 both for other screens which might be useful and for general approaches BUILDER procedure design and implementation.

The screens are displayed in Figures 10-1 through 10-4.

The MPECOMMAND screen simulates the monitor by entering a loop in which the user is prompted with the standard ":" for commands. BUILDER is instructed to treat the user's input as an MPE command request by a line of the form ":%mpecom". MPE command execution from within a system is a nice feature, but its usefulness is somewhat limited by the fact that UDC's cannot be executed this way. The screen has a minimal data structure requirement: a single string buffer called MPECOM (should be at least 80 characters long, better 132).

The RUNEDITOR and RUNTDP commands bring up the HP standard editor and the TDP editor as son processes, automatically doing a SYSTEM $CANCEL to reset the control-Y trap and prompting for a screen rewrite (REFRESH must be given in the caller) when the editor returns control to BUILDER. The routines require two alphanumeric variables, Y and COMMAND.
Figure 10-1. MPECOMMAND Utility Screen

*** SCREEN MPECOMMAND

*** INITIAL

SCROLL "Please give an MPE command at the colon (no UDCs) or RETURN to do nothing."

MPECOM := ""
PROMPT ":", MPECOM
WHILE MPECOM ""
IGNORE ALL ERRORS
:%mpecom
IF $ERROR
   DISPLAY ":$errmsg"
endif
MPECOM := ""
PROMPT ":", MPECOM
ENDWHILE
SYSTEM $CANCEL
PROMPT "Hit RETURN to refresh screen.", MPECOM
COMMAND := ""
RETURN SCREEN
Figure 10-2. RUNTDP Utility Screen

*** SCREEN RUNTDP

*** INITIAL

SCROLL "Entering tdp editor..."
SCROLL ""
:RUN TDP.PUB.SYS
SYSTEM $CANCEL
PROMPT "Hit RETURN to refresh screen ",Y
COMMAND := ""
RETURN SCREEN
*** SCREEN RUNEDITOR

*** INITIAL

SCROLL "Entering HP editor..."

SCROLL ""

:RUN EDITOR.PUB.SYS

SYSTEM $CANCEL

PROMPT "Hit RETURN to refresh screen ",Y

COMMAND := ""

RETURN SCREEN
*** SCREEN SEARCH

*** INITIAL

COND := ""
NUM := $RDBINFO(200)
FLDNUM := 1
WHILE FLDNUM = NUM
LIST := $RDBINFO(201, FLDNUM)
FLDNAM := $ITEM(LIST, 1, 1)
IGNORE ERROR 7313
FLDVAL := %fldnam
IF (NOT $ERROR) AND FLDVAL "" AND FLDVAL "0"
FLDTYP := $ITEM(LIST, 3)
FLDFMT := $ITEM(LIST, 11)
FLDLEN := $ITEM(LIST, 5)
IF FLDTYP=1
COND := $CONCAT(COND, " %fldnam", ":", "%fldval" = "," AND")
FLDVAL := $CONCAT(FLDVAL, $SUBSTR(ZEES, 1, FLDLEN -$LENGTH(FLDVAL) ))
COND := $CONCAT(COND, " %fldnam", ":", "%fldval" = "," AND")
ELSEIF FLDTYP=4 AND $BITS(FLDFMT, 13, 3)=1
COND := $CONCAT(COND, " %fldnam", ":", "%fldval" = "," AND ")
ELSE
COND := $CONCAT(COND, " %fldnam", ":", FLDVAL, "," AND ")
Figure 10-4. SEARCH Utility Screen

ENDIF
ENDIF

FLDNUM := FLDNUM+1
ENDDO
IF COND=""
   DISPLAY "You must fill in the field values you wish a match found for."
   OK := 0
   RETURN SCREEN
ENDIF

COND := $SUBSTR(COND,1,$LENGTH(COND)-4)
SET OPTION QUOTES=NO
SELECT BY %novindex WHERE %cond %modeset
SET OPTION QUOTES:YES
IF NOT $FOUND
   DISPLAY "No match found."
   OK := 0
ELSE
   DISPLAY
   OK := 1
ENDIF
RETURN SCREEN

10-240
The SEARCH screen implements the DBU "S" command. It constructs a selection on the current partition which requires that records returned match all non-blank values of screen variables which match field names on the current path. This allows users to immediately jump to a record in a file if they know enough about the record to uniquely identify it.

A nice feature of the screen is its handling of alpha variables. Instead of requiring an exact match on these, the constructed selection allows a range of values spanning any trailing blanks in the value in the BUILDER variable. For example, if a DBU user wanted all schedule records for ships with class names beginning with "D", he could place "D" in the class field and give the "S" command. The resulting selection's WHERE clause would read WHERE SCENARIO="%scenario" AND CLASS>="D" AND CLASS<="Dzzzzzzzzz", which would produced the desired effect.

The screen's logic is based on a loop which builds up the WHERE clause by concatenation into a buffer. The loop runs over all fields on the current path. The number, names, and data types of the fields are discovered by $RDBINFO calls. Data type is important because string and date values must be delimited by quotes, while numbers must be undelimited. Phrases are added to the clause only for fields which have a non-blank value in the screen variable with the same name (an error results on the attempt to assign the field's value to the buffer FLDVAL if there is no such variable). If all the relevant screen variables are blank then COND will be blank and the routine will print an error message and return. Otherwise the selection is given and the OK flag variable is set to communicate whether at least one matching record was found to the caller. It is up to the caller to retrieve the variable.

SEARCH requires the COND, NUM, FLDNUM, LIST, FLDNAM, FLDVAL, FLDNUM, FLDFMT, FLDLEN, MODESET, NOW_INDEX, and OK variables already exist before it is called. Note that the BY
clause to be used in the selection and any hard-wired WHERE clause conditions must be placed in the NOW_INDEX and MODESET variables before SEARCH is called.

10.5 FORTRAN INCLUDE FILES

This Section presents the code of all ALIAS FORTRAN include files. These files, stored in the .incl group, contain ALIAS common block declarations. They also include some "parameter" blocks, i.e. sets of FORTRAN parameter statements which specify array dimensions and other static system information, and some DATA statements, particularly field lists for relations.

The source code for these global data structures is maintained in files separate from the source of routines to promote standardization and maintainability. See the discussions of the include methodology in Sections 2 and 6 for further information about this.

The include files are presented on the following pages in alphabetical order, often more than one to a page in order to conserve space. Note that their source code was current as of September, 1984.

10.6 MISCELLANEOUS SYSTEM RESOURCES AND THEIR CURRENT UTILIZATION

Resources which are difficult to categorize, but which are nevertheless very important to software developers, include locations in the /lprnts/ FORTRAN common block and usage of extra data segments.

Lprnts array locations are typically reserved for one routine or a few related routines. This way the diagnostic output that results from turning on a given lprnt is precisely targeted. Table 10-8 presents a list of the current usage of lprnts locations.
ALIAS FORTRAN INCLUDE FILES

FILE ADDSUB

1.00 C
2.00 C INCLUDE FILE ADDSUB
2.11 C ---common blocks for the ADDSUB environment utility program
2.12 C ---which supports maintenance of entries.doc and subkeys.doc
3.00 COMMON /entries/ module,source,object,entryn,subprg,
4.00 1 caltyp,prpose,author,cat,mkey
5.00 CHARACTER module*12,source*18,object*18,entryn*12,
6.00 1 subprg*12,caltyp*10,prpose*48,author*12,cat*12,mkey*12
7.00 PARAMETER elen=166
8.00 PARAMETER eflist=':MODULE,SOURCEFILE,OBJECTFILE,ENTRYNAME,SUBPROC'
9.00 IAM,CALLTYPE,PURPOSE,AUTHOR,CATEGORY,MAINKEY:'
10.00 PARAMETER efile=':ENTRIES.DOC:'
11.00 INTEGER ientry(1)
12.00 EQUIVALENCE (module,ientry)
13.00 C
14.00 COMMON /subkey/ entrys,keywrd
15.00 CHARACTER*12 entrys,keywrd
16.00 PARAMETER slen=24
17.00 PARAMETER sflist=':ENTRY.KEYWORD:'
18.00 PARAMETER sfile=':SUBKEYS.DOC:'
19.00 INTEGER isubk(1)
20.00 EQUIVALENCE (entrys,isubk)
21.00 C
22.00 COMMON /escrs/ ecrs,scr,dauth
23.00 INTEGER ecrs,scr
24.00 CHARACTER*12 dauth
25.00 C
FILE ASGN

1.00 C include file /ASGN/ -- MANUAL ASSIGNER DATA BLOCKS
   SEE FILE ASGNDESC.INCL FOR DOCUMENTATION

3.00 C *** PARAMETER lnasgn=30+maxper+2*maxyds, lbasgn=2+6+2*maxper
4.00     PARAMETER mperh = 20, maxyds= 99, maxper= 208, lcasgn= 1240,
5.00     1 inasgn= 436, lbasgn=426, cundef=-9999,
6.00     2 zdel = %5C, mcyds = 12, mcls = 12, mcls1=  11
7.00     PARAMETER qpfx=4,qcls=5,qyr=6,qjob=7
8.00 C
9.00     COMMON /casgn/ scname,perlbl, duratn,cpagep,spromt(2),apromt(16),
10.00    1 ydname(maxyds)
11.00    CHARACTER scname*16, perlbl*8,duratn*8,cpagep*2, cpageh*1,
12.00    1 ydname,mcyds,spromt,apromt1
13.00    INTEGER casgn(1)
14.00    EQUIVALENCE (cpagep,cpageh), (scname,casgn)
15.00 C
16.00    COMMON /casgn/ idurat,ordmod,fdate, npagev,npageh, npromt, numper,
17.00    1 numyds,asntot,grdtot,lmore ,topyd ,lowyd ,topidx,lowidx,
18.00    2 freptr,recalc,ydcalc,mpageh,prompt,fiyear,
19.00    3 nvruse,asnsav,asncas,kpad(6),
20.00    4 supen(maxper),firstp(maxyds),numasn(maxyds),cursjd
21.00    INTEGER idurat,ordmod,fdate, npagev,npageh, npromt, numper, numyds,
22.00    1 asntot,supen,grdtot,topyd ,lowyd ,topidx,lowidx,freptr,
23.00    2 firstp,numasn,ydcalc,mpageh,fiyear,nvruse,asncas,kpad,
23.10    3 cursjd
24.00    LOGICAL lmore ,recalc,prompt,asnsav
25.00 C
26.00    COMMON /uasgn/ uasnr,uasnh,uasnc,inasn, outasn,
27.00    1 prnter,autorf,maline
28.00    INTEGER uasnr,uasnh,uasnc,inasn, outasn, maline
29.00    LOGICAL prnter,autorf
30.00 C
31.00    COMMON /basgn/ nextp,zshpcl,shpord,sumasn,valasn(maxper),
32.00    1 codasn(maxper)
33.00    INTEGER nextp,zshpcl(6),valasn,codasn,sumasn
34.00    INTEGER*4 shpord
35.00    CHARACTE R shpcl*mcls
36.00    EQUIVALENCE (shpcl,zshpcl)
37.00 C
FILE ASGNDESC

1.00 C DESCRIPTION OF PARAMETERS IN /asgn/
2.00 C mperh - maximum number of display periods (20)
3.00 C maxyds - maximum number of yards (99)
4.00 C maxper - maximum number periods
5.00 C lcasgn - length of /casgn/ (char)
6.00 C lnasgn - length of /nasgn/ (*2 words)
7.00 C lbasgn - length of /basgn/ (*2)
8.00 C zdel - nonprinting delimiter character
9.00 C mclis - shpcis character length
10.00 C mclis- 11, the length of the allowable class input
11.00 C mcyds - ydname character length
12.00 C
13.00 C
14.00 C IN /casgn/ ***** character PART OF ASSIGNMENTS WORKING COMMON
15.00 C sname - scenario name (*16)
16.00 C perlbl - period label (*8) [ie. "PERIOD:"
17.00 C duratn - period duration (*8) [ie. "YEARS"]
18.00 C cpagenh - horizontal page character (ie. "A")
19.00 C spromt - short prompt characters (*2)
20.00 C apromt - prompt sequence (*16 max)
21.00 C Ydname - name of each yard (*12)
22.00 C
23.00 C IN /nasgn/ ***** NUMERIC PART OF ASSIGNMENTS WORKING COMMON
24.00 C idurat - period duration index ([1-6]: [FYr CYr Qtr Month Week Day])
25.00 C ordmod - the first 5 bits (from the right) are currently in use.
26.00 C 1: ship class ordering; off=alpha, on=input
27.00 C 2: ship yard ordering; off=alpha, on=input
28.00 C 3: use historical section of database
29.00 C 4: use current section of database
30.00 C 5: use projected section of database
31.00 C fidate- first qtr/wk/day in date row
32.00 C npagev- vertical page number
33.00 C fiyear - starting year
34.00 C asnsav - .T. if last command modified the direct access file
35.00 C npagenh- horizontal page number
36.00 C npromt - number of characters in apromt
37.00 C numper- number of periods defined
38.00 C numyds- number of yards defined
39.00 C asntot- total assignments count
40.00 C grdtot- total ships count
41.00 C cursjd- cursor open on the job description file; for ckpf
42.00 C topyd - loc of top yard displayed
### ALIAS FORTRAN INCLUDE FILES

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>43.00 C</td>
<td>lowyd - loc of lowest yard displayed</td>
</tr>
<tr>
<td>44.00 C</td>
<td>topidx- top index of top yard</td>
</tr>
<tr>
<td>45.00 C</td>
<td>lowidx- low index of low yard</td>
</tr>
<tr>
<td>46.00 C</td>
<td>recalc- .T. if page recalc needed</td>
</tr>
<tr>
<td>47.00 C</td>
<td>ydcalc- yard index for recalc.</td>
</tr>
<tr>
<td>48.00 C</td>
<td>freptr- free chain pointer</td>
</tr>
<tr>
<td>49.00 C</td>
<td>pageh- max horizontal page number</td>
</tr>
<tr>
<td>50.00 C</td>
<td>prompth- T:interactive, F:command</td>
</tr>
<tr>
<td>51.00 C</td>
<td>asncas- bits describing uppercases to be done after input</td>
</tr>
<tr>
<td>52.00 C</td>
<td>bit 1: uppercase the first char of input yard names</td>
</tr>
<tr>
<td>53.00 C</td>
<td>2: uppercase entire input yard names</td>
</tr>
<tr>
<td>54.00 C</td>
<td>3: uppercase the first char of input classes</td>
</tr>
<tr>
<td>55.00 C</td>
<td>4: uppercase entire input classes</td>
</tr>
<tr>
<td>56.00 C</td>
<td>Sumper- total ships by period</td>
</tr>
<tr>
<td>57.00 C</td>
<td>Firstp- assignment chain heads</td>
</tr>
<tr>
<td>58.00 C</td>
<td>Numasn- total number of assignment rows, by yard</td>
</tr>
<tr>
<td>59.00 C</td>
<td>nvruse- the lowest unused record</td>
</tr>
<tr>
<td>60.00 C</td>
<td></td>
</tr>
<tr>
<td>61.00 C</td>
<td>IN /basgn/ ***** BUFFER PART OF ASSIGNMENTS WORKING COMMON</td>
</tr>
<tr>
<td>62.00 C</td>
<td>nextp - next buffer in chain</td>
</tr>
<tr>
<td>63.00 C</td>
<td>shpcls- ship class (*12)</td>
</tr>
<tr>
<td>64.00 C</td>
<td>sumasn- total assignments in this buffer</td>
</tr>
<tr>
<td>65.00 C</td>
<td>Valasn- the value of each assignment</td>
</tr>
<tr>
<td>66.00 C</td>
<td>Codasn- code for each assignment</td>
</tr>
<tr>
<td>67.00 C</td>
<td></td>
</tr>
<tr>
<td>68.00 C</td>
<td>IN /uasgn/ ***** UNSTORED PART OF ASSIGNMENTS WORKING COMMON</td>
</tr>
<tr>
<td>69.00 C</td>
<td>uasnr - unit number for direct access file records</td>
</tr>
<tr>
<td>70.00 C</td>
<td>uasnh - unit number for help file</td>
</tr>
<tr>
<td>71.00 C</td>
<td>uasnc - unit for /asgn/ save</td>
</tr>
<tr>
<td>72.00 C</td>
<td>inasn - input logical unit number</td>
</tr>
<tr>
<td>73.00 C</td>
<td>outasn- output unit number</td>
</tr>
<tr>
<td>74.00 C</td>
<td>the first input character is treated as a default</td>
</tr>
<tr>
<td>75.00 C</td>
<td>and prints out as a blank (right now, New)</td>
</tr>
<tr>
<td>76.00 C</td>
<td>printer- .T. if asnclr should act like a line printer</td>
</tr>
<tr>
<td>77.00 C</td>
<td>autorf- .T. if auto-refresh is on (see asnref)</td>
</tr>
<tr>
<td>78.00 C</td>
<td>maline- maximum number of buffer lines which fit on screen</td>
</tr>
<tr>
<td>79.00 C</td>
<td></td>
</tr>
</tbody>
</table>

---

10-246
ALIAS FORTRAN INCLUDE FILES

FILE ASHLDR

1.00 C include file ashldr
2.00 common /ashldr/hldval(maxper),hldcod(maxper)
3.00 integer hldval,hldcod
4.00 C
5.00 C holding buffer for per-period assignments and series codes
6.00 C used by outbound to transfer to extended memory
7.00 C
ALIAS FORTRAN INCLUDE FILES

FILE ASJD

1.00 C: include file asjd
2.00 common /asjd/ndesc,jclas,jyard,jjtyp,jstyp,jcust,jgrp,
3.00 ljcomm,jmethd,jda,jaa,jas,jsk,jkl,jld,jdadd,jdawd,jtunt
4.00 dimension jclas(asmpt),
5.00 1      jyard(asmpt),jjtyp(asmpt),jstyp(asmpt),
6.00 2      jcust(asmpt),jgrp(asmpt),ljcomm(asmpt),
7.00 3      jmethd(asmpt),jda(asmpt),jaa(asmpt),jas(asmpt),
8.00 4      jsk(asmpt),jkl(asmpt),jld(asmpt),
9.00 5      jdadd(asmpt),jdawd(asmpt),jtunt(asmpt)
10.00 character jclas*10,jyard*8,jjtyp*6,jstyp*6,jcust*8,jgrp*10
11.00 character*6 jmethd,jtunt
12.00 integer ndesc,ljcomm,jda,jaa,jas,jsk,jkl,jld,jdadd
13.00 integer*4 jdawd
14.00 integer jdalin(asmpt,5)
15.00 equivalence (jdalin,jaa(1))
16.00 C
17.00 C     job descriptions buffer; holds one per class in a
18.00 C     complexity group
19.00 C
20.00 C     jdalin alternative address for building period intervals
21.00 C     ndesc    number of description records currently held
22.00 C     jclas    class of job desc
23.00 C     jyard    yard it applies to, or ANY
24.00 C     jjtyp    job type it applies to
25.00 C     jstyp    series type it applies to
26.00 C     jcust    customer for this job
27.00 C     jgrp     complexity-group of this job
28.00 C     ljcomm   commissioning number
29.00 C     jmethd  construction method
30.00 C     jda     design to award time
31.00 C     jaa     appropriation to award time
32.00 C     jas     award to start
33.00 C     jsk     start to keel time
34.00 C     jkl     keel to launch time
35.00 C     jld     launch to delivery time
36.00 C     jdadd   days added to a ship's life by this job
37.00 C     jdawd   default award date in a year; for dpsmode-awards
38.00 C     and time units = years
39.00 C     jtunt   time units schedule intervals given in
ALIAS FORTRAN INCLUDE FILES

FILE ASNOCR

8.00 C: include file asnocr
9.00 common /asnocr/jomain,jodprj,joprg2,joflag,jdesc,
10.00 1 jodol,jolbr,joemp,jomr,jomd,jocom,
11.00 2 jodolf,jolbrf,joempf,jomrf,jomdf,jocomf
12.00 integer jomain,jodprj,joprg2,joflag,jdesc,
13.00 1 jodol,jolbr,joemp,jomr,jomd,jocom
14.00 logical jodolf,jolbrf,joempf,jomrf,jomdf,jocomf
15.00 C
16.00 C CURSOR FOR
16.10 C jomain ncjodat.projj main file when copy algorithm used
17.00 C jodprj projected jobs (ncjodat.projj
18.00 C joprg2 same
19.00 C joflag same, select by scenario,yard,class,jobtyp
20.00 C where flag="YES"
21.00 C jdesc job descriptions (ncjdat.descj
22.00 C jodol ncjodol.projj
23.00 C jolbr ncjolbr."
24.00 C joemp ncjoemp."
25.00 C jomr ncjomr.""
26.00 C jomd ncjomd."
27.00 C jocom ncjocom."
28.00 C OPEN STATUS FLAGS FOR EIGHT OF ABOVE CURSORS
29.00 C jodolf
30.00 C jolbrf
31.00 C joempf
32.00 C jomrf
33.00 C jomdf
34.00 C jocomf
35.00 C
ALIAS FORTRAN INCLUDE FILES

FILE ASNVL0

1.00 C
2.00 C include file /asnvl0/
3.00 C PARAMETER mvcls=200, mvyds= 99, mcvcls=10, mcvyps= 8,
4.00 1 mcdchr=10, mjtchr=10, mcjt = 6, mccd = 6
5.00 COMMON /asnvl0/ vldcls(mvcls),nvcls,vldyds(mvyds),nvyds,
6.00 1 jtidef,jtchar,ljtchr,jtname(mjtchr),jtvlid(mjtchr),
7.00 2 jttype(mjtchr),cdidef,cdchar,lcdchr,cdname(mccdchr)
8.00 CHARACTER cdchar*mcdchr,jtchar*mjtchr,cdname*mccd,jtname*mcjt
9.00 CHARACTER jtype,vldcls*mcvcls,vldyds*mcvyps
10.00 INTEGER cdidef,jtidef,nvcls,nvyds,lcdchr,ljtchr
11.00 LOGICAL jtvld
12.00 C holds lists of valid classes/yards from liston (VALCLS,VALYDS)
13.00 C vldcls- list of valid shipclasses, of length nvcls
14.00 C vldyds- list of valid yards, of length nvyds
15.00 C and holds legal code characters and their translations (.LEGALS):
16.00 C cdchar- character CoDes, like Lead, etc.. of length *lcdchr
17.00 C cdname- names corresponding to character codes; length *lcdchr
18.00 C cdidef- default series code location
19.00 C jtchar- character jobtypes, like New, Repair, etc. : *lcdchr
20.00 C jtname- names corresponding to the job type codes; length *lcdchr
21.00 C jtidef- default jobtype code location
22.00 C jtype- True if menu system (liston) gives this type as valid
23.00 C jtype- N for job types with data in NC relations, 0 for REpairs
24.00 C
ALIAS FORTRAN INCLUDE FILES

FILE ASOPRM

36.00 C include file asoprm
37.00 parameter tupunt=21, asomxc=50, asomxg=20
38.00 parameter asmshp=200, asmpft=10, asmndt=6
38.10 parameter exunit=24, exout=25
39.00 common /asoprm/asucur, asuhis, asamod, asabas, asdbas,
40.00 logical astunt, asfstd, aslstd, memid, perlen
41.00 logical asucur, asuhis
42.00 integer perlen(300)
43.00 integer memid, astunt, asamod, asabas, asdbas
44.00 integer*4 asfstd, aslstd
45.00 C general assigner outbound variables and parameters
46.00 C
47.00 C PARAMETERS
48.00 C asomxc maximum new-construction classes in a single yard
49.00 C asomxg maximum classes in a single complexity group in a yard
50.00 C asmshp maximum ships record buffer can hold at once
51.00 C asmpft max planning factor tuples for one yard-class-jtyp combo
52.00 C tupunt unit number for tupfil
52.10 C exunit unit number for RELATE execution file newhul
53.00 C asmndt number of schedule dates (award, start, etc) for nc jobs
54.00 C VARIABLES
55.00 C asuhis true if historical data brought in on inbound
56.00 C asucur true if current data brought in on inbound
57.00 C perlen length of each display period in days
58.00 C asamod schedule adjust basis mode: 1=none, 2=class, 3=cmplx grp
59.00 C asabas adjust basis date code: 1=award, ..... 5=delivery
60.00 C asdbas display basis date code: 1=award, ..... 5=delivery
61.00 C astunt time unit code: 1=fyear, 2-cyear, ..... 6=days
62.00 C asfstd first day of first bufasn period (clarified ddate)
63.00 C memid id code for extended memory segment
ALIAS FORTRAN INCLUDE FILES

FILE ASRBUF

64.00 C: include file asrbuf
65.00 C NOTE: asoprm must be included above
66.00 common /asrbuf/rnptr,rlptr,rclas,rcode,rdispd,radjd,rfirst,rlast,
67.00 1 rperd
68.00 common /asrcls/rclasl,nscl
69.00 integer rnptr(asmshp),rlptr(asmshp),rclas(asmshp),rcode(asmshp)
70.00 integer rperd(asmshp)
71.00 integer*4 rdispd(asmshp),radjd(asmshp)
72.00 integer*4 rfirst(asmshp),rlast(asmshp)
73.00 integer rclasl(asomxg),nscl(asomxg)
74.00 C
75.00 C assigner outbound I ship--I record buffer
76.00 C holds variables used in generating key sched dates
77.00 C for each ship in a complexity-group in a yard
78.00 C rlptr pointer to last ship in same class
79.00 C rnptr pointer to next ship in same class
80.00 C rclas id number of class of ship i; references rclasn
81.00 C rcode code number of class of ship i; e.g. 'LEAO'
82.00 C rdispd relate*4 format date that display of ship was based on
83.00 C radjd date that schedule adjustment of ship is to be based on
84.00 C rfirst first date that radjd may be set to
85.00 C rlast last date that radjd may be set to
86.00 C rperd period ship's displaydate falls in
87.00 C -rclasn- for character name of each class see hidcls
88.00 C nscl number of ships in each class
89.00 C rclasl pointer to first record holding ship of class j
90.00 C

10-252
ALIAS FORTRAN INCLUDE FILES

FILE ASTFR

1.00 C! include file astfr
2.00 C
3.00 parameter asdtst = 31
4.00 common/astfr/fscen,flas, fhull, fyard, fcom, fjtyp, fjstyp, fcust,
5.00 1 fcmthd, fapp, fa, start, fkeel, flaun, fdeliv, fcomm,
5.10 1 fdad,
6.00 2 fflag, forder, fhmap, fddat, fedat, feby
7.00 common/astup/tscen,tlclass,thull,tyard,tcom,tjtyp,tjstyp,tcust,
8.00 1 tcmthd, tappr, taward, tstart, tkeel, tlaun, tdeliv, tcomm,
8.10 1 tdad,
9.00 2 tflag, torder, thmap, tddat, tedat, teby
10.00 common/asvar/eorein
11.00 C
12.00 logical eorein
13.00 character fscen*12, tscen*12, flas*10, tlclass*10, fyard*8, tyard*8
14.00 character fjtyp*6, tjtyp*6, fcmthd*6, tcmthd*6, feby*8, teby*8
15.00 character fcust*8, tclust*8, fjtyp*6, tjstyp*6, fflag*4, tflag*4
16.00 integer fhull, thull, fcom, tcom, fdad, tdad
17.00 integer*4 fapp, tappr, fa, start, start
18.00 integer*4 fkeel, tkeel, flaun, tlaun, fdeliv, tdeliv, fcomm, tcomm
19.00 integer*4 forder, torder, fddat, tddat, fhmap, thmap, fedat, tedat
20.00 C
21.00 integer tfrali(60), tupali(60)
22.00 equivalence (fscen, tfrali), (tscen, tupali)
23.00 character=46 fld05(4)
24.00 C
25.00 C records for reading/writing of tupfil (tfr) and ncjodat
26.00 C the records are identical: each pair of variables will
27.00 C be described only once
28.00 C
29.00 C asdtst parameter giving location of award in aligned arrays
30.00 C fscen scenario name
31.00 C flas class name
32.00 C fhull hull number
33.00 C fyard yard name
34.00 C fcom commissioning number
35.00 C fjtyp job type
36.00 C fjstyp series type (lead, follow..)
37.00 C fcust customer
38.00 C fcmthd construction method
39.00 C fapp appropriation date
40.00 C faaward award date
41.00 C fa, start start date
ALIAS FORTRAN INCLUDE FILES

42.00 C  fkeel  keel date
43.00 C  flaun  launch date
44.00 C  fdeliv  delivery date
45.00 C  fdad  days added to the life of the ship by this commissioning
46.00 C  fflag  flag setting (YES, NO)
47.00 C  forder  asnorder value
48.00 C  fhmap  the bit map indicating which relations hold hardware
49.00 C  fflag  data for this ship
50.00 C  fddat  data date
50.10 C  fedat  entry date
50.20 C  fcomm  commissioning date
50.30 C  feby  name of user running assigner; entry_by
51.00 C  eoreln  End Of RELation. The end-of-file indicator for
53.00 C  ncjodat.proj  must be global.
54.00 C
ALIAS FORTRAN INCLUDE FILES

FILE BGPMTR

1.00 C
2.00 C include file bgpstr---BATTLE GROUP REPORT GENERATOR PARAMETERS
3.00   PARAMETER MXTYPE = 100
4.00   PARAMETER MXCHOICE = 20
5.00   PARAMETER MXFUNC = 50
6.00   PARAMETER MXMKUP = 400
7.00   PARAMETER MXGROUP = 20
20.00 C

FILE BGTITL

1.00 C
2.00 C include file bgtitl---BATTLE GROUP REPORT GENERATOR TITLE
3.00   CHARACTER SECTITL*20
4.00   COMMON/BGTITL/SECTITL
5.00 C   sectitl current battle group report section title
6.00 C
FILE CHLST

1.00 C  include file chlst---MNUG block; list processed choice menus
2.00 C
3.00 CHARACTER CMNLST*LNAME
4.00 INTEGER*4 PMPTR,CMPTR
5.00 INTEGER NCMENU
6.00 COMMON /CHLST/ NCMENU, CMNLST(MXMENU), CMPTR(MXMENU)
6.10 + ,PMPTR(MXMENU)
7.00 C

FILE CMENU

1.00 C  include file CMENU---MNUR block; data for current choice menu
2.00 C
2.50 CHARACTER MNTXT*OIMNAME, MTTXT*LLINE
3.00 INTEGER*4 IDOPT, OPTPTR, OPTTXT, MHPTR, OPHPTR
4.00 INTEGER OPSECI, IDMENU, NOPTIO, OPTTYP, MHLEN, OPHLEN
5.00 COMMON/CMENU/ IDMENU, MHLEN, NOPTIO, OPTTYP(IS), OPHLEN(IS),
6.00 1 OPTPTR(IS), IDOPT(IS), MHPR, OPHPTR(IS), OPTTXT(IS)
6.10 1 , MTTXT, MNTXT, OPSECI(IS)
7.00 C

Id number current menu, number lines menu level help text
number options on menu, type each option, number lines
help each option, id for each option (a menu id),
pointers to option rnproc call goto indexes for process
options pointer to menu help text,
pointers to option help text, pointer to option text
for menu, pointer to menu title text, pointer to menu name,
security index each option (indexes element of modnum
in /usr/prv/).
FILE COLUSE

53.00 C
54.00 C include file coluse
55.00 common /coluse/ clused(nltrcl,mx1typ)
56.00 logical clused
57.00 C
58.00 C This common block contains information about free space
59.00 C in list type relations.
60.00 C clused Dimensioned (# of columns in each list type relation) by
61.00 c (maximum number of list type relations), this array is
62.00 c a set of flags indicating which columns are in use
63.00 C (.true.).
ALIAS FORTRAN INCLUDE FILES

1.00 C include file comcfl
2.10 parameter mxcfly = 10
3.00 INTEGER CFRECS, INCFL, OUTCFL, CFLEVL, CFTREC
4.00 LOGICAL INUSE, BUILDN, CFECHO
4.10 CHARACTER*8 CFMUI, CFNAM
5.00 COMMON/COMFILE/ INCFL, OUTCFL, INUSE, BUILDN, CFECHO, CFLEVL
5.01 , CFRECS(mxcfly), CFTREC(mxcfly), CFMUI, CFNAM
5.10 C mxcfly maximum nesting level of executing command files
6.00 C incfl file # from which command file should be read
7.00 C outcfl file # to which command file should be written
8.00 C inuse true only if input is being read from incfl
9.00 C buildn true only if command file is begin built
10.00 C cfecho true if readln should echo input to screen
11.00 C cflevl current command file execution nesting level
12.00 C cfreces count of records read from/written to comfile
13.00 C cftrec number of records in file exec-ing at level cflevl
14.00 C cfmu1 menu id for command file exec-ing at cflevl=1
15.00 C cfnam1 name of command file exec-ing at cflevl=1
ALIAS FORTRAN INCLUDE FILES

FILE CONST

1.00 C
2.00 C The following are constant values... /CONST/
3.00 PARAMETER
4.00 1 cr = %15C, lf = %12C,
5.00 2 ff = %14C, large = %077777,
6.00 3 llarge = %17777777777J, pi = 3.141592654,
7.00 4 root2 = 1.414213562, eps = 1.0E-75,
8.00 5 xlarge = 1.0E+75, bell = %7C,
9.00 6 nullc = %0C, bs = %10C,
10.00 7 largec = %177C
11.00 C

FILE CURSRS

1.00 C
2.00 C include file cursrs
3.00 PARAMETER mcchn=22, mcrs=20, mcpth=12
4.00 COMMON /cursrs/ crschn(mcchn),crs(50,mcrs),crspth(mcrs),
5.00 1 crstyp(mcrs),crsidx(mcrs),crsxl(mcrs)
6.00 INTEGER crschn,crs,crstyp,crsidx,crsxl
7.00 CHARACTER crspth*mcpth
8.00 C crschn=cursor chain, crs =cursor pool
9.00 C crspth=cursor path name, crsxl =index calculate length
10.00 C crstyp=[0..3]: 0=not-in-use, 1=relation, 2= synonym, 3=select
11.00 C crsidx=cursor index pointer: 0=no new index, 0 otherwise
12.00 C
ALIAS FORTRAN INCLUDE FILES

FILE DEBUFF

1.00 C
2.00 C include file debuff
3.00 C parameter Data Entry Maximum Buffer Length, Number Buffers, Fields per relation
5.00 parameter demxbl=160,demxnb=15,demxfi=30
6.00 character*255 bflist(2,demxnb)
7.00 common /debuff/ bfpool(demxbl,demxnb),bufrel(demxnb),
7.10 1 bufpth(demxnb),
8.00 1 grpid(demxnb),gpstat(demxnb),gplsta(demxnb),
9.00 2 fldsta(demxfi,demxnb),decurs(demxnb),
10.00 3 bfnfld(demxnb),bfallo(demxnb),bflist
11.00 integer bfpool,grpid,fldsta,bfnfld,decurs
12.00 character*25 bufrel,bufpth
13.00 logical gpstat,gplsta,bfallo
14.00 C equivalencing
15.00 character*2 cbpool(160,demxnb)
16.00 real rbpool(80,demxnb)
17.00 equivalence (cbpool,bfpool),(rbpool,bfpool)
18.00 C data transfer buffers for the data entry module
19.00 C bfpool the buffers
20.00 C bufrel name of relation buffer holds data for
21.00 C grpid id number of entry group or set buffer belongs to
22.00 C bfstat buffer entry status; whether its data is to be sent
23.00 C bflist field list for each buffer

THIS WAS FOR OBSOLETE DATA UPDATING SYSTEM

10-260
ALIAS FORTRAN INCLUDE FILES

FILE ENVIRN

1.00 C include file envirn
2.00 C CHARACTER*8 GROUPN, RELGPN, starty, groupc*2
3.00 CHARACTER*8 GROUPN, RELGPN, starty, groupc*2
4.00 INTEGER LENGPN, ICCTCL(2), LENRLN, lengpc
5.00 CHARACTER*4 CCTCLR
5.10 LOGICAL develp
6.00 COMMON/ENVIRN/ GROUPN, RELGPN, LENRLN, lengpc, cctclr, starty,
7.00 1 groupc, lengpc, develp
8.00 EQUIVALENCE (ICCTCL(1), CCTCLR)
8.10 C ---system core status info, mainly for dev/prod versions cap
9.00 C groupn group name in which files are located
10.00 C lengrp length of group name in characters
11.00 C regpnp group in which relate files are located
12.00 C lenrln length of relgpn in characters
13.00 C cctclr clear screen control characters for terminal
14.00 C icctcl integer version of cctclr
15.00 C starty name of terminal type found or given at startup
16.00 C groupc character suffix for data base relation group names
17.00 c blank if production version; 'T' if development
18.00 C lengpn length of groupc contents
18.10 C develp true if development version is being run
19.00 C
ALIAS FORTRAN INCLUDE FILES

FILE FIELDS

1.00 C
2.00 C include file fields
3.00 C holds data statements for all includes of the 'rcrd##' type
4.00 data flcl01 /""MENUID,RELATION,COLUMN:"/
5.00 C
6.00 C

FILE FLCLASS

1.00 C
2.00 C include file flclass--list of ship class FLRP is dealing with
3.00 INTEGER NCLASS
4.00 CHARACTER CLIST*CNLEN(MAXCLAS)
5.00 COMMON/FLCLASS/ NCLASS, CLIST
6.00 C
FILE FLCONCH---parameters defining acceptable FLRP input keywords

1.00 C
2.00 C include file flconch
3.00  PARAMETER LKEY=8
4.00  PARAMETER LETOT=4
5.00  PARAMETER ETOT='ETOT'
6.00  PARAMETER LJOB = 3
7.00  PARAMETER JOB = 'JOB'
8.00  PARAMETER LEND = 3
9.00  PARAMETER END='END'
10.00 PARAMETER LEITOT=5
11.00 PARAMETER EITOT='EITOT'
12.00 PARAMETER LTYPE=4
13.00 PARAMETER TYPE = 'TYPE'
14.00 PARAMETER LBTO=4
15.00 PARAMETER BTOT='BTOT'
16.00 PARAMETER LSTART=5
17.00 PARAMETER START='START'
18.00 PARAMETER LPRGLB=5
19.00 PARAMETER PRGLB='PRGLB'
20.00 PARAMETER COMMA=','
21.00 PARAMETER LPARAN='(''
22.00 PARAMETER RPARAN=')'
23.00 PARAMETER LTITLE=5
24.00 PARAMETER TITLE='TITLE'
25.00 PARAMETER BLANK='
27.00 C
28.00  PARAMETER LSTOP = 4
29.00  PARAMETER STOP = 'STOP'
30.00  PARAMETER CONTINUE = '+'
FILE FLCONS

1.00 C
2.00 C include file flcons---data for FLRP access to schedule relations
3.00 C historical,current,projected construction relations
4.00 PARAMETER HISCNM='-NCJODAT.HISTJ-
5.00 PARAMETER CURCNM='-NCJODAT.CURRJ-
6.00 PARAMETER PROCNM='-NCJODAT.PROJJ-
7.00 PARAMETER CONSFL='-SCENARIO,CLASS,HULL,COMNUM,APPROP,AWARD,DELIVER,COMMISSION,DAYSADDED,DATADATE,ENTRY_DATE-
8.00 COMMON/FLCONS/hccur1,hccur2,cccurl,cccur2,pccurl,pccur2
9.00 PARAMETER CONSKY=
10.00 COMMON/FLCONS/hccur1,hccur2,cccurl,cccur2,pccurl,pccur2
11.00 character*12 cfscen,cfclas*10
12.00 integer cfhull,cfadlif,cfnumb, cfalin(26)
13.00 integer hccurl,hccur2,cccurl,cccur2,pccurl,pccur2
14.00 integer*4 cfcomd,cfappd,cfawdd,cfdeld,cfdatd,cfdatn
15.00 equivalence (cfalin(1),cfscen), (cfalin(7),cfclas)
16.00 + (cfalin(12),cfhull),(cfalin(13),cfnumb)
17.00 + (cfalin(14),cfappd),(cfalin(16),cfawdd)
18.00 + (cfalin(18),cfdeld),(cfalin(20),cfcomd)
19.00 + (cfalin(22),cfadlif)
20.00 + (cfalin(23),cfdatd),(cfalin(25),cfdatn)
21.00 character*12 ckscen,ckclas*10
22.00 integer ckhull, cknumb, ckalin(17)
23.00 integer*4 cfdatd,ckdatn
24.00 equivalence (ckalin(1),ckscen), (ckalin(7),ckclas)
25.00 + (ckalin(12),ckhull),(ckalin(13),cknumb)
26.00 + (ckalin(14),ckdatd),(ckalin(16),ckdatn)
27.00 integer*12 cffspin,cffspj
28.00 C -- data rekkrd for FLCONS (Force Level CONSTRUCTION) data
29.00 C
30.00 C
31.00 C hccur1 relate virtual cursor for historical construction
32.00 C relation, path 1
33.00 C hccur2 relate virtual cursor for historical construction
34.00 C relation, path 2
35.00 C cccur1 relate virtual cursor for current construction
36.00 C relation, path 1
37.00 C cccur2 relate virtual cursor for current construction
38.00 C relation, path 2
39.00 C pccur1 relate virtual cursor for projected construction
40.00 C relation, path 1
41.00 C pccur2 relate virtual cursor for projected construction
42.00 C relation, path 2

10-264
ALIAS FORTRAN INCLUDE FILES

43.00 C  c_scenr    a scenario id (f=fieldvalue,k=keyvalue)
44.00 C  c_clas     a ship class id
45.00 C  c_hull     hull number for the ship class
46.00 C  c_appd     ship's appropriation date
47.00 C  c_awdd     ship's award date
48.00 C  c_deld     ship's delivery date
49.00 C  c_datd     date this info was entered into relate
50.00 C  c_adlif    days added to ship life by this constructio
51.00 C  c_numb     construction number first=1
52.00 C  c_numb     construction number first=1

10-265
ALIAS FORTRAN INCLUDE FILES

FILE FLD03

1.00 C include file fld03
2.00 C
3.00 data fld03 /"'COMFILENAME,COMFILEDESC,LASTUSED,CREATOR,DATCREATED,S
4.00 STARTMENU,NCOMS"/
5.00 C
6.00 C field list for CFLIST (Command File LIST) relation

FILE FLD05

137.00 C include file fld05
138.00 data fld05 /"'SCENARIO,CLASS,HULL,YARD,COMNUM,JOBTYP,JSTYP",
139.00 1 "CUSTOMER,CMETHD,APPROP,AWARD,START,KEEL,LAUNCH",
140.00 2 ",DELIVERY,COMMISSION,DAYSADDED,AUTOMOD,ASNORDE",
141.00 3 "R,SUBRELMAP,DATA_DATE,ENTRY_DATE,ENTRY_BY+"
142.00 C
143.00 C field list for ncjodat.cproj
144.00 C

FILE FLD06

166.00 C include file fld06
167.00 data fld06 /"'SCENARIO,CLASS,YARD,JOBTYP,JSTYP,CUSTOMER",
168.00 1 "COMPLEXGRP,COMNUM,CMETHD,DSGN_AWD,APPROP_AW",
169.00 2 "D,AWD_ST,ST_KL,KL_LN,LN_DL,DAYSADDED,DEFLTA",
170.00 3 "WDAY,TIMUNT+"
171.00 C
172.00 C field list for ncjdat.descj relation
173.00 C

10-266
ALIAS FORTRAN INCLUDE FILES

FILE FL007

C ---include file fl007
C
data fl007/"USERNAME,RUNGROUP,USERLEVEL,READB,ALTOB,"/
C
C 1 "M1,M2,M3,M4,M5,M6,M7,M8,M9,M10,M11,M12,M",
C 1 "13,M14,M15,M16,M17,M18,M19,M20,M21,M22,M",
C 1 "23,M24,M25,M26,M27,M28,M29,M30,M31,M32,M",
C 1 "33,M34,M35,M36,M37,M38,M39,M40,M41,M42,M",
C 1 "43,M44,M45,M46,M47,M48,M49,M50+"/
C
C ---field list for DB read into /usrprv/
C

FILE FL008

C include file fl008
C
data fl008/"SCENARIO,CREATOR,READALLOW,WRALLOW,CREATED,LASTUSED+"/
C
C field list for scenist relation
C
C
FILE FLDECM

1.00 C include file fldecm
2.00 C contains decommissioning data for all ships, first=1
3.00 C
4.00 PARAMETER DECMNM='DEACT.MISC'
5.00 PARAMETER DECMFL=
6.00 'SCENARIO,CLASS,HULL,COMNUM,DEACT,DATADATE'
7.00 PARAMETER DECMKY='SCENARIO,CLASS,HULL,COMNUM'
8.00 COMMON /FLDECM/ dcur, dscen, dclas, dhull, dfalin(17), dkalin(15), ddean, dfdead, dfdatd
9.00 EQUIVALENCE (dcur, dscen, dclas, dhull, dfalin(17), dkalin(15), ddean, dfdead, dfdatd)
10.00 EQUIVALENCE (dcur, dscen, dclas, dhull, dfalin(17), dkalin(15), ddean, dfdead, dfdatd)
11.00 CHARACTER*12 dkscen, dklas*10
12.00 EQUIVALENCE (dkalin(1), dkscen, dklas, dhull, dfalin(17), dkalin(15), ddean, dfdead, dfdatd)
13.00 EQUIVALENCE (dkalin(17), dkalin(15), ddean, dfdead, dfdatd)
14.00 CHARACTER*12 dkscen, dklas*10
15.00 EQUIVALENCE (dkalin(1), dkscen, dklas, dhull, dfalin(17), dkalin(15), ddean, dfdead, dfdatd)
16.00 EQUIVALENCE (dkalin(17), dkalin(15), ddean, dfdead, dfdatd)
17.00 CHARACTER*12 dkscen, dklas*10
18.00 EQUIVALENCE (dkalin(1), dkscen, dklas, dhull, dfalin(17), dkalin(15), ddean, dfdead, dfdatd)
19.00 EQUIVALENCE (dkalin(17), dkalin(15), ddean, dfdead, dfdatd)
20.00 CHARACTER*12 dkscen, dklas*10
21.00 EQUIVALENCE (dkalin(1), dkscen, dklas, dhull, dfalin(17), dkalin(15), ddean, dfdead, dfdatd)
22.00 EQUIVALENCE (dkalin(17), dkalin(15), ddean, dfdead, dfdatd)
23.00 C -- data record for DECOMM (DECOMMISSIONING) data
24.00 C
25.00 C d_cur = relate virtual cursor
26.00 C d scn = a scenario id
27.00 C d cls = a ship class id
28.00 C d hull = hull number for the ship class
29.00 C d_dean = the number of this ship's deactivation
30.00 C d_dead = the date of this deactivation
31.00 C d_datd = date this info was entered into relate
32.00 C
33.00 C
FILE FLHEAD

1.00 C
2.00 C include file flhead FLRP page header info
3.00 CHARACTER TITLES*LLONG(MXTITL),PERHED*LLONG(LINPH)
4.00 CHARACTER LABPRG*LENRLB(MAXPRG), UNTOTL*LLONG
5.00 INTEGER NTITLE
6.00 COMMON/FLHEAD/ NTITLE,TITLES,PERHED,LABPRG,UNTOTL
7.00 C -- nttitle number of lines of title
8.00 C -- title title text lines
9.00 C -- perhed period header text lines
10.00 C -- labprg label for program i
11.00 C -- untotl equal marks above total line
12.00 C

FILE FLIOC

1.00 C
2.00 C include file flioc io unit numbers for FLRP
3.00 INTEGER OCNTRL, IOUTFL, FLRPTF
4.00 COMMON/FLIOC/ OCNTRL, IOUTFL, FLRPTF
5.00 C -- ocntrl force level report input file
6.00 C -- ioutfl write report here
7.00 C -- flrptf write permanent report here
8.00 C
ALIAS FORTRAN INCLUDE FILES

FILE FLJLST

1.00 C
2.00 C include file fljlst
3.00 LOGICAL NONEON
4.00 INTEGER NINCLUD
5.00 CHARACTER*12 INCLST
6.00 COMMON /FLJLST/ NONEON, NINCLUD, INCLST(MAXLST)
7.00 C -- noneon true if nothing from joblist was on
8.00 C -- ninclud number of joblist candidates on
9.00 C -- inclst list of joblist candidates
10.00 C

FILE FLPAGE

1.00 C
2.00 C include file flpage
3.00 INTEGER INBUF,NONPAG,flpds
4.00 C CHARACTER PAGEBUF*LLONG(LENBUF)
4.10 character pagebuf*llomg
4.20 integer ipageb(70)
4.30 equivalence (pagebuf,ipageb)
5.00 COMMON /FLPAGE/ flpds,INBUF,NONPAG
6.00 C -- nonpage number already printed to output page
7.00 C -- inbuf number already in output buffer
8.00 C -- pagebuf page buffer, will go to output page
9.00 C
10.00 C NOTE: the array version of pagebuf was made scalar
11.00 C and local due to HP memory restrictions; storage is
12.00 C now in extended memory segment 8001.
ALIAS FORTRAN INCLUDE FILES

FILE FLPERD

1.00 C
2.00 C include file flperd
3.00 INTEGER NPERIOD
4.00 INTEGER*4 DATPER
5.00 COMMON/FLPERD/ NPERIOD,DATPER(MXPERD)
6.00 C -- datper any ship active on datper(i)is in period(i
7.00 C -- nperiod number of periods being examined
8.00 C

FILE FLPMTR

1.00 C
2.00 C include file flpmtr FLRP parameters
3.00 INTEGER CNLEN=12
4.00 INTEGER MAXCLAS=100
5.00 INTEGER MXTITL=10
6.00 INTEGER LLONG=132
7.00 INTEGER BLTITL=''
8.00 +
9.00 +
10.00 INTEGER LENLLB=12
11.00 INTEGER LEMLB=12
12.00 INTEGER WOTCOL=5
13.00 INTEGER LINPH=3
14.00 INTEGER MAXLST=100
15.00 INTEGER LENBUF=80
16.00 INTEGER LENPAG=60
17.00 INTEGER MXPERD=20
18.00 INTEGER MAXPRG = 4
19.00 INTEGER MXTOTL=15
20.00 C

10-271
FILE FLRJOB

1.00 C
2.00 C include file flrjob
3.00 C contains repair job data for historical, current, projected ships
4.00 PARAMETER HISJNM='REJODAT.HISTJ-'
5.00 PARAMETER CURJNM='REJODAT.CURRJ-'
6.00 PARAMETER PROJNM='REJODAT.PROJJ-'
7.00 PARAMETER RJOBFL=
8.00 + '-SCENARIO,REJOBT,CLASS,HULL,JO81D,START,DELIVERY,DAYSADDED,DAT'+
9.00 +DATE-' 
10.00 PARAMETER RJOBA1='-SCENARIO,REJOBT,CLASS,JO81D, 
11.00 + DATADATE:D,ENTRY_DATE:D-' 
12.00 PARAMETER RJOBK2=
13.00 1 '-SCENARIO,CLASS,HULL,JO81D,DATADATE:D,ENTRY_DATE:D-' 
14.00 COMMON /FLRJOB/ hjcurl,hjcur2,cjcurl,cjcur2,pjcurl,
15.00 + pjcur2
16.00 integer hjcurl,hjcur2,cjcurl,cjcur2,pjcurl, 
17.00 + pjcur2
18.00 character jfscen*12,jfclas*10,jfjobn*6 
19.00 integer jfhull, jfaln(26), jfadda,jfjobid 
20.00 integer*4 jfbegd,jfendd, jfdatd 
21.00 equivalence (jfalin(1),jfscen), (jfalin(7),jfjobn) 
22.00 equivalence (jfalin(10),jfclas), (jfalin(15),jfhull) 
23.00 equivalence (jfalin(16),jfjobid) 
24.00 equivalence (jfalin(17),jfbegd), (jfalin(19),jfendd) 
25.00 equivalence (jfalin(21),jfadda), (jfalin(22),jfdatd) 
26.00 C
27.00 character jklscen*12,jklclas*10,jkljobn*6 
28.00 integer jklalin(22),jkljid 
29.00 integer*4 jkldatd,jkldate 
30.00 equivalence (jklalin(1),jklscen), (jklalin(7),jkljobn) 
31.00 equivalence (jklalin(10),jklclas) 
32.00 equivalence (jklalin(15),jkljid), (jklalin(16),jkldatd) 
33.00 C
34.00 character*1? jkzscen,jkzclas*10 
35.00 integer jkzhull, jkzalin(20), jkzjid 
36.00 integer*4 jkzdatd,jkzdatn 
37.00 equivalence (jkzalin(1),jkzscen), (jkzalin(7),jkzclas) 
38.00 equivalence (jkzalin(12),jkzhull), (jkzalin(13),jkzjid) 
39.00 equivalence (jkzalin(14),jkzdatd), (jkzalin(15),jkzdatn) 
40.00 C
41.00 C -- data record for FLRJOB( Repair JOBS) data
42.00 C

10-272
ALIAS FORTRAN INCLUDE FILES

43.00 C  j_curs  relate virtual cursor(f=fieldvalue,kl=key1
44.00 C  j_scenr a scenario id ,k2=key2)
45.00 C  j_clas  a ship class id
46.00 C  j_hull  hull number for the ship class
47.00 C  j_appd ship's appropriation date
48.00 C  j_awdd ship's award date
49.00 C  j_deld ship's delivery date
50.00 C  j_datd  date this info was entered into relate

10-273
ALIAS FORTRAN INCLUDE FILES

FILE FLTABLES

1.00 C
2.00 C include file fltabls
3.00 INTEGER*4 PRGBEG
4.00 INTEGER NPROGS, FLTABL(MAXCLAS,MXPERD,MXPGR)
5.00 COMMON /FLTABLES/ NPROGS,FLTABL,PRGBEG(MXPGR)
6.00 C -- nprogs number of program types studied
7.00 C -- fltabl force level table, number of that class in
8.00 C that period for that program type
9.00 C -- prgbeg date of program start
10.00 C

FILE FLTOTT

1.00 C
2.00 C include file flttotl
3.00 CHARACTER TOTID*LENRLB
4.00 INTEGER NTOTAR,TOTALS,INTOTL
5.00 COMMON/FLTOTO/NTOTAR,TOTALS(MXTOTL,MXPERD)
6.00 COMMON/FLTOTO/NTOTAR,TOTALS(MXTOTL,MXPERD)
7.00 C ntotal number of rows (arrays) in total being computed
8.00 C totals holds all totals for each period in computation
9.00 C totid holds label for total
10.00 C
ALIAS FORTRAN INCLUDE FILES

FILE FLVALU

* C
* C include file flvalu
* LOGICAL FLKEEP
* INTEGER*4 FLBXER
* INTEGER*4 FLFXER
* CHARACTER*8 FLMRET
* CHARACTER*8 FLPLEN
* CHARACTER*8 FLINFR
* CHARACTER*8 FLPMLS
* CHARACTER*24 FLRJOB
* COMMON/FLVALU/FLKEEP,FLBXER,FLFXER,FLMRET,FLPLEN,FLINFR
* + ,FLPMLS, FLRJOB
* C

FILE FUNCBG

1.00 C
2.00 C include file funcbg
3.00 INTEGER NFUNC,FDEFINE(MXCHOICE,MXFUNC)
4.00 CHARACTER FNMLST*CNLEN(MXFUNC)
5.00 COMMON/FUNCBG/NFUNC,FDEFINE,FNMLST
6.00 C nfunc number of functional families defined
7.00 C fdefine for each function, this holds index into type arrays
8.00 C of type which will perform the function in order of
9.00 C choice, 1=highest priority
10.00 C fnmlst function names for cross referencing into fdefine
20.00 C
ALIAS FORTRAN INCLUDE FILES

FILE GNTUPD

352.00 C**include file gntupd
353.00 parameter ntyp = 2, mxtlen = 28, nfil = 6, mxtlenb = 56
354.00 parameter mxvc=200,mxvy=100,mxvj=20
355.00 common /gntudp/nrels,nomore(nfil),itup(mxtlen,nfil),gnfld(ntyp),
356.00 1 klenb,gncurs(nfil),gnfile(nfil),gntype(nfil),
357.00 2 vlcias(mxvc),vlyard(mxvy),vljobt(mxvj),
358.00 3 nowcls(nfil),nowyrd(nfil),nowjob(nfil),
359.00 4 nvclas,nvyard,nvjobs
360.00 integer nrels,itup,klenb,gncurs
361.00 character gnfld*62
362.00 character gnfile*18,ctup*mxtlenb(nfil)
363.00 character vlcias*10,vlyard*8,vljobt*6
364.00 integer nowcls,nowyrd,nowjob
365.00 integer nvclas,nvyard,nvjobs
366.00 logical nomore
367.00 equivalence (itup,ctup)
368.00 C
369.00 C ---static storage for gntup routine
370.00 C nrels number of relations to be accessed (max nfil)
371.00 C nomore true if no more data in relation i
372.00 C itup storage for current tuples
373.00 C klenb length of key section of tuple in bytes (for chash)
374.00 C gncurs cursor indexes for each relation
375.00 C gnfile name of each relation
376.00 C gnfld field list for two relation types
377.00 C gntype type of each relation
378.00 C vlcias,yard,jobt lists of valid field values
379.00 C nvclas,yard,jobt number of members on each valid list
380.00 C**ENDBLK
381.00 C
ALIAS FORTRAN INCLUDE FILES

FILE GROUPBG

1.00 C
2.00 C include file groupbg
3.00 INTEGER NGROUP,GRPFILP(MXPERD,MXGROUP),GMKPTR(MXGROUP)
4.00 INTEGER DGRPLEVL(MXPERD,MXGROUP),AGRPLEVL(MXPERD,MXGROUP)
5.00 CHARACTER GRPLAB*LENRLB(MXGROUP),GRPLST*CNLEN(MXGROUP)
6.00 COMMON/GROUPBG/NGROUP,GRPFILP,GMKPTR,DGRPLEVL,AGRPLEVL,
   + GRPLAB,GRPLST
7.00  ngroup number of battle groups
8.00  grpfilp group's fill priority order(1high)
9.00  dgrplevl desired number of this group
10.00 agrplevl actual number of this group found
11.00 grplab label for output group line
12.00 grplst list of group names

ALIAS FORTRAN INCLUDE FILES

FILE INCPAR

1.00 C
2.00 C include file incpar

MNUR parameters

3.00 PARAMETER MXMENU = 100
4.00 PARAMETER MAXOPT = 15
5.00 PARAMETER LLINE = 72
6.00 PARAMETER LTEXT = 70
7.00 PARAMETER SNAME = 6
8.00 PARAMETER LNAME = 6
9.00 PARAMETER DIMNAME = 8
10.00 PARAMETER LHTXT = 800

11.00 C mxmenu maximum number of choice or parameter menus
12.00 C maxopt maximum options per choice or param. menu
13.00 C lline length of an input line
14.00 C ltext length of input descriptive text
15.00 C lname max. length of a parameter name
16.00 C sname max. length of a subroutine name
17.00 C dimname dimension of a name=name; divisible by 12
18.00 C lhtxt max. length of a menu's help text
19.00 C
ALIAS FORTRAN INCLUDE FILES

FILE INDEXS

1.00 C
2.00 C include file indexs    DBIF index tracking info
3.00    PARAMETER michn= 6,midx= 4,mcidx=132
4.00    COMMON /indexs/ idxchn(michn),idx(midx)
5.00    INTEGER idxchn
6.00    CHARACTER idx*mcidx
7.00 C  idxchn=index chain     idx=index pool
8.00 C

FILE INPUTL

1.00 C
2.00 C include file inputl    CORE (READLN) LAST LINE BUFFER
3.00    INTEGER LENLN
4.00    CHARACTER LASTLN*LLINE
5.00    COMMON / INPUTL/ LENLN,LASTLN
6.00 C  lenln    non-blanked length of lastln
7.00 C  lastln   last line read from command input file
8.00 C
ALIAS FORTRAN INCLUDE FILES

FILE IO

1.00 C
2.00 C include file io
3.00 COMMON /io/ in, iout, itty
4.00 INTEGER in, iout, itty
5.00 C
6.00 C in = standard INput unit  iout = standard OUTput unit
7.00 C itty = standard user input unit
8.00 C

FILE IOC

1.00 C
2.00 C include file io:
2.10 parameter salp = 8, daisy = 7, termnl = 6
3.00 parameter syslhp=49, sysmp=48, dpdec=51, drunp=52
4.00 parameter dcmem=53, dmroot=54, dpref=55, dpequi=56
5.00 parameter dpdec=57, dcmem=58, dhtxt=59, ddtxt=60
6.00 parameter deunit=22, idunit=23
7.00 C units 24-27 used by scenario system
8.00 C parameter modlhp=28
9.00 C
10.00 COMMON/IOC/ IN, IOUT, IOUTLP, ITTYIN, ITTYYOU, IOCEXTRA(IO)
10.10 C in = file number for PMS 92 line printer
10.20 C daisy = unit number for SEA 90 daisy wheel printer
10.30 C termnl = unit number for $STDLIST
11.00 C ioutlp = file assigned to desired output printer
12.00 C in = file number from which input is expected
13.00 C iout = file to which output is written
14.00 C drunp = file to which subroutine runprocs is written
15.00 C ittyin = file assigned to input from screen file
16.00 C ittyou = file assigned to output to screen file
17.00 C dpdec = file to which common/pdec/ is written
18.00 C dlistm = file to which list memory is written
19.00 C dmroot = file to which common/mroot/ is written
20.00 C dpref = file to which parameter menu field lists
21.00 C and pointers are written
22.00 C dpequi = file to which equivalence statement between
23.00 C pvalue and parameter names is written

10-280
ALIAS FORTRAN INCLUDE FILES

24.00 C    dpdec file to which parameter type statements are written
25.00 C    deunit data maintenance subsystem def file unit
26.00 C    idunit unit on which unique id code generator file opens
27.00 C    locextra use this space when adding to common block
29.00 C    rather than recompile the world.
30.00 C    modhp unit module help/menu text file is opened on
31.00 C
ALIAS FORTRAN INCLUDE FILES

FILE LINKM

1.00 C include file linkm --- list memory global storage
2.00 C PARAMETER lenlk = 4000, llinkm = 8006
3.00 COMMON /linkm/iavail, iavind, nbcell, linkm(lenlk)
4.00 INTEGER*4 iavail, iavind, nbcell, linkm
5.00 INTEGER*4 iavail, iavind, nbcell, linkm
6.00 C iavail- free cell chain ptr iavind- 'available' indicator value
7.00 C nbcell- # words in list mem linkm - Linked list memory array
8.00 C

FILE LISTYP

42.00 C include file listyp
43.00 C COMMON /listyp/ nityps, ltyps(mxltyp)
44.00 INTEGER nityps
45.00 CHARACTER*8 ltyps
46.00 C
47.00 C
48.00 C Common listyp contains a list of all valid list menu types.
49.00 C nityps Number of List menu TYPES generated by mnugen.
50.00 C ltyps names of List menu TYPES generated by mnugen.
51.00 C
FILE LMENU

1.00 C
2.00 C include file lmenu

3.00 CHARACTER*8 LMID, LMRELT
4.00 INTEGER*4 ITXTP, LHPRTR, LHLEN
5.00 COMMON /LMENU/ ITXTP, LMRELT, LMID, LHPRTR, LHLEN
6.00 C ITXTP pointer to list menu's title text
7.00 C LMID unique list menu id for data base
8.00 C LMRELT relation type of list menu for current parameter
8.10 C LHPRTR pointer to list help in dhtxt
8.20 C LHLEN number of lines of help text

FILE LPRNTS

1.00 C
2.00 C include file lprnts

3.00 COMMON /lprnts/ IOUTP, LPRNTS(160)
4.00 INTEGER IOUTP
5.00 LOGICAL LPRNTS, LPRNT
**ALIAS FORTRAN INCLUDE FILES**

**FILE LTPAC**

98.00 C
99.00 C include file ltypac
100.00 common /ltypac/ itypcr, tflist
101.00 integer itypcr
102.00 character*30 tflist
103.00 C
104.00 C Common List TyPe relation ACcess allows a single open and
105.00 C field-list construction to occur when a given list menu
106.00 C is to be updated. Saving the field-list, which contains
107.00 C the field name of the proper status column, saves much
108.00 C redundant DB accessing.
109.00 C itypcr cursor number for the current list type relation
110.00 C tflist field list for the current list type menu

**FILE LVAL**

1.00 C
2.00 C include file lval: MNUR---list menu candidates/statuses
3.00 PARAMETER MAXLST = 200
4.00 INTEGER NLITMS, LENHLP
4.10 INTEGER*4 LHLPPTR
5.00 CHARACTER*12 CNAME
6.00 LOGICAL STATUS
7.00 COMMON/LVAL/NLITMS, CNAME(MAXLST), STATUS(MAXLST)
7.10 + ,LHLPPTR, LENHLP
8.00 C
ALIAS FORTRAN INCLUDE FILES

FILE MKUPBG

1.00 C
2.00 C include file mkupbg
3.00 INTEGER LASTREC, BGMAKUP (MXMKUP, 3)
4.00 COMMON/MKUPBG/LASTREC, BGMAKUP
5.00 C lastrec last row of bgamkup filled
6.00 C bgmakup holds group makeup definitions, 1 col=ptr
to next row in bgmakup holding next part of
definition, 0 if last. 2 col index into function arrays, 3 col % of this function needed to makeup
7.00 C
8.00 C
9.00 C
10.00 C
20.00 C

FILE MNUPRM

25.00 C
27.00 C include file mnuprm
28.00 parameter mxifld=12, mxltyp=50, mxrels=200, nltrec=09
29.00 C parameter maxmlt=nltrec/mxscen; # columns in list type rel/mxscen
30.00 parameter mxlimit=5
31.00 C
32.00 C
33.00 C
34.00 C
35.00 C
36.00 C
37.00 C
38.00 C
39.00 C
40.00 C

Contains parameters used exclusively by the menu system.
mxifld Maximum List menu relation candidate Field size.
Be sure to alter hardwired creation call in gistrl if
this parameter's value is changed.
mxltyp Maximum number of List menu Type relations.
mxrels Maximum number of RELations creatable by mnugen.
nlrec Number of List Type Relation Columns.
mxlimit Maximum List Menus per List Type

FILE MROOT

1.00 C 10-285
ALIAS FORTRAN INCLUDE FILES

2.00 C include file mroot:
3.00 INTEGER*4 MRTPTR
4.00 COMMON /MROOT/MRTPTR
5.00 C
ALIAS FORTRAN INCLUDE FILES

FILE PARAMS

2.00 C
3.00 C include file params
4.00   parameter rnmxch=6, mxscen=10
5.00 C
6.00 C   Contains system-level parameters.
7.00 C   rnmxch  Relation Name Maximum Characters. May be no more
8.00 C   than 7 for HP RELATE.
9.00 C   mxscen  Maximum number of SCEnarios.
10.00 C

FILE PARLST

1.00 C
2.00 C include file parlst:  
3.00 CHARACTE PMNLST*LNAME
4.00 COMMON /PARLST/ PMNLST(MXMENU)
5.00 C
FILE PCREAT

1.00 C include file pcreat: MNUG---data for parameter storage setup
2.00 C
3.00 INTEGER ISTRC, NPADS, IDFLTV
4.00 PARAMETER MXLSTR = 300
5.00 CHARACTER*1 STRCTR, CDEFLT*4(100)
6.00 REAL DEFLTV
7.00 COMMON /PCREAT/ ISTRC, DEFLTV(100), STRCTR(MXLSTR)
8.00 1, NPADS, IDFLTV
9.00 EQUIVALENCE (CDEFLT(1), DEFLTV(1))
10.00 C instrc # of characters in strctr
11.00 C npads # of padding variables used
12.00 C idfltv # of words used in defltv
13.00 C defltv default values for relation command in listqueue
14.00 C cdeflt character rep. of defltv
15.00 C listqu actual queue of list commands,
16.00 C   first out = listqu(inque),
17.00 C

FILE PDESC

1.00 C include file pdesc:
2.00 C
3.00 INTEGER*4 MPTXTP, PHPTR
5.00 INTEGER MPINDX, MTYPE, MPLEN, PHLEN
6.00 COMMON/PDESC/ MPTXTP(MAXOPT), MPLEN(MAXOPT), PHLEN(MAXOPT),
7.00 1 MPINDX(MAXOPT), MTYPE(MAXOPT), PHPTR(MAXOPT)
8.00 C
ALIAS FORTRAN INCLUDE FILES

FILE P6SYS

1.00 C include file pgsys
2.00 parameter pgunit=29
3.00 common /pgsys/pgatln,pgnext,pgout,pglenn,pgplen,pgwfc,
4.00 integer pgmod,pgfmod,pqut,pgqchr,pqlast,pqtop
5.00 integer pgatln,pqomod,pgfmod,pqout,pglenn,pgplen
6.00 integer pqlast,pqtop,pqnext
7.00 logical pqout,pgwfc
8.00 character*1 pgqchr
9.00 C control data for page printer
10.00 C pglenn location of last line added to page buffer
11.00 C pgnext location of next free line in buffer
12.00 C pgout unit to send output to
13.00 C line length of output page
14.00 C number of lines per output page
15.00 C true if first call to pgwrite has been made
16.00 C operating mode:
17.00 C 1=prompt user for page feed, print until eopage; prompt
18.00 C 2=don't prompt user, print continuously, header each page
19.00 C 3=don't prompt, print continuously with header top only
20.00 C 4=don't prompt user, let user print on eopage
21.00 C line feed mode: 1 for sclear, 2 for lhi
22.00 C quit? prompting in effect
23.00 C recognition character indicating quit
24.00 C location in buffer of last line guaranteed to fit
25.00 C this page
26.00 C location in buffer of first line awaiting printing
FILE PMENU

1.00 C
2.00 C include file pmenu: MNUR---current parameter menu info
2.10 CHARACTER RNTXT*DIMNAME, MPTTXT*LLINE
3.00 INTEGER*4 MPHPTR
4.00 INTEGER IDPMEN, MAXPGI, MPHLEN
5.00 COMMON/PMENU/ IDPMEN, MAXPGI, MPHLEN, MPHPTR
5.00 1 , RNTXT, MPTTXT
7.00 C

FILE PPINDEX

1.00 C
2.00 C include file ppindex: MNUG---parameter set up data
3.00 INTEGER NXTPGI, PVINDX
4.00 COMMON/PPINDEX/ NXTPGI, PVINDX
5.00 C nxtpgi the next parameter to be defined will use
6.00 C index=nxtpgi in the arrays of common/pdesc
7.00 C pvindx the value of the next parameter to be
8.00 C defined will begin at pvalue(pvindx)
9.00 C
ALIAS FORTRAN INCLUDE FILES

FILE PRMCRS

1.00 C INCLUDE FILE PRMCRS
2.00 common /prmcrs/cflcrs,lcrcrs,snucrs
3.00 integer cflcrs,lcrcrs,snucrs
4.00 C
5.00 C cflcrs cursor for command file CFLIST relation
6.00 C lcrcrs cursor for lccref--list menu crossref
7.00 C snucrs cursor for snusers--scenarios now in use

FILE PVALUE

1.00 C
2.00 C include file pvalue: MNUR---parameter values storage
3.00 PARAMETER PTLEN=500
4.00 REAL PVALUE(PTLEN)
5.00 INTEGER IVALUE(2,PTLEN)
6.00 INTEGER*4 DVALUE(PTLEN)
7.00 CHARACTER CVALUE*4(PTLEN)
8.00 LOGICAL LVALUE(2,PTLEN)
9.00 COMMON /PVAL/ PVALUE
10.00 EQUIVALENCE (PVALUE,IVALUE,DVALUE,CVALUE,LVALUE)
11.00 C
ALIAS FORTRAN INCLUDE FILES

FILE PVDECL

CORE---mnug-written decls for /pvalue/ equivs

* CHARACTER* 8 TTYTYP
* CHARACTER* 12 LPUNIT
* LOGICAL PAD1, PSMNUH
* CHARACTER* 8 PDURAT
* INTEGER*4 DPFRST
* INTEGER*4 DPLAST
* CHARACTER* 24 ASNYDS
* CHARACTER* 24 ASNCLS
* CHARACTER* 24 ASNJTP
* CHARACTER* 8 DISBAS
* CHARACTER* 8 ADJBAS
* CHARACTER* 12 ADJMOD
* CHARACTER* 12 JEPOCH
* CHARACTER* 12 SRTC
* CHARACTER* 12 SRTYRD
* CHARACTER* 4 REFRESH
* LOGICAL PAD2, RPKEEP
* INTEGER*4 RPBXER
* INTEGER*4 RPFXER
* CHARACTER* 8 RPMRET
* CHARACTER* 8 RPPLEN
* CHARACTER* 8 RPINF
* CHARACTER* 8 RPPMLS
* CHARACTER* 24 RPRJOB

10-292
ALIAS FORTRAN INCLUDE FILES

FILE PVEQIV

CORE---mnuq-written equiv to /pvalue/

EQUIVALENCE (PVALUE(1),TTYTP)
EQUIVALENCE (PVALUE(3),LPUNIT)
EQUIVALENCE (IVALUE(1, 6),PA01)
EQUIVALENCE (IVALUE(2, 6),PSMNUH)
EQUIVALENCE (PVALUE(7),PDURAT)
EQUIVALENCE (PVALUE(9),DFRST)
EQUIVALENCE (PVALUE(10),DPLAST)
EQUIVALENCE (PVALUE(11),ASNYDS)
EQUIVALENCE (PVALUE(17),ASNCPLS)
EQUIVALENCE (PVALUE(23),ASNTP)
EQUIVALENCE (PVALUE(29),DISBAS)
EQUIVALENCE (PVALUE(31),ADJBAS)
EQUIVALENCE (PVALUE(33),ADJMOD)
EQUIVALENCE (PVALUE(36),JEPOCH)
EQUIVALENCE (PVALUE(39),SRTCLS)
EQUIVALENCE (PVALUE(42),SRTYRD)
EQUIVALENCE (PVALUE(45),REFRSH)
EQUIVALENCE (IVALUE1, 46),PAD2)
EQUIVALENCE (IVALUE(2, 46),RPKEEP)
EQUIVALENCE (PVALUE(47),RPXER)
EQUIVALENCE (PVALUE(48),RPFXER)
EQUIVALENCE (PVALUE(49),RPMRET)
EQUIVALENCE (PVALUE(51),APPLEN)
EQUIVALENCE (PVALUE(53),RPINFR)
EQUIVALENCE (PVALUE(55),RPPMLS)
EQUIVALENCE (PVALUE(57),RPRJOB)

10-293
ALIAS FORTRAN INCLUDE FILES

FILE PXREF

1.00 C
2.00 C include file pxref
3.00 C
4.00 C include file pxref
5.00 C
6.00 C
7.00 C
8.00 C
dflen  "delemited field list's max. length"
9.00 C
10.00 C
cfledl "delemited field list for idpmen = 1"
11.00 C
12.00 C
FILE QUEUE

1.00 C
2.00 C include file queue
3.00 INTEGER INQUE
4.00 CHARACTER*12 LISTQU
5.00 COMMON /QUEUE/ LISTQU(36), INQUE
6.00 C inque number of list command in listqueue
7.00 C listqu actual queue of list commands,
8.00 C first out = listqu(inque),
9.00 C last out = listqu(1)
10.00 C
ALIAS FORTRAN INCLUDE FILES

FILE RCRD01

1.00 C
2.00 C include file rcrd01
3.00 common /rcrd01/menu01,rltn01,colm01
4.00 character colm01*4
5.00 character*8 menu01,rltn01
6.00 character*34 fld01
7.00 integer alin01(10)
8.00 equivalence (alin01,menu01)
9.00 C
10.00 C Common /rcrd01/ provides a record for use in passing
data to the lcrref relation.
11.00 C scen01 a scenario id
12.00 C colm01 column number given menu is stored in in relation
13.00 C menu01 a list menu identifier
14.00 C rltn01 a list type relation name
15.00 C fld01 field list for the lcrref relation
16.00 C
17.00 C

FILE RCRD03

730.00 C
731.00 C include file rcrd03
732.00 C
733.00 common /rcrd03/cflnam,cflusc,cflusd,cflctr,cflctd,cflismu
733.10 1,cflrec
734.00 1,cflrec
735.00 character*8 cflnam,cflctr,cflismu
735.10 character*42 cflusc
736.00 character*70 fld03
736.10 real cflctd,cflusc
736.20 integer alin03(38)
736.20 equivalence (alin03,cflnam)
737.00 C
738.00 C data record for CFLIST (Command File LIST) relation
739.00 C
cflnam command file name
740.00 C
cfldsc command file description
741.00 C
cflusc command file last-used date
742.00 C
## ALIAS FORTRAN INCLUDE FILES

<table>
<thead>
<tr>
<th>Line</th>
<th>Alias</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>743.00 C</td>
<td>cflctr</td>
<td>command file creator</td>
<td></td>
</tr>
<tr>
<td>744.00 C</td>
<td>cflctd</td>
<td>command file creation date</td>
<td></td>
</tr>
<tr>
<td>745.00 C</td>
<td>cflsmu</td>
<td>command file start-execution menu</td>
<td></td>
</tr>
<tr>
<td>746.00 C</td>
<td>cflrec</td>
<td>number of records in this command file</td>
<td></td>
</tr>
</tbody>
</table>

10-297
FILE RCRD06

145.00 C: include file rcrd06
146.00 common /rcrd06/scen06,clas06,yard06,jtyp06,
147.00   styp06,cust06,grp06,com06,mthd06,
148.00 2 da06,aa06,as06,sk06,k106,l06,dd06,daw06,t06
group06,jobtype06
class06, Yard06, J06, S0, A, T0, K106, L06, D06, A, T06
149.00 character clas06*10,yard06*8,jtyp06*6, styp06*6, cust06*8, grp06*10
150.00 character tun06*6,scen06*12,mthd06*6
151.00 integer*4 daw06
152.00 integer da06,aa06,as06,sk06,k106,l06,dd06,com06
153.00 integer alin06(46)
154.00 equivalence (scen06,alin06)
155.00 character*43 fl06(4)
156.00 C holds a tuple returned from the job description relation
157.00 C variables are, in order, scenario, class, yard, jobtype,
158.00 C job series type, customer, complexity group, commissioning number,
159.00 C construction method, design-award time,
160.00 C approp-award, award-start, start-keel, keel-launch, launch-delivery,
161.00 C days added to life of ship, default award day in year,
162.00 C and time units planning factors are in
163.00 C
164.00 C
165.00 C

10-298
FILE RCRD08

include file rcrd08

parameter len08 = 22

common /rcrd08/senam,creatr,racces,wacces,creatd,lstusd

character senam*12,creatr*8,racces*8,wacces*8

integer creatd,lstusd

character fld08*70

integer alin08(len08)

equivalence (alin08,senam)

record for communication with scenlst relation

senan  name of scenario
creatr creator of scenario
racces read permission ("PUBLIC" or creator name)
wacces write permission ("PUBLIC" or creator)
creatd date created
lstusd date last used

10-299
ALIAS FORTRAN INCLUDE FILES

FILE READC

1.00 C
2.00 C include file readc stores last line read by rdln for lwarn
3.00 INTEGER ILINE
4.00 COMMON/READC/ ILINE
5.00 C number of line last read from input file
6.00 C ( for preproc )
7.00 C

FILE RELNAM

12.00 C
13.00 C include file relnam
14.00 common /relnam/ nrels, rlnams(mxrels)
15.00 character rlnams*8
16.00 integer nrels
17.00 C
18.00 C Common RELation NAMes holds the names of all relations to be created by the menu generation processor.
19.00 C nrels Number of RELationS. Count of relations created by mnugen.
20.00 C rlnams Relation NAMEs. List of names of relation created by mnugen.
21.00 C
22.00 C
23.00 C
24.00 C
ALIAS FORTRAN INCLUDE FILES

FILE RPSUBS

1.00 C
2.00 C include file rpsubs: MNUG---rnproc creation data
3.00 PARAMETER MAXSUBS= 100
4.00 INTEGER NPROCS
5.00 CHARACTER SUBLST$SNAME
6.00 COMMON/RPSUBS/SUBLST(MAXSUBS), NPROCS
7.00 C nprocs number of special purpose processes that
8.00 C have been referenced.
9.00 C sublist list of all special purpose processes' subroutine names.
10.00 C
11.00 C
ALIAS FORTRAN INCLUDE FILES

FILE SCENAR

* . C include file scenar
  SCENARIO SYSTEM DATA
* . C  parameter sncurs=20
  * . C  common /scenar/actsen,curse(sncurs),dlsen(sncurs),
  * . C  1                         wrtprv(sncurs),snwovr
  * . C  character*12 actsen,curse(dlmse=14
  * . C  logical wrtprv,snwovr
  * . C  integer alinsen
  * . C  equivalence (alinsen,actsen)
  * . C
  * . C  current scenario data for application routines
  * . C  sncurs  max number of relate cursors
  * . C  actsen  name of current scenario
  * . C  curse  scenario key value for relation i
  * . C  dlmse  delimited version of curse
  * . C  wrtprv  true if user may write on the current cursor
  * . C  snwovr  scenario write privilege override; allows
  * . C  write on cursors regardless of scenario
  * . C  status if true; used by scenario creator
  * . C

10-302
ALIAS FORTRAN INCLUDE FILES

FILE SCRCHR

1.00 C include file SCRCHR -- screen characters
2.00 C
3.00 PARAMETER sprev = '-', sfolow = '+', spop = '
',
4.00 1 sleft = '<', sright = '>', srfrsh = '&',
5.00 2 sadd = 'A', sdel = 'D', smod = 'M',
6.00 3 shelp = '?', sinart = 'I', sprint = 'P',
7.00 PARAMETER scopy = 'C', ssnap = 'S', snam = 'N',
8.00 5 srloc = 'R', squit = 'Q', sendpg = 'E',
8.10 6 stopmu = '/', sune = '<', sstbit = '>',
8.20 7 sjmpto = '=', sedit = 'T', scifld = 'K',
8.30 8 sjrewnd = 'B', supdat = 'U', sverfy = 'V',
8.40 9 smode = 'L', sdaw = '*'
9.00 PARAMETER scrchr = '-+<>&AOM?IPCSNRQE/{}-TKBUVL.', lenscr=29
9.10 PARAMETER nothat = '^/QE', lenoth = 4
10.00 PARAMETER jprev = 1, jfolow = 2, jpop = 3,
11.00 1 jleft = 4, jright = 5, jrfresh = 6,
12.00 2 jadd = 7, jdel = 8, jmod = 9,
13.00 3 jhelp = 10, jsinart = 11, jprint = 12,
14.00 4 jcopy = 13, jswap = 14, jname = 15,
15.00 5 jreloc = 16, jquit = 17, jendpg = 18,
15.10 6 jstopmu = 19, juse = 20, jstbit = 21,
15.20 7 jjmppto = 22, jedit = 23, jclfld = 24,
15.30 8 jrewnd = 25, jupdat = 26, jverfy = 27,
15.40 9 jmode = 28, jdraw = 29
16.00 C
FILE SCREEN

1.00 C
2.00 C include file screen
3.00 PARAMETER WSCREEN = 80
4.00 PARAMETER LSCREEN = 24
5.00 C wscreen wide of terminal screen is characters
6.00 C lscreen length of terminal screen in lines

FILE SENPRM

C include file senprm
parameter snmxrl=200, snmxgp=20, snmxgr=50
parameter snrsun=24, snunit=5
parameter snread=26, snwrit=27
C scenario system parameters
C snmxrl maximum relation in ALIAS system
C snmxgp maximum number of related groups of relations
C snmxgr maximum number of relations in a group
C snrsun unit number for relsnl file
C snunit unit used by snok
C snread "
C snwrit "
FILE SHLIFE

1.00 C include file shlife
2.00 C contains standard lifetimes for all ship classes
3.00 C FLRP/FLBG ship life data
4.00 PARAMETER SLIFNM='SHLIFE.MISCJ'
5.00 PARAMETER SLIFFL='SCENARIO,CLASS,LIFE,TIMUNT,DATDATE'
6.00 PARAMETER SLIFKY='SCENARIO,CLASS,DATDATE:O,ENTRY_DATE:O'
7.00 COMMON /SHLIFE/ slcurs
8.00 character*12 sfscen
8.10 character*10 sfclas,sfunt*6
9.00 integer slcurs,sflife,sfallin(17)
10.00 integer*4 sfdatd
11.00 equivalence (sfallin(1),sfscen), (sfallin(7),sfclas)
12.00 equivalence (sfallin(12),sflife), (sfallin(13),sfunt)
12.10 equivalence (sfallin(16),sfdatd)
13.00 C
14.00 character*12 skscen
14.10 character*10 skclas
15.00 integer skalin(15)
16.00 integer*4 skdatd,skdate
17.00 equivalence (skalin(1),skscen), (skalin(7),skclas)
18.00 equivalence (skalin(12),skdatd),(skalin(14),skdate)
19.00 C
20.00 C -- data record for SHLIFE (SHIP class LIFEtime in DAYS)
21.00 C
22.00 C slcurs relate virtual cursor
23.00 C s_scen a scenario id
24.00 C s_clas a ship class id
25.00 C s_life ship class's standard life in years
26.00 C s_datd date this info was entered into relate
27.00 C

10-305
FILE SNRREF

C include file snrref
parameter lensnrref = 18/2 * snmxrl + 12/2 * snmxrl + 1
parameter lensnrref = 3001
common /snrref/sndsid,nsets,relset(snmxgp),nsreln
C
integer sndsid,nsreln,nsets
character*18 snrlnm,snrlsn*12,relset*8

scenario system choice/creation data
nsets number of system relation families
relset id of system relation families
nsreln number of relations known to scenario system
snrlnm names of relations known to scenario system
snrlsn scenario field key value for relation 1 for current scenario

FILE STACK

1.00 C include file stack stack data type storage
2.00 C PARAMETER lstack=128
3.00 COMMON /stack/ stkflg, stkidx, stack(lstack)
4.00 INTEGER*2 stkidx
5.00 INTEGER*4 stack
6.00 LOGICAL stkflg
7.00 C stkidx - index to top of the... stack - stack contents
8.00 C stkflg - can be used to mean pop to top menu, when set
9.00 C
10.00 C
include file strings DBIF string chain data storage
PARAMETER mschn=5, mstr=3, mswstr=500, mcstr=1000
COMMON /strings/ strchn(mschn), intstr(mswstr, mstr)
INTEGER strchn, intstr
CHARACTER str(mcstr, mstr)
EQUIVALENCE (intstr, str)
strchn=string chain str =string pool
allows convenient buffering of command strings and
delimited text strings (OTS)
ALIAS FORTRAN INCLUDE FILES

FILE TDDATE

1.00 C Include File TDDATE
2.00 C Defines the DATA TYPE "ODATE", meaning DatabaseDATE; it refers
to the integer*4 format in which RELATE returns date specs.
3.00 C
4.00 C
5.00 C
6.00 C REPRESENTATION: INTEGER*4- Bits 2-13 (L1-12) are YEAR (ie. 1983)
7.00 C - Bits 17-20 (R0- 3) are MONTH (ie. 1)
8.00 C - Bits 21-25 (R4- 8) are DAY (ie. 30)
9.00 C
10.00 C SUBTYPES:
11.00 C Raw RELATE ODATE: a date obtained directly from RELATE tuple.
12.00 C Clarified ODATER: one whose unused bits are guaranteed zero.
13.00 C
14.00 C OPERATIONS:
15.00 C CDTODD(date_string*10 "MM/DD/CCCC") Returns(Clarified ODATER)
16.00 C CKDATE(date_string*10, len_of_string)Returns(Boolean: T if valid)
17.00 C CKDATI(in*2: mm,dd,yy) Returns(Boolean: T if valid)
18.00 C DATEPI(in/out*2: mm,dd,yy) -- increments date by one day
19.00 C ODATER (dummy*2) Returns(Today's date as Clarified ODATER)
20.00 C DCLRFY(Raw ODATER) Returns(Corresponding Clarified ODATER)
21.00 C DTTOD(Any ODATER) Returns(date_string*10 "MM/DD/CCCC")
22.00 C ODTOID(in: Any ODATER, out*2: month, out*2 day, out*2 year)
23.00 C IDAYS (in*2: mm,dd,yy1, in*2: mm2,dd2,yy2)Returns(I*2: 2-1)
24.00 C JDAY (in*2: mm,dd,yy1, in*2: mm2,dd2,yy2)Returns(I*2: 2-1)
25.00 C IDTODD(out: Clarified ODATER, Rest are in*2: month,day,year)
26.00 C LMONTH(in*2: mm,yy) Returns(I*2 number of days in that month)
27.00 C NUMDAY(in*2: mm,dd,yy) Returns(I*2: 1==Sunday...7==Saturday)
28.00 C NWDATER(in: Any ODATER, in*2: ndays) Returns(ODATER+ndays)
29.00 C NWODATU(in: Any ODATER, in*2: npers, inC*: per_type)Returns(ODATER)
30.00 C NWIDAT(in*2: mm,dd,yy, in*2: ndays, out*2: [mm,dd,yy]+ndays)
31.00 C
32.00 C
33.00 C
34.00 C
35.00 C
36.00 C
37.00 C
38.00 C
39.00 C
40.00 C

DECLARATIONS:

EXTERNAL ckdate,ddtod,cdtodd,odater ,dclrfy,dttod,idtodd

10-308
ALIAS FORTRAN INCLUDE FILES

41.00 EXTERNAL idays, jdays, datepl, nwdate, nwdat, nwdatu, lmonth
42.00 CHARACTER ddiocd*10
43.00 INTEGER*4 cdtodd, ddate, dclrfy, jdays, nwdate, nwdatu, mrkday
44.00 INTEGER*4 ddzqzl, ddzqz2, ddzqzc
45.00 INTEGER idays, lmonth, numday
46.00 LOGICAL ckdate, dclosr, dearlr, dlater, dequal
47.00 C
48.00 C STATEMENT FUNCTIONS:
49.00 dclosr(ddzqzl, ddzqz2, ddzqzc) =
50.00 1 jabs(ddzqzl-ddzqzc), LT, jabs(ddzqz2-ddzqzc)
51.00 dearlr(ddzqzl, ddzqz2) = ddzqzl.LT. ddzqz2
52.00 dequal(ddzqzl, ddzqz2) = ddzqzl.EQ. ddzqz2
53.00 dlater(ddzqzl, ddzqz2) = ddzqzl.GT. ddzqz2
54.00 numday(kmn, kdy, kyr) = iint(jmod(mrkday(kmn, kdy, kyr), 7J)) + 1
55.00 C
ALIAS FORTRAN INCLUDE FILES

FILE TODAYC

1.00 C
2.00 C include file todayc FLRP/BGRP data
3.00 INTEGER*4 TODAY, LASTDAY
4.00 COMMON /TODAYC/ TODAY, LASTDAY
5.00 C -- today's clarified date
6.00 C -- maximum clarified date
7.00 C

FILE TRNS03

730.00 C
731.00 C include file trns03
732.00 C
733.00 common /trns03/cftnam,cftdsc,cftusd,cftctr,cftctd,cftsmu
734.00 character*8 cftnam,cftctr,cftsmu
735.00 character*42 cftdsc
736.00 real cftctd,cftusd
737.00 C
data record used to store directory info about the
738.10 C command file currently being built for inclusion in
738.20 C the CFLIST relation on successful build termination.
738.30 C Twins most of rcd03. NOT ALIGNED. DO NOT USE AS RELATE RECORD.
739.00 C
740.00 C cftnam command file name
741.00 C cftdsc command file description
742.00 C cftusd command file last-used date
743.00 C cftctr command file creator
744.00 C cftctd command file creation date
745.00 C cftsmu command file start-execution menu

10-310
ALIAS FORTRAN INCLUDE FILES

FILE TTY

94.00 C
95.00 C include file /tty/ alternative screen clear method storage
96.00 COMMON /tty/ typtty, formfd
97.00 CHARACTER*6 typtty
98.00 INTEGER formfd
99.00 C

FILE TXTCNT

1.00 C
2.00 C include file txtcnt : MNUG--text size tracking
3.00 INTEGER*4 NHLPLN, NDSCLN
4.00 COMMON /TXTCNT/ NHLPLN, NDSCLN
5.00 C
FILE TYPEBG

1.00 C
2.00 C include file typebg

FILEB6---type definition data

3.00 INTEGER NTYPE, TYPTOT(MXPERO, MXTYPE)
4.00 CHARACTER TYPLAB*LENRLB(MXTYPE), TYPNAM*CNLEN(MXTYPE)
5.00 COMMON /TYPEBG/NTYPE, TYPTOT, TYPLAB, TYPNAM

6.00 C
7.00 C ntype number of types defined
8.00 C typtot force level totals for each period, each type
9.00 C typlab label to be displayed under BALANCE section for type
20.00 C typnam name which type is cross referenced by
FILE UNTREF

1.00 c include file untref
2.00 c FIOPN/FILCLS utility storage
3.00 common /untref/unums(99),uinuse(99)
4.00 integer unums
5.00 logical uinuse
6.00 c unums holds mpe file numbers for each logical unit
7.00 c uinuse true if a file has been iopked

FILE UZPRV

C include file uzprv

parameter lenuzr=61,dbalev=3,maxsec=50
common /uzprv/uzrnam,uzrgrp,uzrlev,rdprm,wrtprm,modprm(maxsec)
common /uzprv2/readok,writok
character*8 uzrnam,uzrgrp
integer uzrlev
logical rdprm,wrtprm,modprm,readok,writok
---point alignment
integer alin07(lenuzr)
character*40 fld07(6)
equivalence (alin07,uzrnam)
---module permission equivs
logical runmnu,crscen
equivalence (crscen,modprm(12))

holds user privilege info extracted from sysusr.pub

uzrnam name of this user
uzrgrp group user may execute from (usually home or "ANY")
uzrlev user privilege summary level:1=read;2=reg;3=dba;4=super
rdprm true if user may read data base
wrtprm true if user may write to data base
modprm (i) is true if user may execute module (i)
readok true if user may read this scenario
writok true if user may alter this scenario

10-313
Table 10-8. ALIAS Extra Data Segment Usage

<table>
<thead>
<tr>
<th>SEGMENT</th>
<th>USER</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>636</td>
<td>BUILDER</td>
<td>ID of segment which is used for communication with the default RELATE son process.</td>
</tr>
<tr>
<td>9012</td>
<td>DBU</td>
<td>Segment used for storage of cursors associated with the various RELATE son processes which the DBU starts up; i.e. DBU file management subsystem's global storage.</td>
</tr>
<tr>
<td>1</td>
<td>Core</td>
<td>Segment used by mrump/iniprc FORTRAN routines to swap contents of Core common blocks into son process data memory. Used only as a communication segment.</td>
</tr>
<tr>
<td>101</td>
<td>DBU</td>
<td>Segments used for communication between the DBU and the various RELATE son processes the DBU starts up using its file management subsystem.</td>
</tr>
<tr>
<td>102</td>
<td></td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SCEN</td>
<td>Segment used by the scenario system to store the SCENARIO field key values for each relation for the current scenario.</td>
</tr>
<tr>
<td>8001</td>
<td>FLRP</td>
<td>Temporary storage buffer used by the Force/BGRP Battle Group Report Generators' internal page printing system.</td>
</tr>
<tr>
<td></td>
<td>BGRP</td>
<td></td>
</tr>
<tr>
<td>191</td>
<td>ASGN</td>
<td>Temporary storage for partially processed tuples during the update phase of execution.</td>
</tr>
<tr>
<td>201</td>
<td>DBIF</td>
<td>Sequents used for RELATE son processes supervised by the DBIF.</td>
</tr>
<tr>
<td>202</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
It is crucial that modules using extra data segments, either explicitly or implicitly via the __MEM FORTRAN utilities, not compete with one another for the same segment by specifying identical segment id numbers in the call to the getdseg intrinsic. Errors which are very difficult to trace can result from one module writing over another module's segment.

Table 10-9 presents a list of the current usage of segment id numbers.
<table>
<thead>
<tr>
<th>LPRNT MODULE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 LIST MEMORY</td>
<td>INLIST and OUTLIST routine diagnostics</td>
</tr>
<tr>
<td>2 HRELATE</td>
<td>High level RELATE utility diagnostics</td>
</tr>
<tr>
<td>3 LRELATE</td>
<td>Low level RELATE utility diagnostics</td>
</tr>
<tr>
<td>4 MENU SYSTEM</td>
<td>Input checking for MNUGEN</td>
</tr>
<tr>
<td>5 DATA ENTRY</td>
<td>Turns on file echo; uses DMTEST version</td>
</tr>
<tr>
<td>6 DATA ENTRY</td>
<td>Inhibits record adds</td>
</tr>
<tr>
<td>7 ASSIGNER</td>
<td>All Manual assigner system diagnostics</td>
</tr>
<tr>
<td>8 UTILITIES</td>
<td>filopn</td>
</tr>
<tr>
<td>10 ASSIGNER</td>
<td>Outbound high-level diagnostics; yard-oriented</td>
</tr>
<tr>
<td>10 ASSIGNER</td>
<td>routines asycls; ascmpg; asndbr; asnoui</td>
</tr>
<tr>
<td>11 ASSIGNER</td>
<td>Outbound job-of-interest diagnostic; asnjo</td>
</tr>
<tr>
<td>12 ASSIGNER</td>
<td>Removal of hist/curr jobs from buffer before out</td>
</tr>
<tr>
<td>12 ASSIGNER</td>
<td>outbound processing. Routine asncin.</td>
</tr>
<tr>
<td>13 ASSIGNER</td>
<td>Echoes relate procedure file to screen which upd</td>
</tr>
<tr>
<td>13 ASSIGNER</td>
<td>updates hull numbers in ncjodat.proj. asnhul</td>
</tr>
<tr>
<td>14 ASSIGNER</td>
<td>High level class-oriented processing outbound.</td>
</tr>
<tr>
<td>14 ASSIGNER</td>
<td>Routines asgnxt ashtrb ashard astupf</td>
</tr>
<tr>
<td>15 ASSIGNER</td>
<td>Low level print for ashtrb; cvnt buf to rcrd</td>
</tr>
<tr>
<td>16 ASSIGNER</td>
<td>Low level outbound hardware chex; ashard</td>
</tr>
<tr>
<td>17 ASSIGNER</td>
<td>Low level for astupf; produces tuple images.</td>
</tr>
<tr>
<td>18 ASSIGNER</td>
<td>Outbound actual DB update. asndbr asodel</td>
</tr>
<tr>
<td>19 ASSIGNER</td>
<td>Outbound job description retrieval. aspfld asgpf</td>
</tr>
<tr>
<td>20 ASSIGNER</td>
<td>Outbound schedule date calculators. ascdsp ascds</td>
</tr>
<tr>
<td>22 ASSIGNER</td>
<td>Aspred: outbound date spreading</td>
</tr>
<tr>
<td>25 SCENARIO</td>
<td>All scenario system debug prints</td>
</tr>
<tr>
<td>26 UTILITY</td>
<td>Extra data segment system (xxxMEM)</td>
</tr>
</tbody>
</table>
FORCE IMPACT ASSESSMENT MODULE

11.1 PURPOSE

The force impact assessment module projects future Navy force levels in terms of both raw numbers of deployable ships and deployable battle groups. It can also be used to report on past force levels if that is desired. The module is designed to permit report contents and formats to be customized to a high degree.

Sample reports of each type are shown in Figures 11-1 and 11-2.

11.2 SUMMARY OF STRUCTURE

The module is centered around two independently executable programs, each producing one of the two types of report. The programs are executed as son processes when options 2 and 3 of the Force Level Report Generator choice menu of the ALIAS Command System are chosen.

The programs are quite similar in internal structure and operation. Each reads a user-specified ASCII file, called a format control file, which specifies the contents and format of the report desired. Report contents are specified in terms of a list of ship classes for which force levels are desired (the classes may be combined into summary groups, referred to here as ship "types"). After obtaining the class list, each program searches the database for commissionings and decommissionings of ships in the classes (as defined by construction, conversion, and reactivation jobs and decommissioning dates), for repair jobs which temporarily take ships out of the force, and for a standard service lifetime for each class. The report is then constructed and written to the device the user specifies on the User Environment Parameter menu and, optionally, to a disk file.
Figure 11-1. Sample Force Level Report Generator Output.

POM 86 FORCE IMPACT PROJECTION
BASED ON STANDARD SERVICE LIVES
(ALL DATA NOTIONAL)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CV INVENTORY</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>CVN INVENTORY</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>CVN PROGRAM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CARRIER DEPLOYABLE</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>14</td>
<td>14</td>
<td>15</td>
<td>15</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>CV IN SLEEP</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CARRIER TOTAL</td>
<td>14</td>
<td>14</td>
<td>13</td>
<td>14</td>
<td>14</td>
<td>15</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>BB PROGRAM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BB TOTAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGN INVENTORY</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>CG INVENTORY</td>
<td>23</td>
<td>27</td>
<td>31</td>
<td>34</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>37</td>
<td>31</td>
<td>28</td>
<td>26</td>
<td>24</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>CG PROGRAM</td>
<td>2</td>
<td>5</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>CRUISER TOTAL</td>
<td>32</td>
<td>36</td>
<td>40</td>
<td>43</td>
<td>47</td>
<td>48</td>
<td>49</td>
<td>47</td>
<td>44</td>
<td>42</td>
<td>40</td>
<td>35</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>DDG INVENTORY</td>
<td>37</td>
<td>37</td>
<td>37</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>DDG TOTAL</td>
<td>37</td>
<td>37</td>
<td>37</td>
<td>38</td>
<td>38</td>
<td>42</td>
<td>46</td>
<td>52</td>
<td>57</td>
<td>62</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>DD INVENTORY</td>
<td>32</td>
<td>32</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>DD TOTAL</td>
<td>32</td>
<td>32</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>FFG INVENTORY</td>
<td>56</td>
<td>56</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>56</td>
<td>52</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>FFG TOTAL</td>
<td>56</td>
<td>56</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>56</td>
<td>52</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>FF INVENTORY</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>55</td>
<td>54</td>
<td>50</td>
<td>49</td>
<td>48</td>
<td>44</td>
</tr>
<tr>
<td>FF TOTAL</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>57</td>
<td>54</td>
<td>50</td>
<td>49</td>
<td>48</td>
<td>44</td>
<td>39</td>
</tr>
<tr>
<td>FRIGATE TOTAL</td>
<td>113</td>
<td>113</td>
<td>114</td>
<td>114</td>
<td>114</td>
<td>114</td>
<td>112</td>
<td>111</td>
<td>107</td>
<td>105</td>
<td>100</td>
<td>95</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>GRAND TOTAL</td>
<td>228</td>
<td>232</td>
<td>236</td>
<td>241</td>
<td>245</td>
<td>251</td>
<td>255</td>
<td>258</td>
<td>260</td>
<td>259</td>
<td>256</td>
<td>247</td>
<td>242</td>
<td>237</td>
</tr>
</tbody>
</table>
Figure 11-2. Sample Battle Group Report Generator Output.

**Deployable Battle Group Projection for POM-86**
**Based on Surface Combatant Requirements Only**
*(All Data Notional)*

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Battle Group</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Carrier BG</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Surface AG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Marine AF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supply Escort</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Convoy</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>8</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td><strong>Balance</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Carrier</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>BB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cruiser</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>DDG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DD</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>9</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFG</td>
<td>33</td>
<td>33</td>
<td>34</td>
<td>34</td>
<td>40</td>
<td>40</td>
<td>50</td>
<td>40</td>
<td>47</td>
<td>37</td>
<td>31</td>
<td>26</td>
<td>21</td>
<td>16</td>
</tr>
</tbody>
</table>

11-3
Figure 11-3. Structure of Force Analysis Modules
Since the actual numbers on the reports depend only on the contents of the data base, and particularly on schedules, force impact reports will always reflect the latest data available for a scenario.

The programs' only direct interaction with the user is the prompt for the name of the input format control file; all other control values, such as the dates specifying the period for which the report is desired, are taken from the settings of the Command System's Force Level Parameters Menu.

Although FLRP and BGRP are completely separate in terms of executability, they do share a good deal of source code, particularly the data base search logic.

This structure is summarized in Figure 11-3.

11.3 INPUTS AND OUTPUTS

This section will describe FLRP and BGRP inputs and outputs in more detail. They will be discussed here as though they were a single program, since the structure of their inputs and outputs is identical.

The sole outputs of the module (with the exception of prompts and error messages) are the reports themselves, in the general form shown in Figures 11-1 and 11-2. These reports are always written to the user's default hard copy output device (the one specified on the User Environment Parameters menu), and may also be saved in a disk file for editing by the user and subsequent re-printing.

Six types of input are required. In order of first use by the module, these are the System Core process data swap with its parameter menu contents, the name of the format control file from the user, the contents of the Out of Force Repair Jobs List Menu associated with the Force Level Parameters Menu in the Command
System, the contents of the format control file, scenario key field values for various relations as supplied by the scenario system, and the contents of various data base relations for the user's current scenario.

11.3.1 Core Swap Data Used

As described in Section 8, the System Core will swap out the contents of many of its key global arrays into an extra data segment just before activating a son-process module, if the developer desires. The data may then be read into identical global arrays in the son by a call to the iniprc utility.

FLRP and BGRP make use of this facility to obtain the current values of the variables appearing on the Force Level Parameter Menu in the Command System. These variables hold many of the control values for FLRP/BGRP program execution. In order to minimize the number of routines in which the /pvalue/ array must be included, the values are moved by the flinit routine into a common block called /flvalu/ which is used only by FLBG/FLRP.

A sample of the parameter menu is shown in Figure 11-4. In the order in which they appear, the use of the parameters is:

1) KEEP REPORT: If set to YES, the output of FLRP/BGRP will be saved in a file in the log-on group called flrept. If flrept already exists at module run time, the user is prompted for an alternative name.

2) REPORT START: The first day of the first period the user is interested in. Ships reaching final retirement before this date will never appear on a report regardless of the contents of the format control file. If the user specifies a date that is not the first day of its period, the date is moved back to the first day.

3) REPORT END: The last day of the last period the user is interested in. Determines the number of periods in conjunction with REPORT START.

4) RETIRE SHIPS BY: If DATE, the data base search logic will look for specific retirement dates for each ship in the deact.miscj relation for those ships not already retired as of the day the report is being run. If it
Figure 11-4. Command System Parameter and List Menus
Serving the Force Module

Menu is FLREPT * ALIAS COMMAND SYSTEM * Scenario is DEMO

-----------------------------------------------------------------------------------
FORCE LEVEL AND BATTLEGROUP REPORT GENERATOR PARAMETERS
1. KEEP REPORT ON-LINE = YES (YES, NO)
2. REPORT START DATE = 1/1/1986 (MM/DD/YYYY)
3. REPORT END DATE = 12/31/2005 (MM/DD/YYYY)
4. RETIRE SHIPS BY = LIFE (LIFE, DATE)
5. TIME PERIOD LENGTH = CALYR (DAY, WEEK, MONTH, QTR, CALYR)
6. IN FORCE DAY = END (BEGIN, END)
7. PROGRAM MILESTONE = APPROP (APPROP, AWD, DELIV)
8. OUT OF FORCE REPAIR JOBS = LIST (ALL/LIST)

COMMAND:

Menu is FLREPT * ALIAS COMMAND SYSTEM * Scenario is DEMO

-----------------------------------------------------------------------------------
REPAIR JOBS THAT REMOVE A SHIP FROM FORCE DURING EXECUTION
1. REFUEL
2. REPAIR
3. * SLEP
4. TESTRE

COMMAND:
cannot find a date specification there, the standard service life for the class, in combination with the amount of service the ship has seen (not including periods of deactivation), will be used to determine the final retirement date. If LIFE, then the standard service life will be used for all ships not retired by the day the report is run.

5) TIME PERIOD LENGTH: The time units the period of interest should be measured in. A variety of choices is available, but no report may span more than 20 periods.

6) IN FORCE DAY: There must be a rule to determine whether ships retiring in the middle of a period are in the force for that period or not. If this parameter is END then they are not; if BEGIN then they are.

7) PROGRAM MILESTONE: In the format control file the user may specify that the force availability for ships of each type be separated into multiple lines, or "programs", based on a milestone in each ship's construction process. The purpose of this is to let the user see, for example, the relative impact of ships already built compared to ships in the POM compared to those in the EPA. The user will specify start dates for each era or program in the control file; those dates will be compared to the construction/conversion/reactivation milestone date for each ship specified here. Thus if APPROP is chosen, ships appropriated after the first day of the POM era will be placed in POM lines on the report.

8) OUT OF FORCE REPAIR JOBS: This is a gate to a list menu whose role is discussed in Section 11.3.3.

11.3.2 Format Control File Name

Since format control files are just standard editor files containing a particular syntax, a large number and variety of them can exist. For this reason the user is prompted for the name of the file he wishes to use, rather than limiting the choice to a small selection of values on the parameter menu. Most public format control files are maintained in the .fmtfil group.

11.3.3 Out of Force Repair Jobs List Menu

A sample of this list menu is shown with the parameter menu in Figure 11-4. In this menu the user specifies which kinds of repair job will cause a ship to be temporarily out of the force.
level for purposes of a force impact study. The contents of the
candidates list on this menu is managed by the RR_JOB_TYPES
screen of the DBU, so the user can cause any or all of the legal
repair job type code names to appear on the list. He can turn
any, all, or none of them on (only SLEP is on in the sample). If
none are on then no temporary removals will be effected.

The out-of-force-repair-job concept was implemented to deal
with SLEP jobs in particular; it has been common to consider a
carrier in SLEP as not in the force level in force impact studies
in the past.

11.3.4 The Format Control File

Sample Force Level and Battle Group Report format control
files are shown in Figures 11-5 and 11-6. Syntax rules for the
files are discussed in the ALIAS User's Guide, and will be
discussed here only as appropriate.

Both files are keyword-oriented; that is, the logic which
reads them identifies the type of data appearing on a given line,
and the actions to take on that data, according to keyword com-
mands which appear as the first word on the line. Where a line
must be longer than 72 characters (the maximum width allowed)
then continuation lines beginning with a "+" may follow it.

In both files the TITLE keyword must appear before other
keywords; the same applies to PRGLB in the Force Level file.

Note that although the Force Level format control file is
larger, the Battle Group file has a larger variety of keywords
and is more complex. Here key word lines must appear in groups
and strictly in the order TYPE, FUNCTION, BGROUP, MAKEUP. The
user specifies the TYPES of ships available in terms of classes;
then the FUNCTIONS each type can perform; then the battle groups
desired and the target number and priority for each (note that a
This is a force level report format control file. Any line beginning with '%' is considered a note and ignored.

The next two lines tell FLRP to split the force level into two lines for each class, based on ship approp date.

PRGLB Inventory, 1/1/1990
PRGLB Program, 10/1/1986

The TITLE lines give the title that will be printed (centered) on the top of each report page.

TITLE POM 86 Force Impact Projection
TITLE Based on Standard Service Lives
TITLE (All Data Notional)

Start the report specification. BTOT lines tell FLRP to start keeping a running total, ETOT where in the body to print the total; last two words on ETOT lines are the left and right labels actually printed. Label on BTOT line and first one on ETOT for internal FLRP use. EITOT is like ETOT except specifically designed for subtotals; it ensures no page feed happens in the middle of a type being printed.

TYPE lines specify ship types by giving the names of all the classes in the type.

START
BTOT grand
BTOT subn
TYPE SSBN, SSBN-726, SSBN-640, SSBN-627, SSBN-616, SSBN-611, SSBN-610, SSBN-609, SSBN-601, SSBN-599, SSBN-598
ETOT subn, SSBN, TOTAL
BTOT sub
ETOT sub, SSN, TOTAL

Note job line causes carriers in SLEP to be printed; they do not appear in the deployable total.

BTOT carrier
BTOT dcarrier
TYPE CV, CV-41, CV-59, CV-63, CV-67
TYPE CVN, CVN-65, CVN-68
EITOT dcarrier, CARRIER, DEPLOYABLE
JOB CV, IN SLEP, SLEP, CV-41, CV-59, CV-63, CV-67
ETOT carrier, CARRIER, TOTAL

BTOT bb
TYPE BB, BB-61
ETOT bb, BB, TOTAL
Figure 11-5. Sample Force Level Report Format Control File

<table>
<thead>
<tr>
<th>Type</th>
<th>Subtypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cruisers</td>
<td>CGN-25, CGN-36, CGN-38, CGN-35, CGN-9, CG-16, CG-26, CG-47, CRUISER, TOTAL</td>
</tr>
<tr>
<td>DDG</td>
<td>DDG-2, DDG-37, DDG-51, DDG-993, DDG, TOTAL</td>
</tr>
<tr>
<td>DD</td>
<td>DD-945, DD-963, DD, TOTAL</td>
</tr>
<tr>
<td>FFG</td>
<td>FFG-1, FFG-7, FFG, TOTAL</td>
</tr>
<tr>
<td>FF</td>
<td>FF-1037, FF-1040, FF-1052, FF, TOTAL</td>
</tr>
<tr>
<td>Mine</td>
<td>MCM-1, MSH-1, MSO-422, MTS, MINE SHIPS, TOTAL</td>
</tr>
<tr>
<td>Grand</td>
<td>GRAND, TOTAL</td>
</tr>
</tbody>
</table>

STOP
Figure 11-6. Sample Battle Group Report Format Control File

% ALIAS BATTLE GROUP REPORT FORMAT/CONTENTS DEFINITION FILE
% format is: title; start; type; function; bgroup; makeup; end
% title line has titles for report
% start line indicates start of processing
% type line indicates ship classes making up a type
% function line lists types which can perform a function, in
% order of preference
% bgroup describes battle groups to be made up
% makeup describes which functions each battle group requires
%
TITLE Deployable Battle Group Projection For on ROM-86
TITLE Based on Surface Combatant Requirements Only
TITLE (All Data Notional)
%
START
%
% type format similar to force level report: name,label,class list
%
TYPE CARRIER, CARRIER, CV-41,CV-59,CV-63,CV-67,CVN-65,CVN-68
TYPE BB, BATTLESHIP, BB-61
TYPE CRUISER, CRUISER, CGN-25,CGN-36,CGN-38,CGN-35,CGN-9,CG-16,CG-26,CG-47
TYPE DDG, DDG, DDG-2,DDG-37,DDG-51,DDG-993
TYPE DD, DD, DD-945,DD-963
TYPE FFG, FFG, FFG-1,FFG-7
TYPE FF, FF, FF-1037,FF-1040,FF-1052
%
% function format is name,list of types which can perform it
% in order of preference
FUNCTION CRUISER, CRUISER, BB
FUNCTION CARRIER, CARRIER
FUNCTION DDG, DDG
FUNCTION DD, DD
FUNCTION FRIGATE, FFG, FF
%
% bgroup format is name,output label,priority,target level,
% begin date this defn takes effect, end date this defn effective
BGROUP CVEG, CARRIER BG, 1,15,1/1/1900,1/1/2111
BGROUP SAG, SURFACE AG, 3, 4,1/1/1900,1/1/2111
BGROUP MAP, MARINE AF, 2, 2,1/1/1900,1/1/2111
BGROUP ESC, SUPPLY ESCORT, 4,10,1/1/1900,1/1/2111
BGROUP CON, CONVOY, 5,10,1/1/1900,1/1/2111
%
% makeup format is battle group name, function, #reqd, func, #reqd
MAKEUP CVEG, CARRIER, 1, CRUISER, 1, DDG, 4, DD, 2, FRIGATE, 4
MAKEUP SAG, CRUISER, 2, DDG, 2, FRIGATE, 2
MAKEUP MAP, CRUISER, 2, DDG, 2, DD, 8, FRIGATE, 10
MAKEUP ESC, DDG, 1, DD, 1, FRIGATE, 2
MAKEUP CON, DD, 1, FRIGATE, 4
%
STOP
given target applies through the date given on the BGROUP line); then the functions which are required to MAKEUP each group.

The Force Level format file specifies only TYPES, since the force report is raw numbers available by type, but also has the capability to total and subtotal types. A stack-like logic is used in which the user "pushes" another total onto the list of those FLRP is making up with a BTOT line, and "pops" it off (causing it to be printed) with an ETOT or EITOT line. Types and totals will appear on the output in the order in which they appear in the file.

Likewise, battle groups and type balances appear on the Battle Group report in the order in which they are named in the Battle Group report control file.

The necessity of letting the user specify both the contents and order-of-output of both types of report was what prompted the use of format control files. These are non-standard ALIAS constructs because they require the user to know some syntax, use the editor, and operate in a fairly unsupervised and unaided fashion. However, a method of report specification relying only on standard facilities such as list menus would have been very clumsy and limiting.

11.3.5 Scenario Key Field Values

FLRP and BGRP make use of scenario system services via the DBIF and the contents of the cursen array in the /scenar/ common block (i.e. in the usual fashion) when constructing search keys for retrievals from the relations.

11.3.6 Relations

FLRP and BGRP read the contents of ten relations.

Vlrjob.mnurel holds the contents of the Out-Of-Force-Repair-Jobs list, and is searched by the programs via a call to
the liston utility routine in order to recover the names of any job types which are "on". Liston also reads the lccref.mnurel cross referencing relation, which is opened by iniprc. These relations are managed by the Command System and need be of no great concern.

The ncjodat.histj, ncjodat.currj, and ncjodat.projj relations hold schedules for historical, current, and projected new construction, conversion, and reactivation jobs. These schedule records are vital data for this module, since they indicate the number of ships that enter the force over time and the timing of each entry.

The .histj and .currj versions of the relations can contain both actual and projected schedule records for a given ship, and for multiple data dates. The rule used by this module for selecting which single record to use for a given ship is based solely on datadate: the record with the latest datadate is used. The intuition behind this is that regardless of whether the DATETYPE field indicates the data is actual or projected, the record with the latest datadate is most likely to contain the Navy's best guess as to the commissioning of a given ship. Note that if no commissioning date is given, the delivery date is used instead, and that if no delivery date is given, the ship is simply ignored.

The rejodat.histj, rejodat.currj, and rejodat.projj relations form a similar structure containing repair job schedule records. They are searched with similar rules, but their data is less central to force impact studies since they are consulted only for job types which are "on" in the Out-Of-Force-Repair-Jobs list menu.

The deact.miscj relation holds actual and projected ship deactivation dates (note that a known deactivation date will appear here regardless of which new construction schedule rela-
tion the given ship's activating job record appears in). This relation is searched for every activation found in the ncjodat relations. This date is used if it is earlier than the date the report is being run on (i.e., the ship has already retired), or if the user has chosen DATE for the RETIRE SHIPS BY parameter.

The shlife.miscj relation must have one record for each ship class which appears in a report (this condition is met automatically as long as the DBU is used for data base maintenance). The record gives the standard service life for ships of that class, which is used to estimate a given ship of that class's retirement date if no deactivation date projection can be found for the ship in deact.miscj (or if the LIFE option is chosen on the parameter menu).

The relations are accessed through standard DBIF calls. FLRP and BGRP do require that a number of special indexes exist for the relations to support their POINT-oriented search logic.

11.4 DATA STRUCTURES

11.4.1 Data Structures Used by Both Programs

Both FLRP and BGRP use the data relations discussed in the preceding section, the relevant format control file, an extra data segment, a direct-access ASCII file, and a number of common blocks.

The only additional thing to be noted about the relations is that both the repair and the new construction schedule relations are each opened twice. The construction relations are opened twice on the same index (and naturally on different cursors/partitions since the DBIF is used) because after a construction/conversion job is found for a given ship, a search must be conducted for possible reactivation jobs. The two searches would interfere with one another if conducted through
the same cursor. The repair relations are opened on different indexes to support two different kinds of searches.

One of the problems which had to be solved for FLRP in particular was the matter of appropriate location of page feeds as the report file is printed. It is desirable to have all the lines in a particular subtotal group appear on the same page (the assumption here is that in reports with multiple lines for a TYPE totaling will be specified over the lines, as is done in the sample in Figure 11-5). To ensure this, output lines are sent to a holding buffer rather than directly to the output file. The contents of the buffer are flushed to output only when an ETOT line is encountered. This buffer is actually an extra data segment (it was originally a common block, but memory limits required use of the segment). Transfer of lines to and from the segment is managed by calls to the _mem utilities (e.g. getmem).

Rather than being sent directly to the unit which is to produce the hard copy output (a problem if the output device is not spooled, since exclusive access will be required, thus tying the device up), report lines are instead sent (from the extra data segment buffer) to a sequential-access ASCII file. When the report is complete the contents of the file are read and sent to the output device in a tight loop. If the user has specified that the report be kept in a disk file, then the given file is just saved rather than being deleted.

Table 11-1 presents an annotated listing of the common blocks used by the Force Impact module programs. These common blocks form the principal working data structure for computational purposes.

The most important block used by both programs is contained in the fltabs.incl include file. A large array, indexed by ship classes, period, and programs (FLRP format control PRGLB keyword)
# TABLE 11-1. Include Files Used By the Force Report Generators.

<table>
<thead>
<tr>
<th>FILENAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGPMTR</td>
<td>FORTRAN Parameters defining battle group report generator capacity limits, e.g. maximum number of functions definable.</td>
</tr>
<tr>
<td>BGTITL</td>
<td>Title of the battle group report section now being printed (either BATTLEGROUP or BALANCE).</td>
</tr>
<tr>
<td>FLCLASS</td>
<td>List of ship classes named by the user on TYPE lines in the format control file, i.e. list of classes whose ships are to be retrieved from the data base and their force increments computed.</td>
</tr>
<tr>
<td>FLCONCH</td>
<td>FORTRAN parameters defining key words acceptable in the force level report format control file.</td>
</tr>
<tr>
<td>FLCONS</td>
<td>Relation names, index lists, field lists, and record buffers for opening and retrieving data from all three of the ncjodat.(projj currj histj) new construction schedule relations.</td>
</tr>
<tr>
<td>FLDECM</td>
<td>Same as FLCONS but for the deactivations relation deact.miscj.</td>
</tr>
<tr>
<td>FLHEAD</td>
<td>User-specified titles to appear on the report and the period-labeling portion of the page header.</td>
</tr>
<tr>
<td>FLIOC</td>
<td>FORTRAN io unit assignments for flrp and bgrp. Included are the format control file unit number, the hard-copy output unit, and the save-to-file unit.</td>
</tr>
<tr>
<td>FLJLST</td>
<td>List of force-affecting repair job types, i.e. those repair jobs which cause a ship to be temporarily removed from the force level while undergoing the jobs.</td>
</tr>
<tr>
<td>FLPAGE</td>
<td>Line numbers and record used to manage/communicate with the extra data segment in which output is temporarily stored by flbg's full-page printing subsystem.</td>
</tr>
<tr>
<td>FLPERD</td>
<td>Number of periods being considered this run and an array holding the date of the first day of each period.</td>
</tr>
<tr>
<td>FLPMTR</td>
<td>FORTRAN parameters defining flbg capacity limits, e.g. maximum number of classes specifiable on TYPE</td>
</tr>
</tbody>
</table>
### TABLE 11-1. Include Files Used By the Force Report Generators.

<table>
<thead>
<tr>
<th>FILENAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLRJOB</td>
<td>Like FLCONS but for the repair job schedule relations rejodat. (projj currj histj).</td>
</tr>
<tr>
<td>FLTABS</td>
<td>The array constructed by the data base search algorithm, containing the number of deployable ships in each class by program type in each period.</td>
</tr>
<tr>
<td>FLTOTL</td>
<td>The arrays holding the running totals which are output on force level reports when an ETOT or EITOT keyword is found in the control file.</td>
</tr>
<tr>
<td>FLVALU</td>
<td>The values of the variables shown as parameters on flbg's command system parameter menu, as extracted from the /pvalue/ array by the flinit routine.</td>
</tr>
<tr>
<td>FUNCBG</td>
<td>Names of battle group functions, as defined on FUNCTION lines, and array locations of the TYPEs which can perform those functions.</td>
</tr>
<tr>
<td>GROUPBG</td>
<td>Information describing the battle groups defined as desired on the BGROUP lines of battle group format control files. See also MKUPBG.</td>
</tr>
<tr>
<td>INCPAR</td>
<td>System Core (command system) capacity defining FORTRAN parameters, mainly used here to specify terminal input line length.</td>
</tr>
<tr>
<td>IOC</td>
<td>Standard ALIAS FORTRAN io unit numbers; mainly in and iout used here.</td>
</tr>
<tr>
<td>LPRNNTS</td>
<td>Array of diagnostic print switches.</td>
</tr>
<tr>
<td>MKUPBG</td>
<td>Linked list of functions (and amount of each) required to make up each battle group.</td>
</tr>
<tr>
<td>READC</td>
<td>Line number of last line read from input file.</td>
</tr>
<tr>
<td>SCENAR</td>
<td>Scenario system information; current scenario name and scenario field key value for queries on each relation opened using the DBIF.</td>
</tr>
<tr>
<td>SHLIFE</td>
<td>Like FLCONS but for the ship class standard lifetime specification relation shlife.mscj.</td>
</tr>
</tbody>
</table>
TABLE 11-1. Include Files Used By the Force Report Generators.

<table>
<thead>
<tr>
<th>FILENAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDDATE</td>
<td>The ALIAS date data type/date utility system declarations file.</td>
</tr>
<tr>
<td>TODAYC</td>
<td>Today's date in ddate form and the maximum possible ddate.</td>
</tr>
<tr>
<td>TYPEBG</td>
<td>Storage for availability figures by period for each TYPE defined on bgrp format control file TYPE lines; also labels and type names.</td>
</tr>
</tbody>
</table>
is declared here. The raw results of the data base search (number of ships of each class available in each period, by era of construction) are placed here for refinement into the final report format.

11.4.2 Key BGRP Common Blocks

In addition to these data structures, program BGRP makes use of several additional important common blocks. The /groupbg/ block contains the name, priority, etc. for each requested battle group as well as the actual numbers which are computed to be achievable. The /mkupbg/ block holds an array, managed as a linked list, which lists the "functions" which make up each battle group and their numerical requirements. /funcbg/ contains information which supports cross-referencing between function names and lists of ship "types" which can perform the functions, and /typebg/ contains the number of each type available in each period (summarized from the contents of /fltabs/).

11.5 PROCESSING LOGIC

11.5.1 FLRP

Table 11-2 lists the routines which comprise the FLRP program (not including general-purpose ALIAS FORTRAN utilities or the routines in the DBIF) and indicates which source file family they reside in. Table 11-3 provides a complete annotated listing of all the routines in FLRP and BGRP. See Section 11.8 for complete abstracts of routines. A calling tree diagram for FLRP appears in Figure 11-7.

This section will summarize the logic of the program.

FLRP initialization includes retrieval of the data placed in the swap segment by the Core, retrieval of the list of Out-Of-Force-Repair-Jobs, prompting for the name of the format control file, and opening of all the relations required. This activity is supervised by the flinit routine.
### TABLE 11-2. Alphabetical Listing of Routines in FLRP

A Program

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>HOST FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFLTBL</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLADPG</td>
<td>FLRPA</td>
</tr>
<tr>
<td>FLBRPT</td>
<td>FLRPA</td>
</tr>
<tr>
<td>FLBUGI</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLCHK1</td>
<td>FLRPA</td>
</tr>
<tr>
<td>FLCHK2</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLCHK3</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLCAS</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLCLOS</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLDECR</td>
<td>FLRPA</td>
</tr>
<tr>
<td>FIGLIF</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLINCL</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLINCR</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLINIT</td>
<td>FLRPA</td>
</tr>
<tr>
<td>FLJOB</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLNPER</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLNPXTP</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLPARS</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLPDAY</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLPMT</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLPQTR</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLPRT</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLPRTN</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLPROC</td>
<td>FLRPA</td>
</tr>
<tr>
<td>FLREK</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLPYER</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLRDCN</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLRDLN</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLREPT</td>
<td>FLRPA</td>
</tr>
<tr>
<td>FLRFN</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLTYPE</td>
<td>FLRPA</td>
</tr>
<tr>
<td>FLWRT</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLWTOP</td>
<td>FLRPA</td>
</tr>
<tr>
<td>FNDPRD</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>GETJOB</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>GETLIF</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>PAR2LN</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>PAR3LN</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>READFL</td>
<td>FLRPA</td>
</tr>
<tr>
<td>SKIPFL</td>
<td>FLBGxxx</td>
</tr>
</tbody>
</table>
Table 11-3. Annotated List of Force Level and Battle Group Report Generator Routines

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGADPG</td>
<td>Adds an availability-row line (i.e. a set of numbers by periods) to the line buffer (extra data segment).</td>
</tr>
<tr>
<td>BGBRPT</td>
<td>Like FLBRPT but more complex, this routine is the executive for actual production of the battle group report text. It processes the format control file and prepares the data structure for report production. Then calls BGGET and BGMRPT for output construction.</td>
</tr>
<tr>
<td>BGFUNC</td>
<td>Processes a battle group format control file function line. Principal output is the fnmlst and fdefine arrays, a list of defined function names and an array which allows cross referencing from the name to TYPE storage array elements.</td>
</tr>
<tr>
<td>BGGET</td>
<td>Computes battle group availability when /fltabs/ and format control file processing is complete. Outer loop is over periods, inner over battle groups in order of priority. Groups are made up with provisional decrementing of the available TYPE pools, which is committed when a group is fully constructed.</td>
</tr>
<tr>
<td>BGINIT</td>
<td>Zeros relevant arrays.</td>
</tr>
<tr>
<td>BGMKUP</td>
<td>Processes a MAKEUP line from the battle group format control file. Output is a linked list in the BGMKUP array which specifies which functions, and how many of each, are required to makeup the given battle group.</td>
</tr>
<tr>
<td>BGMRPT</td>
<td>Actually writes lines of the computed report to the line buffer (extra data segment).</td>
</tr>
<tr>
<td>BGPROC</td>
<td>Similar to FLPROC, makes the first pass through the battle group format control file in order to construct the list of ship classes of interest which ffltbl requires. Also reads and stores TITLE lines.</td>
</tr>
<tr>
<td>*BGREPT</td>
<td>Main program unit and chief executive for the BGRP battle group report generation program. Calls other high-level routines to do the actual work.</td>
</tr>
<tr>
<td>BGSETV</td>
<td>Increments a row array with a given value for the period between two given dates. Used to set the target number of each battle group; this target can</td>
</tr>
</tbody>
</table>
Table 11-3. Annotated List of Force Level and Battle Group Report Generator Routines

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGTYP</td>
<td>Processes TYPE lines from the battle group report format control file. Output is the TYPTOT array, which is the number of ships of all the classes named on the type line which are available in each period.</td>
</tr>
<tr>
<td>BGWRIT</td>
<td>Similar to FLWRIT, now unused.</td>
</tr>
<tr>
<td>BGWTOP</td>
<td>Writes titles and section header lines to the sequential storage file when a new page is called for.</td>
</tr>
<tr>
<td>BTLGRP</td>
<td>Processes BGROUP lines from the format control file. Output is the /groupbg/ common block describing the battle groups and their target and achieved amounts.</td>
</tr>
<tr>
<td>FFLTBL</td>
<td>Executive for the search of the data base for data on commissionings and decommissionings of classes of interest. The bulk of the logic is actually in this routine. It searches every construction and repair relation for jobs done on ships in the classes specified in the format control file, and appropriates increments the /fltables/ data structure. See the text for more information on the algorithm.</td>
</tr>
<tr>
<td>FLADPG</td>
<td>Writes an array line to the text buffer (extra data segment).</td>
</tr>
<tr>
<td>FLBRPT</td>
<td>Executive for actual creation of the Force Level report output. Re-reads the format control file and constructs output lines based on its directives and using the data created by FFLTBL.</td>
</tr>
<tr>
<td>FLBUGI</td>
<td>A service routine called to read a file line; written in response to a compiler bug which caused legal code using the usual utilities to be uncompileable.</td>
</tr>
<tr>
<td>FLCHK1</td>
<td>These three logical function utilities take one, two, and three pairs of string arguments, respectively and return whether or not they are equal. They are used mostly in checking tuples retrieved from relations to see if they have the proper key values.</td>
</tr>
<tr>
<td>FLCLAS</td>
<td>Subsidiary of FLPROC which supervises construction of...</td>
</tr>
</tbody>
</table>
Table 11-3. Annotated List of Force Level and Battle Group Report Generator Routines

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>the list of classes of interest. Takes a TYPE line, makes additions to /flclass/.</td>
</tr>
<tr>
<td>FLCLOS</td>
<td>Closes all files and relations. The act of closing the report output file starts printing if the &quot;file&quot; is in fact a spooled device.</td>
</tr>
<tr>
<td>FLDECR</td>
<td>Like FLDECR except subtracts one from the elements of the array. Used to remove a ship from the reported force for the periods it is out for repairs, if any.</td>
</tr>
<tr>
<td>FLGLIF</td>
<td>Given a construction/deactivation/reactivation history for an individual ship and its standard lifetime, calculates the ship's projected final retirement date.</td>
</tr>
<tr>
<td>FLINCL</td>
<td>Takes a character variable and a list in the form of an array and adds the variable's contents to the list if it is not already on the list. Used to, e.g., manage additions to the list of classes of interest.</td>
</tr>
<tr>
<td>FLINCR</td>
<td>Increments elements of a row of the main /fltables/ array (corresponds roughly to a row on the report output) by one for those elements representing the period between two given dates. Essentially, adds a ship to the reported force for its lifetime.</td>
</tr>
<tr>
<td>FLINIT</td>
<td>Main initialization routine for both FLRP and BGRP, with source code in recomp.src (since it must read the /pvalue/ data structure. Transfer parameter values from /pvalue/ into /flvalu/, prompts for and opens the format control file, opens the output file and/or device, and opens all the relations which will be involved in the base search.</td>
</tr>
<tr>
<td>FLJOB</td>
<td>Processes a JOB line from a force level report output control file, producing a line for the output report. Constructs a list of classes, taken from the input line, and calls GETJOB to find out how many ships of each class were out for each job in each period. Then formats and sends the output.</td>
</tr>
<tr>
<td>FLNPER</td>
<td>Using the start and end dates of interest and the time units specification from the parameter menu, figures out how many periods there are in the exercise.</td>
</tr>
</tbody>
</table>
Table 11-3. Annotated List of Force Level and Battle Group Report Generator Routines

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLNXTP</td>
<td>Figures out what report period (output column) a date falls in by comparing it with the array of period-start dates set up during initialization.</td>
</tr>
<tr>
<td>FLPARS</td>
<td>Parses an input string consisting of a list delimited by commas into individual elements, placing the elements in an array.</td>
</tr>
<tr>
<td>FLPDAY</td>
<td>A series of date utilities, written before the gdatep /gpern/fddate general purpose period utilities, which figure out how many periods of the given type there are between two given dates. Also, fills in the array of start dates of each period.</td>
</tr>
<tr>
<td>FLPMTH</td>
<td>/gpern/fddate general purpose period utilities, which figure out how many periods of the given type there are between two given dates. Also, fills in the array of start dates of each period.</td>
</tr>
<tr>
<td>FLPQTR</td>
<td>Integer function used to find out which program line a given job will fall on based on its basis date. I.e. will a job be, e.g., inventory (1) or program (2).</td>
</tr>
<tr>
<td>FLPRGN</td>
<td>During report construction, output lines are first stored in an extra data segment until a complete set of lines (i.e. including all associated totals) can be sent; the set of lines are then printed to a sequential holding file (the one the report will be saved in if the user has requested a save on disk). This routine rewinds the holding file and writes the report to the output device when report construction is completed.</td>
</tr>
<tr>
<td>FLPROC</td>
<td>Conducts the first read of the format control file, the object of which is construction of a list of the ship classes the user is interested in (/flclass/). This list is required by fftbl. Also looks for the PROGRAM keyword lines to find out how many lines to split each classes members into, and stores the user-specified TITLE lines.</td>
</tr>
<tr>
<td>FLRDCN</td>
<td>Utility for reads of the format control file. Calls FLRDLN and counts the number of lines returned.</td>
</tr>
<tr>
<td>FLRDLN</td>
<td>Reads a line from the file open on a given FORTRAN unit number.</td>
</tr>
</tbody>
</table>
### Table 11-3. Annotated List of Force Level and Battle Group Report Generator Routines

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>*FLREPT</td>
<td>Main program unit for FLRP, the force level report generator. An executive which calls six high-level routines to do the work of producing the report.</td>
</tr>
<tr>
<td>FLRPGN</td>
<td>Obsolete version of FLPRGN.</td>
</tr>
<tr>
<td>FLTYPE</td>
<td>Processes a format control file TYPE line during actual report construction, converting it into output lines. Uses the list of classes to search /fltabs/ for rows of deployable ships in each period, summing these rows, and writing out the result when finished.</td>
</tr>
<tr>
<td>FLWRIT</td>
<td>Writes the contents of the line buffer to the sequential-access output file.</td>
</tr>
<tr>
<td>FLWTOP</td>
<td>Writes the title and period header lines to the line buffer/extra data segment, i.e. starts a new page.</td>
</tr>
<tr>
<td>FNDPRD</td>
<td>Given a date, finds which period it belongs in. Obsolete.</td>
</tr>
<tr>
<td>GETJOB</td>
<td>Given a class name and job type code, searches the repair schedule relations for instances of that job on that class. Increments a row-array for each one found for the period the given ship is undeployable.</td>
</tr>
<tr>
<td>GETLIF</td>
<td>Retrieve (from shlife.miscj) and convert to days the standard lifetime of the ships in a given class.</td>
</tr>
<tr>
<td>GMAKUP</td>
<td>Returns a record from the linked list of functions comprising a given battle group in the /mkupbg/ common block.</td>
</tr>
<tr>
<td>PAR2LN</td>
<td>FBLG utility which takes a line of elements separated by commas, separates off the first element, and returns that element and the remainder of the line. Used to, e.g., extract labels from format control input lines.</td>
</tr>
<tr>
<td>PAR3LN</td>
<td>Like PAR2LN except returns the first two elements and the remainder.</td>
</tr>
<tr>
<td>READBG</td>
<td>Reads a line from a battle group format control file and decodes its keyword. Strips off the keyword and returns the rest of the line.</td>
</tr>
</tbody>
</table>
Table 11-3. Annotated List of Force Level and Battle Group Report Generator Routines

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>READFL</td>
<td>Reads a line from the FLRP format control file and decodes its keyword. Strips the keyword from the line and returns the remainder to the caller.</td>
</tr>
<tr>
<td>SKIPFL</td>
<td>Reads and discards lines from the format control file until a line with a START keyword is found.</td>
</tr>
</tbody>
</table>
Figure 11-7. FLRP Calling Tree Diagram
Flproc is then called to read through the format control file in order to construct a list of the classes of interest for the report, as specified on TYPE lines. Flproc also reads and stores away the user-specified report title and the program-era specifications given on any TITLE and PRGLB lines.

With this information in hand the search of the data base can be conducted. This search is managed by the ffltbl routine. The logic of the routine is built around (or inside) a loop over the list of classes of interest, as shown in Figure 11-8.

For each class, its standard service life is first retrieved. Then all jobs on ships of that class are retrieved from the ncjodat.(histj currj projj) relations, one relation at a time, from historical to projected. For each job found, an additional search of the ncjodat relations is made to see if there are any subsequent reactivations. Also, a search is made for a specific retirement date for the given ship. The proper program line(s) of a holding buffer are incremented for each period when the given ship was active. When all the activating jobs have been processed, a loop over the repair job relations retrieves all repair jobs of interest for the given class, and the holding buffer is decremented in the appropriate periods. When all processing is complete for the given class, the holding buffer is moved into the /fltabls/ storage array.

This step consumes almost all of the large amount of execution time required by FLRP. The low apparent rate of progress is caused by the large number of data base queries which are required; these queries are each relatively time consuming because of RELATE response time limitations.

Once the raw per-period force availabilities are computed by class, the actual output report can be constructed. This process is supervised by the flbrpt routine, which rewinds and
FOR EVERY CLASS OF INTEREST
   Get standard life of ships in class

   FOR EVERY NC RELATION (histj to projj)
      Find next job of interest in class
      Find associated decommissioning date
      Compute force level increment
      Look for reactivations
         When found, note life added and
         look for later decommissionings
   NEXT NC RELATION

   FOR EVERY RE RELATION (histj to projj)
      Look for jobs in this class that are
         job types turned "on" on list menu
         When found, decrement force level
   NEXT RE RELATION

   SAVE INFO FOR THIS CLASS INTO /FLTABLS/

   NEXT CLASS
re-reads the format control file, now processing every keyword line (except TITLE and PRGLB). In particular, for each TYPE line encountered, a list of the classes in the type is constructed and the per-period availability of each class is extracted from /fltabs/ and summed into a holding array. This array can be thought of as being composed of the rows which appear on the output—as many rows as there are eras or "program lines". The contents of the array are also added to any totaling buffers which are active (i.e. to as many rows of the /fltotl/ block as there have been BTOT lines given). Then the program rows are formatted and sent to the output buffer by calls to fladpg.

When an ETOT or EITOT keyword is encountered the "topmost" total row is sent to the output buffer and the number of active totals is decremented by one.

The JOB keyword line is unusual. Its purpose is to allow ships temporarily in out-of-force-repair-job status to appear on the report, so that they may be totaled. In this way an accurate representation of the number of ships actually in existence may be given in addition to an accurate representation of the number deployable. JOB line processing is undertaken by the fljob routine, which takes the given class list and searches the rejodat,(histj currj projj) relations for instances of jobs of interest on ships in the classes. Any found were certainly removed from the deployable force totals by the logic in ffltbl, so no double counting can result.

When flbrpt has completed processing of the format control file all work is essentially done. The flprnt routine rewinds the sequential disk file to which all output has been sent and writes its contents to the output device. Flclos then closes all files and relations and program processing terminates with a STOP. This automatically reactivates the System Core process and the user is returned to the Force Impact choice menu.
11.5.2 BGRP

An alphabetical list of the routines in program BGRP is given in Table 11-4. A calling tree diagram for BGRP is given in Figure 11-9. The logic of this program is very similar to that of FLRP up to the point of actual construction of the report. Note that ffiltbl is used in both cases to conduct the data base search.

Bgbrpt (Battle Group Build RePoRt) faces a much different task than does flbrpt, however. Instead of summarizing the number of ships deployable in fairly raw terms, this routine must allocate scarce resource (the ships) among competing demands (the battle groups).

It does this on a period-by-period basis (i.e. its outer loop is over periods)---the allocation in any one period is independent of that in any other. Within a period, ships are allocated to the highest-priority battle group until its target number is reached, or until a constraint makes it impossible to have more of that particular group.

The requirements of each group are specified in terms of (possibly) broad functions, each of which may be filled (in descending order of preference) by several ship types, each of which in turn may be composed of several classes.

The type, therefore, is the lowest common denominator for purposes of the allocation. The number of ships of each type available in each period is computed from the contents of /fltabs/ and placed in the /typtot/ summary array before allocation begins.

The allocation is done on a trial-and-error basis, proceeding in order of priority and preference. For example, when computing the number of carrier battle groups as specified in the format control file in Figure 11-6, bgget (the allocation
<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>HOST FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGADPG</td>
<td>BGRPxxx</td>
</tr>
<tr>
<td>BGBRPT</td>
<td>BGRPxxx</td>
</tr>
<tr>
<td>BGFUNC</td>
<td>BGRPxxx</td>
</tr>
<tr>
<td>BGGET</td>
<td>BGRPxxx</td>
</tr>
<tr>
<td>BGINIT</td>
<td>BGRPxxx</td>
</tr>
<tr>
<td>BGMRPT</td>
<td>BGRPxxx</td>
</tr>
<tr>
<td>BGPROC</td>
<td>BGRPxxx</td>
</tr>
<tr>
<td>BGREPT</td>
<td>BGRPxxx</td>
</tr>
<tr>
<td>BGSETV</td>
<td>BGRPxxx</td>
</tr>
<tr>
<td>BGTYPE</td>
<td>BGRPxxx</td>
</tr>
<tr>
<td>BGWRIT</td>
<td>BGRPxxx</td>
</tr>
<tr>
<td>BGWTOP</td>
<td>BGRPxxx</td>
</tr>
<tr>
<td>BTLPGRP</td>
<td>BGRPxxx</td>
</tr>
<tr>
<td>FFLTB</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLADPG</td>
<td>FLRPA</td>
</tr>
<tr>
<td>FLBUGI</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLCHK1</td>
<td>FLRPA</td>
</tr>
<tr>
<td>FLCHK2</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLCHK3</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLCLAS</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLCLCS</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLDIF</td>
<td>FLRPA</td>
</tr>
<tr>
<td>FLCLIT</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLINCR</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLINIT</td>
<td>FLRPA</td>
</tr>
<tr>
<td>FLJOB</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLNPER</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLNXTP</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLPARS</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLPDAY</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLPMTH</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLPOTR</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLPRGN</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLPRNT</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLRWEK</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLPYER</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLRDCH</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLRDLN</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLRAGN</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLRIT</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>FLTOP</td>
<td>FLRPA</td>
</tr>
<tr>
<td>FNMPRD</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>GETJDB</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TABLE 11-4. Alphabetical Listing of Routines in BGRP Program

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>HOST FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETLIF</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>GMARUP</td>
<td>BGRPxxx</td>
</tr>
<tr>
<td>PAR2LN</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>PAR3LN</td>
<td>FLBGxxx</td>
</tr>
<tr>
<td>READBG</td>
<td>BGRPxxx</td>
</tr>
<tr>
<td>READFL</td>
<td>FLRPA</td>
</tr>
<tr>
<td>SKIPFL</td>
<td>FLBGxxx</td>
</tr>
</tbody>
</table>
Figure 11-9. BGRP Calling Tree Diagram

```
FLINIT  BGINIT  BGPROC  PFLTBL  BGRPT  FLPRT  FLCLS
  FLNPER  PAR2LN  FLPRGN  READBG
   FLPDAY  FLINCR  FLGLIF  FLPIFL
   FLEWEEK  FLINCL  FLRDCN  FLRDLN
   FLPMTK  FLCHK2  READBG  FLNXTP
   FLPQTR  FLPARS  PAR2LN
   FLPYER  FLINCR  BGWTOP

BGMKUP
  PAR3LN
  FLINCL
  PAR2LN
  BGSETVE
  READBG

BGMRPT
  FLWRIT
  BGADPG

BGGET
  GMAKUP
```
executive) would start by making up one battle group. It would do this by decrementing 1 carrier from function carrier, i.e. from the number of ships of type carrier available; then it would decrement 1 cruiser from function/type cruiser. If there were no ships left of type cruiser, it would try type BB, since that is an alternative for the cruiser function.

The decrementing that is taking place is being done on a temporary copy of the type availability array, so that if construction of a given group cannot be completed no "backing out" must be done to restore the actual count available.

Once all computations are complete the report is written out and flprnt and flclos are called to close files and clean up.

11.6 FILES USED BY THE FORCE IMPACT MODULE

Source code for FLRP alone is in flrpa.src. That used only by BGRP is in bgrpa.src, bgrpbgi.src, and bgrpbgw.src. Code for routines used jointly is in flbga.src, flbgflg.src, flbgflp.src, and flbgflr.src. Object code is in the complementary files in the .obj group. Combined (PREPable) object code is in flrp.obj and bgrp.obj. Program files are tflrp.prog, flrp.prog, tbgrp.prog, and bgrp.prog (development and production versions).

The format control input files are conventionally stored in the .fmtfil group.

Default output file when the report is saved to disk is flrept in the log-on group for both BGRP and FLRP.

Relations used are iccref.mnurel, vlrjob.mnurel, ncjodat.projj, ncjodat.currj, ncjodat.histj, rejodat.projj, rejodat.currj, rejodat.miscj, deact.miscj, and shlife.miscj.
11.7 SUMMARY OF INTERFACES

The Force Impact Module is fairly independent of other system components. It does use the standard Core services (scenario system, DBIF, and swap of Core data via a call to iniprc). The module is very dependent on the structure of the data base. Any change to the file or indexing structures of the relations listed in the previous section will be likely to render the module inoperative.

11.8 SUBROUTINE ABSTRACTS

Abstracts for both program FLRP and program BGRP are given on the following pages in alphabetical order. See Table 11-4 for a summary of the routines.
*** FORMAL PARAMETER DECLARATIONS ***

*** ABSTRACT ***

exec for battle group force level report generator

MEMutchler 27-JUN-83 AUTHOR

main program

none

menu system choice menu

C Initialize and open necessary relations and files.
C Parse output control file creating an alphabetized list of
C all ship classes found on type lines. Fill in force level table,
C one row for each ship class found, one column for each time period,
C as number of ships built of that class in that time period - number
C of ships of that class and period out for major deactivaing jobs or
C in temporary retirement. Process output control file along with
C the force level table to build battle group force level report file.
C Print force level report file.

C LOCAL VARIABLES
C err  error flag
C
SUBROUTINE BGADPG(TOTARRY, LAB)
   ** FORMAL PARAMETER DECLARATIONS **

   %INCLUDE FLPMTR
   %INCLUDE FLPERO
   %INCLUDE FLHEAD
   CHARACTER LAB*LENRLB
   INTEGER TOTARRY(MXPERD)

   ** ABSTRACT **

   ** PURPOSE ** write total line to pagebuf
   ** AUDIT HISTORY **
   C          MEMutchler     28-may-83    AUTHOR
   C          force level io routine
   C          none
   C          flipage holds pagebuf
   C          flhead holds output text specs
   C          fltotl holds line to be output
   C          flperd holds period info
   ** METHOD **
   C          write text to buf keeping track of lines used
   C**
SUBROUTINE BGBRPT (ERR)

C* *** FORMAL PARAMETER DECLARATIONS ***
C* LOGICAL ERR

C* *** ABSTRACT ***
C* PURPOSE use output control file and fltabl to create
C* force level report file
C* AUDIT HISTORY
C MEMutchler 23-may-83 AUTHOR
C*TYPE force level report utility
C*FORMAL PARAMETERS
C cin err error flag
C cin incpar global parameters
C cin charcon output control file keywords
C cin readc holds line number last read
C cio fltotl holds totaling arrays
C cin fltabs holds force level tables for each program
C*CALLER fireport
C*METHOD
C process output control file by line creating the structure
C for a force level report use values in fltabs for data.
C Determine which type of line just read from output
C control file and process accordingly
C START read everyline between a START and STOP line
C only title and program labels are acknowledged
C prior to the initil start
C + this is a continuation of the text of last line, only
C for JOB or TYPE lines
C TITLE center text on the top of each report page
C must be read before initil start
C TYPE text=typename,classes to make up type: defines a type
C get values from force level tables for program types
C and add class levels together
C JOB text=typename,jobname,job as known to RELATE,classes
C in type: total all job done to each class for programs
C LOCAL VARIABLES
C protot program totaling arrays
C
C SUBROUTINE BGFUNC(INFILE,IKEY,LINE,LENLINE,ERR,EOF)
C
*** FORMAL PARAMETER DECLARATIONS ***
%INCLUDE INCPAR
%INCLUDE FLPMTR
   INTEGER INFILE,IKEY,LENLINE
   LOGICAL EOF,ERR
   CHARACTER LINE*LLINE
C
*** ABSTRACT ***
C* PURPOSE process a function line from infile to get battle group
C* HISTORY
C MEMutchler 16-may-83 AUTHOR
C* TYPE process force level output control file
C* FORMAL PARAMETERS
Cin infile read from this file
Cio ikey,line,rlenline,err,eof results of readfl
C* COMMON BLOCKS
Cio readc holds line counter
C* CALLER fibldr
C* METHOD
C get label off line, split rest of line into type names
C assume first type found has top priority, etc.
C process type names by storing index intotype totals as
C fdefine(priority of choice, function it will achieve)
C , process next line until it is not a
C continuation line
C**
SUBROUTINE BGGET(ERR)

*** FORMAL PARAMETER DECLARATIONS ***

*** ABSTRACT ***

PURPOSE
Compose the battle groups from the available ship pool.

AUDIT HISTORY
MEMutchler 29-JUN-83 AUTHOR

TYPE battle group counting

FORMAL PARAMETERS

Cout err true if major error was found

Cin groupbg descrip of battle group compositions, tgts

Cin typebg ships available by type

Cin funcbg functional ship family definitions

CALLER bgbrpt

METHOD

given data from forcelevel table and battlegroup output control, make up battle group force, one period at a time, filling first priority groups first, with first choice type to do a function

LOCAL VARIABLES

emptypt buffer of per-period ships avail for a type

grprior group priority buffer
SUBROUTINE BGINIT

*** FORMAL PARAMETER DECLARATIONS ***

C** PURPOSE initialize BGRP internal buffers.
enddate

C**AUDIT HISTORY
MEMutchler 31-may-83 AUTHOR

C**TYPE initialize arrays to zero

C**FORMAL PARAMETERS

C**COMMON BLOCKS

Cout  typebg  ship type avail info
Cout  groupbg  battle groups defns
Cout  funcbg  functional families

C**METHOD

C  Loops setting array locations to zero.
C**
SUBROUTINE BGMKUP(INFILE, IKEY, LINE, LENLINE, ERR, EOF)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER INFILE, IKEY, LENLINE
LOGICAL EOF, ERR
CHARACTER LINE*LLINE

*** ABSTRACT ***

PURPOSE process a makeup line from infile to get forcelevl

HISTORY
MEMutchler 27-JUN-83 AUTHOR

TYPE process battlegroup force level output control file

FORMAL PARAMETERS
Cin infile read from this file
Cio ikey, line, lenline, err, eof results of readfl

COMMON BLOCKS
Cio readc holds line counter
Cin ficlass holds class list

CALLER flbldr

METHOD
get label off line, split rest of line into function names
number of function members needed for this battlegroup
When done with all
functions from line, process next line until it is not a
continuation line
C                      BGMRPT------------------------------------------
$CONTROL check=3, segment=BGRP
SUBROUTINE BGMRPT
C*                *** FORMAL PARAMETER DECLARATIONS ***
C*                *** ABSTRACT ***
C* PURPOSE writes battle group report to ioutfl
C* AUDIT HISTORY
C* MEMutchler       28-may-83 AUTHOR
C* TYPE fore level io routine
C* FORMAL PARAMETERS none
C* COMMON BLOCKS
C* io flpage holds pagebuf
C* cin flhead holds header text
C* METHOD
C* write text to buf keeping track of lines used
C*
SUBROUTINE BGPROC(ERROR)

LOGICAL ERROR

*** ABSTRACT ***

PURPOSE parse output control file and create an alphabetized list of each ship class mentioned on a TYPE line.

MEMutchler 27-JUN-83 AUTHOR

TYPE find which classes are to be examined

FORMAL PARAMETERS non

COMMON BLOCKS

incoc i/o file assignments

incpar global parameters

CALLER fireport

METHOD Starting at top of FILE OCNTRL look at a line

IF line begins with "TYPE" THEN extract class names from line and following lines beginning with "+", add the names to a list of names IF not already there. ELSE GO TO next line until end of file. Alphabetize the list of names.

LOCAL VARIABLES

line one line from ocfile

lenline deblanked length of line

eof true IF end of file ocfile has been read

lenkey deblanked length of key

11-46
SUBROUTINE BGSETV(BEGDATE, ENDDATE, PROGRAM, VALUE)

*** FORMAL PARAMETER DECLARATIONS ***

%INCLUDE FLPMTR
%INCLUDE FLPERD

INTEGER*4 BEGDATE, ENDDATE
LOGICAL DEARLY
INTEGER VALUE, PROGRAM (1, MXPERD)

*** ABSTRACT ***

PURPOSE increment program total for period from begdate to enddate

AUTHOR MEMutchler 31-may-83

FORMAL PARAMETERS

begdate begining date
enddate ending date
program holds current program totals
fperd time horizon this run

METHOD
get first period after begindate, get last period before enddate. Increment program between these two periods

11-47
SUBROUTINE BGTYPE(INFILE, IKEY, LINE, LENLINE, ERR, EOF)

* FORMAL PARAMETER DECLARATIONS ***

INTEGER INFILE, IKEY, LENLINE
LOGICAL EOF, ERR
CHARACTER LINE*LLINE

*** ABSTRACT ***

PURPOSE process a type line from infile to get forcelevel

MEMutchler 16-may-83 AUTHOR

PARAMETERS
Cin infile read from this file
Cio ikey, line, lenline, err, eof results of readfl

BLOCKS
Cio readc holds line counter
Cin flclass holds class list

CALLER flbldr

METHOD
get label off line, split rest of line into class names
process class names by adding force levels of eachclass
mentioned to the appropriate prgbuf. When done with all
classes from line, process next line until it is not a
continuation line.
SUBROUTINE BGWRIT

*** FORMAL PARAMETER DECLARATIONS ***

*** ABSTRACT ***

PURPOSE write pagebuf to ioutfl

AUDIT HISTORY

MEMutchler 28-may-83 AUTHOR

TYPE battle group level io routine

FORMAL PARAMETERS none

COMMON BLOCKS

Cio flage holds pagebuf
Cin flhead holds output text specs

METHOD

write text to buf keeping track of lines used

11-49
SUBROUTINE BGWTOP

*** FORMAL PARAMETER DECLARATIONS ***

*** ABSTRACT ***

PURPOSE writes title line and period header to pagebuf

AUDIT HISTORY

MEMutchler 28-may-83 AUTHOR

TYPE fore level io routine

FORMAL PARAMETERS none

COMMON BLOCKS

pio pagebuf

pio fihead header text

METHOD

write text to buf keeping track of lines used

LOCAL VARIABLES

none

**
SUBROUTINE BTLGRP(INFILE, IKEY, LINE, LENLINE, ERR, EOF)

C* ** FORMAL PARAMETER DECLARATIONS ***

%INCLUDE INCPAR
%INCLUDE FLPMTR

INTEGER INFILE, IKEY, LENLINE
LOGICAL EOF, ERR
CHARACTER LINE*LLINE

C* *** ABSTRACT ***

C* PURPOSE process a bgroup line from infile to get forcelevl

C* HISTORY

C MEMutchler 27-JUN-83 AUTHOR

C* TYPE process battlegroup force level output control file

C* FORMAL PARAMETERS

Cin infile read from this file
Cio ikey, line, lenline, err, eof results of readfl

C* COMMON BLOCKS

Cio reac holds line counter
Cin flclass holds class list

C* CALLER flbldr

C* METHOD

C get groupname off line, split rest of line into group label(to

C be printed on actual output), priority with which this group must

C be filled, force level to achieve, date this info begins at,

C date this info ends with, process next line untill it is not a

C continuation line

C**
SUBROUTINE FFLTBL(ERR)

*** FORMAL PARAMETER DECLARATIONS ***

LOGICAL ERR

*** ABSTRACT ***

PURPOSE fill in the force level table (fltabl) buffer
giving ships available each period

MEMutchler/MCarey 23-may-83 AUTHOR

FORMAL PARAMETERS

none

COMMON BLOCKS

READC number of last input file line, for lwarn
FCONS file and field names for relation access
FLDECM
FLRJOB
SCLIFE
SCENARIO
FLTABLS

CALLER FLREPORT

METHOD

for each of the alphabetized ship classes
get the standard service life of any in the class
for each hull # in the class
search the construction relations for first construction
search for first decommissionings
search for all other construction and decommissionings
search job relation for all jobs
keep job info IF it adds to service life or in major job list

LOCAL VARIABLES

endlIF true IF no decommission date found for
a recommissioning
maxcons max. # of constructions done to one ship
maxjob max # of repair jobs done to one ship
joblife years added to life due to this job
clife years added to life due to this construction
condat date of construction delivery
decdat date of decommissioning
jobdat date of beginning of repair job
jedat date of ending of repair job
InvBuf inventory buffer
prgbuf program buffer
majjob true IF job is one of lston
conpro construction is program not inventory
jobpro repair job is program not inventory
SUBROUTINE FLADPG(TOTARRY, LLAB, RLAB, TOTALIN)

C*                                   *** FORMAL PARAMETER DECLARATIONS ***
%INCLUDE FLPMTR
%INCLUDE FLPERD
%INCLUDE FLHEAD

CHARACTER LLAB*LENLLB, RLAB*LENRLB
INTEGER TOTARRY(MXPERD)
LOGICAL TOTALIN

C*                                   *** ABSTRACT ***
C* PURPOSE write total line to pagebuf
C* AUDIT HISTORY
C* TYPE force level io routine
C* FORMAL PARAMETERS none
C* COMMON BLOCKS
Cio  flpage  holds pagebuf
Cin  flhead  holds output text specs
Cin  fltot1  holds line to be output
Cin  flperd  holds period info
C* METHOD
C write text to buffer keeping track of lines used
C##

11-53
SUBROUTINE FLBRPT (ERR)

LOGICAL ERR

** PURPOSE use output control file and fltabl to create
force level report file

** ABSTRACT

use output control file and fltabl to create
force level report file

** HISTORY

MEMutchler 23-may-83 AUTHOR

** TYPE

force level report utility

** PARAMETERS

err error flag

** COMMON BLOCKS

incpar global parameters
charcon output control file keywords
readc holds line number last read
fltotl holds totaling arrays
fltabl holds force level tables for each program
fileport

** METHOD

process output control file by line creating the structure
for a force level report use values in fltabl for data.
Determine which type of line just read from output
control file and process accordingly
START read everyline between a START and STOP line
only title and program labels are acknowledged
prior to the initial start
+
this is a continuation of the text of last line, only
for JOB or TYPE lines
TITLE center text on the top of each report page
must be read before initial start
PROGLB text is the program label, program start date
at least one must be read before start
BTOT begin a new totaling array of beginname=text
add a new totaling array
ETOT end the last totaling array begun, text=name, left
label, right label, be sure name=beginname
write report lines to report file
delete last totaling array
EITOT end the last totaling array begun, text=name, left
label, right label, be sure name=beginname
don't do a page feed after, delete totaling array
TYPE text=typename, classes to make up type; defines a type
get values from force level tables for program types
and add class levels together
JOB text=typename, jobname, job as known to RELATE, classes
in type; total all job done to each class for programs

** LOCAL VARIABLES

protot program totaling arrays

CONTROL
check=3, segment=FLRP

11-54
LOGICAL FUNCTION FLCHK1(SCEN1, SCEN2, CLAS1, CLAS2, INT1, INT2)

*** FORMAL PARAMETER DECLARATIONS ***

CHARACTER*12 SCEN1, SCEN2
CHARACTER*10 CLAS1, CLAS2
INTEGER INT1, INT2

*** ABSTRACT ***

C&PURPOSE make sure all *1 = *2

C&AUDIT HISTORY

C TYPE force level utility

C&FORMAL PARAMETERS

Cin scen scenario name

Cin clas class name

Cin int number

Cout flchk1 true if all *1 = *2

C check each pair

C##
LOGICAL FUNCTION FLCHK2(SCEN1, SCEN2, CLAS1, CLAS2,
+       INTA1, INTA2, INTB1, INTB2)

*** FORMAL PARAMETER DECLARATIONS ***

CHARACTER*12 SCEN1, SCEN2
CHARACTER*10 CLAS1, CLAS2
INTEGER INTA1, INTA2, INTB1, INTB2

*** ABSTRACT ***

PURPOSE make sure all *1=*2; used for end-of-data detection
when reading along a RELATE index

HISTORY
MEMutchler 31 May 83 AUTHOR

TYPE force level utility

FORMAL PARAMETERS

scenario name
class name
number

flchk2 true if all *1=*2

METHOD
check each pair
LOGICAL FUNCTION FLCHK3(SCEN1, SCEN2, CLAS1, CLAS2, + INTA1, INTA2, JOBID1, JOBID2)

*** FORMAL PARAMETER DECLARATIONS ***
CHARACTER*12 SCEN1, SCEN2
CHARACTER*10 CLAS1, CLAS2
CHARACTER*8 JOBID1, JOBID2
INTEGER INTA1, INTA2

*** ABSTRACT ***
PURPOSE make sure all *1==2: for RELATE end-of-data-group detection.
HISTORY
MEMutchler 31 may 83 AUTHOR
TYPE force level utility
FORMAL PARAMETERS
CIN scen scenario name
CIN clas class name
CIN int number
COUT flchk2 true if all *1 = *2
METHOD
C check each pair
C##
C  FLCLAS  ********************************************
$CONTROL  check=3,segment=FLBG
   SUBROUTINE FLCLAS( LINE, LENLINE, CONTLN, ERR)
C*  *** FORMAL PARAMETER DECLARATIONS ***
   INTEGER LENLINE
   LOGICAL ERR
   CHARACTER LINE*(LENLINE)
   LOGICAL CONTLN
C*  *** ABSTRACT ***
C*PURPOSE parse the text part of a type line from the format
C   file, adding class names to the list of class names
C   IF not yet present
C*AUDIT HISTORY
C   MEMutchler  17-may-83 AUTHOR
C*TYPE parse output control file
C*FORMAL PARAMETERS
Cin   line     the text following the keyword in the output control
C    file
Cin   lenline length of the line
Cin   contln  true IF this was a continue line, else false
C*COMMON BLOCKS
Cio   classes   holds class list
Cin   incpar   global parameter list
Cin   charcon constant strings
Cin   readc   holds line number just read
C*CALLER parsoc
C*METHOD remove first phrase if not contln. Parse line, one
C   phrase at a time, adding to list if possible.
C*LOCAL VARIABLES
C   string unparsed part of line
C   lstring deblanked length of string
C   clname one class name
C**
SUBROUTINE FLCLOS

*** FORMAL PARAMETER DECLARATIONS ***

*** ABSTRACT ***

PURPOSE close files and relations only needed by firept

AUDIT HISTORY

MEMutchler 19-may-83 AUTH0

TYPE clean up

FORMAL PARAMETERS

none

COMMON BLOCKS

flioc fortran io units
fldcm relation names
flrjob
shlife

CALLER firept

METHOD

Calls to filcis and rvcls; also call lpsend to start
"printing."

11-59
SUBROUTINE FLDECR(BEGDATE, ENDDATE, PROGRAM)

INTEGER*4 BEGDATE, ENDDATE
LOGICAL DEARLY
INTEGER PROGRAM (1, MXPERD)

*** ABSTRACT ***
PURPOSE decrement program total for period from begdate to enddate

MEMutchler 31-may-83 AUTHOR

TYPE decrement due to dates

PARAMETERS

begdate repair job beginning date
enddate repair job ending date
program holds current program totals

COMMON BLOCKS

flperm first day of each period

METHOD

get first period after begindate, get last period before enddate. Decrement program between these two periods

11-60
SUBROUTINE flglif(stanlf,stunit,ncom,condat,decdat,
   addlif,addunt,njob,joblif,jobunt,
   jedate,err)

*** FORMAL PARAMETER DECLARATIONS ***
integer ncom,njob,stanlf,addlif(ncom),joblif(njob)
integer*4 condat(ncom),decdat(ncom),jedate(njob)
character*6 stunit,addunt(ncom),jobunt(njob)
logical err

*** ABSTRACT ***
get last decommissioning date for ship

MEMutchler 31-may-83 AUTHOR
MSCarey 28-apr-84 Major logic change to use
time units in calculations

C\$FORMAL PARAMETERS
Cin stanlf standard life of ship
Cin stunit in these units
Cin ncom number of commissionings ship has
Cin condat commissioning dates
Cin decdat deactivation dates: output in decat(ncom) ----
Cin addlif amount of life added each commissioning
Cin addunt in these time units
Cin njob number of repair jobs ship has had
Cin joblif amount of life added by each repair job
Cin jobunt in these time units
Cin jedate end date of each repair job
Cout err true if any jedate>decdat(ncom)

C\$COMMON BLOCKS
C none

C\$CALLER ffltbl

C\$METHOD
Obtain the number of days used during each commissioning.
Obtain the retirement date which would have occurred if
there was only one commissioning, and if there was no
life added by any job, using the standard lifetime/units.
Find the number of days represented by this lifetime.
Find the number of days between the first activation and
the last activation, and compare this to the number
of days used during active periods to get the amount of
time spent in mothballs. Get a final deactivation date.
Then cycle through the repair
jobs and push out this date by the amount of time added
by each job. Then cycle through the commissioning jobs
and do the same. The result is a computed deactivation
date based on (possibly) different time units used to
to specify the various life-length increments.

C\$LOCAL VARIABLES
comusd days of life used up by commissioned time
mthadd days of life 'added' by time spent in mothballs
rawday length of standard life in days
rawdat raw retirement date
deact working retirement date
SUBROUTINE FLINCL( STR, LSTR, LSIZE, MLSIZE, LIST, ELSIZE, IINDEX, ERR)

CHARACTER STR*(ELSIZE), LIST*(ELSIZE)(MLSIZE)
LOGICAL ERR
INTEGER IINDEX, LSTR, LSIZE, MLSIZE, ELSIZE

*** FORMAL PARAMETER DECLARATIONS ***

*** ABSTRACT ***

PURPOSE include string in a list

MEMutchler 8 JUN 83 AUTHOR

FORCE level report generator utility

IN STR NAME TO GO ON LIST
IN LSTR NUMBER OF CHARS IN STR
CIO LSIZE NAME-LIST SIZE. WILL BE INCREMENTED IF STR NOT
C ALREADY ON THE LIST.
CIO MLSIZE MAX ALLOWED VALUE OF LSIZE
CIO LIST NAME-LIST
CIO ELSIZE MAX CHARS IN EACH ELEMENT OF 'LIST'
COUT IINDEX INDEX OF STR ON LIST

COMMON BLOCKS
Cin readc # of line last read from input file

METHOD
C CONSTRUCTS LIST 'LIST' OF NAMES

C
SUBROUTINE FLINCR(BEGDATE, ENDOATE, PROGRAM)

INTEGER*4 BEGDATE, ENDOATE
LOGICAL DEARLY
INTEGER PROGRAM (1, MXPERD)

*** ABSTRACT ***

CPURPOSE increment program total for period from begdate to enddate
CAUDIT HISTORY
CMEMutchler 31-may-83 AUTHOR
CSTYLE increment due to dates
CFORMAL PARAMETERS
Cin begdate begining date
Cin enddate ending date
Cio program holds current program totals
CCOMMON BLOCKS
Cin flperd first day of each period
CMETHOD
C get first period after begindate, get last period before enddate. Increment program between these two periods
C#
SUBROUTINE FLJOB(INFILE,KEY,LINE,LINLEN,ERR,EOF)

%INCLUDE INCPAR
%INCLUDE FLPMTR
    INTEGER INFILE,KEY,LENLINE
    LOGICAL EOF,ERR
    CHARACTER LINE,LINLEN

*** ABSTRACT ***

**PURPOSE**
get the total effect on classes seen on one job
C line of the output control file as performed by periods

**AUDIT HISTORY**
MEMutchler 31-may-83 AUTHOR

**TYPE**
fill total array by processing ocfile

**FORMAL PARAMETERS**
Cin  infile read from this file
Cio  key,line,linlen,err,eof  results f readfl

**COMMON BLOCKS**
Cio  readc  holds line counter

**CALLER** flbldr

**METHOD**
get label and jobname off line split rest of line into classes
C process the job by totalin for each period how many times
C that job was performed on any of the classes mentioned
C When done processing all of the
C classes from line, process next line until it is not a
C continuation line

""
C SUBROUTINE FLNPER(ERROR)

*** FORMAL PARAMETER DECLARATIONS ***

LOGICAL ERROR

*** ABSTRACT ***

C* PURPOSE determine nperiod and fill in datper(1..nperiod)
C* and makes up period header
C* MEMutchler      31-May-83  AUTHOR
C* TYPE report utility
C*COMMON BLOCKS
Cin  value    menu parameter values
Cin  pvedcl  menu parameter declarations
Cin  pveqiv  menu parameter equivalences
Cin  flperd  period info
C*CALLER    FLINIT
C*METHOD
C* Calls to a subsidiary routines, depending on period length
C* Note--implemented before standard TOODATE ALIAS date utilities
C**
SUBROUTINE FLNXTP(ANYDAT, PERDAT)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER*4 ANYDAT, PERDAT
LOGICAL DEARLY

*** ABSTRACT ***

CPURPOSE get first datper following anydat
CAUDIT MEMutchler 2-june-83 AUTHOR
CTYPE force level date utility
CFORMAL PARAMETERS
Cin anydat relate clarified *4 date
Cou perdat first datein datper following anydat
CCOMMON BLOCKS
Cin flperd holds period info
CCALLER flglif
CMETHOD
C loop through datper array untill a date .ge. anydat is found
CLOCAL VARIABLES
C iperd period index
C##
SUBROUTINE flpars(line, lenlin, list, lenlst, mxlchr, mxnum, num, 
                   tomany, tolong)
   character*(lenlin) line
   character*(mxlchr) list(mxnum)
   integer lenlin, mxnum, num, lenlst(mxnum)
   logical tomany, tolong

*** ABSTRACT ***

CSPURPOSE  Parses an input string into substrings delimited by commas.

CSASUIT HISTORY

C MSCarey  03-jun-83 AUTOR

CFORMAL PARAMETERS

   Cin  line  string to be parsed
   Cin  lenlin  length of line in chars
   Cout list  list of output substrings
   Cin  mxlchr  max length of any substring
   Cout lenlst  length of each substring
   Cin  mxnum  maximum number of substrings returnable
   Cout num  number of substrings found
   Cout tomany  true if more than mxnum substrings found
   Cout tolong  true if a substring longer than mxlchr found

CCOMMON BLOCKS

C none

CALLER various

METHOD

   Look for commas and extract the intermediate text.
SUBROUTINE FLPDAY( ERROR)

*** FORMAL PARAMETER DECLARATIONS ***

LOGICAL ERROR

*** ABSTRACT ***

PURPOSE determine nperiod and fill in datper(1..nperiod)
and makes up period header when period length=day

AUDIT HISTORY
MEMutchler 31-may-83 AUTHOR

TYPE report utility

COMMON BLOCKS
pvalue menu parameter values
pvdecl menu parameter declarations
pveqiv menu parameter equivalences
flipf over

CALLER FLINIT

METHOD
Low-level date utility calls and straight string concats.
C             FLPMTH

*CONTROL check=3,segment=FLBG
SUBROUTINE FLPMTH(ERROR)
C*                *** FORMAL PARAMETER DECLARATIONS ***
C*         LOGICAL ERROR
C*
C*                *** ABSTRACT ***
C* PURPOSE determine nperiod and fill in datper(1..nperiod)
C* and makes up period header when period length=year
C*AUDIT HISTORY
C     MEMutchler       31-may-83  AUTHOR
C*TYPE   report utility
C*COMMON BLOCKS
Cin    pvalue       menu parameter values
Cin    pvdecl       menu parameter declarations
Cin    pveqiv       menu parameter equivalences
Cout   flperd       period info
C*CALLER  FLINIT
C*METH0D
C Low-level date utility calls and straight string concats.
C**
** FORMAL PARAMETER DECLARATIONS **

LOGICAL ERROR

*** ABSTRACT ***

PURPOSE determine nperiod and fill in datper(1..nperiod)

and makes up period header when period length=year

MEMutchler 31-may-83 AUTHOR

TYPE report utility

COMMON BLOCKS

pvalue menu parameter values

pvdecl menu parameter declarations

pvequiv menu parameter equivalences

flperd period info

CALLER FLINIT

METHOD

Low-level date utility calls and straight string concats.

##
C FLPRGN

*CONTROL check=3,segment=FLBG
INTEGER FUNCTION FLPRGN(APPROP,AWARD,DELIV)

C*** FORMAL PARAMETER DECLARATIONS ***
INTEGER*4 APPROP,AWARD,DELIV

C*** ABSTRACT ***
C$PURPOSE get program number this will fall in according
to appropriate date
C$AUDIT HISTORY
MEMutchler 31-may-83 AUTHOR
CTYPE force level utility
C$FORMAL PARAMETERS
Cin  approp  appropriation date
Cin  award  award date
Cin  deliv  delivery date
C$COMMON BLOCKS
Cin  pvalue  menu parameter values
Cin  pvdecl  menu parameter declarations
Cin  pvequiv  menu parameter equivalences
Cin  fltabs  program beginning dates
C$CALLER  ffltbl
C$METHOD  determine which date to use to determine program
        and use it with beginning program dates to find program
        the date falls in.
C**

11-71
C FLPRNT

*CONTROL check=3,segment=FLB6
SUBROUTINE FLPRNT

C** FORMAL PARAMETER DECLARATIONS **
C** ABSTRACT **

C** PURPOSE print report to daisy or lp
C**AUDIT HISTORY
C   MEMutchler  16-may-83 AUTHOR
C**FORMAL PARAMETERS
C none
C**COMMON BLOCKS
Cin  flioc  flrp unit numbers
C**METHOD
C The sequential file contains the actual force level report to
C be displayed using SUBROUTINE FLPRNT. It contains the actual
C lines of text,titles,non-printing comments, and page feed markers.
C It must be made permanent IF it is to be saved, and may be edited
C IF desired. Print all printable lines literally, and use page ejec
C**LOCAL VARIABLES
C line   a line of text to be printed
C**
SUBROUTINE FLPROC(ERROR)

*** FORMAL PARAMETER DECLARATIONS ***

LOGICAL ERROR

*** ABSTRACT ***

PURPOSE parse output control file and create an alphabetized
list of each ship class mentioned on a TYPE line.

HISTORY
MEMutchler 16-may-83 AUTHOR

TYPE find which classes are to be examined

FORMAL PARAMETERS non

COMMON BLOCKS

CALLER flreport

METHOD Starting at top of FILE OCNTRL look at a line
IF line begins with "TYPE" THEN extract class names from line and
following lines beginning with "+", add the names to a
list of names IF not already there.  ELSE GO TO next line until
end of file. Alphabetize the list of names.

LOCAL VARIABLES

line one line from ocfile
lenline deblanked length of line
eof true IF end of file ocfile has been read
lenkey deblanked length of key

C
SUBROUTINE FLPWEK( ERROR)
*C
*** FORMAL PARAMETER DECLARATIONS ***
LOGICAL ERROR
*** ABSTRACT ***
PURPOSE determine nperiod and fill in datper(1..nperiod)
and makes up period header when period length=week
AUDIT HISTORY
MEMutchler 31-may-83 AUTHOR
TYPE report utility
COMMON BLOCKS
Cin pvalue menu parameter values
Cin pvdecl menu parameter declarations
Cin pveqiv menu parameter equivalences
Cou flperd period info
CALLER FLINIT
METHOD
Low-level date utility calls and straight string concats.
**
SUBROUTINE FLPYER( ERROR)

*** FORMAL PARAMETER DECLARATIONS ***

LOGICAL ERROR

*** ABSTRACT ***

PURPOSE determine nperiod and fill in datper(1..nperiod)

and makes up period header when period length=year

AUDIT HISTORY

MEMutchler 31-may-83 AUTHOR

TYPE report utility

COMMON BLOCKS

 pvdecl menu parameter declarations
 pveqiv menu parameter equivalences
 flperd period info

CALLER FLINIT

METHOD

Low-level date utility calls and straight string concats.

##
SUBROUTINE FLRDCN (IUNIT, LINE, EOF)

C* *** FORMAL PARAMETER DECLARATIONS ***

LOGICAL EOF
INTEGER IUNIT
CHARACTER LINE*LLINE

C* *** ABSTRACT ***

C* PURPOSE read from file IN and keep track of lines read

C* AUDIT HISTORY

C MEMutchler 17 JAN 83 AUTHOR
C MEMutchler 8 FEB 83 TESTER (program treadc)

C* TYPE mnugen utility

C* FORMAL PARAMETERS

Cin iunit file number from which to read
Cout line input line read
Cout eof true iff eof read from iunit

C* COMMON BLOCKS

Cin incpar global parameter statement
Cin reads holds iline

C* METHOD. An unformatted read is done from unit =
C iunit. EOF = false unless an end of file is read
C in which case EOF = true. If command file building
C is in use, LINE is echoed to unit = icomfile.
C Icount is incremented.

C* LOCAL VARIABLES

C recch '%' recognition character for comment card
C
C

11-76
SUBROUTINE FLRLDN (IUNIT, LINE, EOF)

*** FORMAL PARAMETER DECLARATIONS ***

%INCLUDE INCPAR
%INCLUDE IOC
INTEGER IUNIT
LOGICAL EOF
CHARACTER LINE*LLINE, BUFFER*LLINE

*** ABSTRACT ***

C* PURPOSE read a line from IUNIT
C* AUDIT HISTORY
C* TYPE firept utility
C* FORMAL PARAMETERS
Cin iunit unit number from which to read
Cout line line that was read
Cout eof true iff eof was read
C* COMMON BLOCKS
Cin incpar global parameter statement
Cin comcfl holds command file info.
C* METHOD An unformatted read is done from unit =
C  iunit. EOF = false unless an end of file is read
C  in which case EOF = true. If command file building
C  is in use, LINE is echoed to unit = icomfile.
C* LOCAL VARIABLES none
C**
C FLEP

$CONTROL check=3, segment=FLRP

PROGRAM FLEP

C* *** FORMAL PARAMETER DECLARATIONS ***

C* *** ABSTRACT ***

C* PURPOSE main for force level report generator

C* AUDIT HISTORY

C MEMutchler 16-MAY-83 AUTHOR

C* TYPE main program

C* COMMON BLOCKS none

C* CALLED BY menu system choice menu

C* METHOD

C Initialize and open necessary relations and files.

C Parse output control file creating an alphabetized list of

C all ship classes found on type lines. Fill in force level table,

C one row for each ship class found, one column for each time period,

C as number of ships built of that class in that time period - number

C of ships of that class and period out for major deactivating jobs or

C in temporary retirement. Process output control file along with

C the force level table to build force level report file.

C Print force level report file.

C* LOCAL VARIABLES

C error error flag

C**
C  FLPNG

$CONTROL check=3,segment=FLBG
  INTEGER FUNCTION FLPNG(DATE,NCONS,CONDATE)
C
  *** FORMAL PARAMETER DECLARATIONS ***
  INTEGER NCONS
  INTEGER*4 DATE, CONDATE(NCONS)
  LOGICAL DEARLY

C  *** ABSTRACT ***
C$PURPOSE get program number this will fall in according
C to date
C$AUDIT HISTORY
C  MEMutchler  31-may-83 AUTHOR
C$TYPE force level utility
C$FORMAL PARAMETERS
Cn  date  repair beginning date
C$COMMON BLOCKS
Cn  pvalue  menu parameter values
Cn  pvdecl  menu parameter declarations
Cn  pvequiv  menu parameter equivalences
Cn  fitabis  program beginning dates
C$CALLER ffltbl
C$METHOD
C  use date with beginning program dates to find program
C  the date falls in.
C$
SUBROUTINE FLTYPE(INFILE,IKEY,LINE,LENLINE,ERR,EOF)

FORMAL PARAMETER DECLARATIONS

INTEGER INFILE,IKEY,LENLINE
LOGICAL EOF,ERR
CHARACTER LINE*LLINE

*** ABSTRACT ***

PURPOSE process a type line from infile to get force level

AUTHOR MEMutchler 16-may-83

PARAMETERS

Cin infile read from this file
Cio ikey,line,alenline,errnof results of readfl

COMMON BLOCKS

Cio readc holds line counter
Cin flclass holds class list

CALLER flbldr

METHOD

get label off line, split rest of line into class names
process class names by adding force levels of each class
mentioned to the appropriate prgbuf. When done with all
classes from line, process next line until it is not a
continuation line

C**
SUBROUTINE FLWRIT

C*...

CSPURPOSE write pagebuf to loutfl
C*AUDIT HISTORY
C  MEMutchler  28-may-83  AUTHOR
C*TYPE force level io routine
C*FORMAL PARAMETERS none
C*COMMON BLOCKS
Cio  flage holds pagebuf
Cin  flhead holds output text specs
C*METHOD
C  write text to unit keeping track of lines used
C**
C FLWTOP

*CONTROL check=3, segment=FLRP

SUBROUTINE FLWTOP

C* *** FORMAL PARAMETER DECLARATIONS ***
C* *** ABSTRACT ***
C* PURPOSE writes title line and period header to pagebuf
C* AUDIT HISTORY
C* MEMutchler  28-may-83  AUTHOR
C* TYPE fore level io routine
C* FORMAL PARAMETERS none
C* COMMON BLOCKS
C1o flpage holds pagebuf
Cin flhead holds header text
C* METHOD
C write text to buf keeping track of lines used
C**
SUBROUTINE FNDPRD ( DATE, PERIOD )

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER PERIOD
INTEGER*4 DATE
LOGICAL DEARLY

*** ABSTRACT ***

C PURPOSE find number of period to which date belongs
C like the gpern utility
C\*AUDIT HISTORY
C MEMutchler 31-may-83 AUTHOR
C\*FORMAL PARAMETERS
Cin date date to look for
Cout period number of period to which date belongs
C\*COMMON BLOCKS
CIN flperd first date each period
C\*METHOD search through datper array until datper gt date
C period = iper
C**
SUBROUTINE GETJOB (CLASS, JOBTYP, TOTAL)

*** FORMAL PARAMETER DECLARATIONS ***

%INCLUDE FLPMTK
%INCLUDE FLPERD

CHARACTER CLASS*10
CHARACTER JOBTYP*6
INTEGER TOTAL(MXPERD)

*** ABSTRACT ***

PURPOSE find all ships in this class having a repair of this type and adds them all up by period

MEMutchler 16 jn 83 AUTHOR

TYPE get info from relate for force level report generator

FORMAL PARAMETERS

class class name to find repairs for

TOTAL number of ships repaired in each period

COMMON BLOCKS

flrjob relate repair relation info

CALLER fljob

METHOD

get repair job schedule record for latest data date

11-84
SUBROUTINE GETLIF(CLASS, LIFIND, ERR, LIFUNT)

CHARACTER LIFUNT*6, CLASS*10
LOGICAL ERR
INTEGER LIFIND

C* PURPOSE find standard life of all ships in the class
C* AUDIT HISTORY
C MEMutchler 31 MY 83 AUTHOR
C TYPE get info from relate for force level report generator
C FORMAL PARAMETERS
Cin class class name to find life for
Cot lifind standard life IN Days
Cout lifunt time units life duration is in
C COMMON BLOCKS
Cin shlife lifetimes for all classes
C METHOD
C get standard lifetime for latest data date
C**
SUBROUTINE GMAKUP(INPTR,NEXTPTR,FINDX,NUMNEED)
INTEGER INPTR,NEXTPTR,FINDX,NUMNEED

*** FORMAL PARAMETER DECLARATIONS ***

*** ABSTRACT ***

C* PURPOSE get the record from bgmakup at inptr

C* AUDIT HISTORY
C MEMutchler 29-JUN-83 AUTHOR

C* TYPE battlegroup io

C* FORMAL PARAMETERS
C inptr get this record
C nextptr ptr to next record needed for function makeup
C =0 if no more there for function
C findx index into fdefine for this function needed by makeup
C numneed number of this function needed by group makeup

C* COMMON BLOCKS
C IN gpmkup holds records to read

C* METHOD
C Transfer data from the bgmakeup array to the arguments.

C*
SUBROUTINE PAR2LN(STRN, LLIN, HAF1, LHAF1, HAF2, LHAF2, ERR)

C** FORMAL PARAMETER DECLARATIONS ***

INTEGER LLIN, LHAF1, LHAF2
LOGICAL ERR
CHARACTER STRN*(LLIN), HAF1*(LHAF1), HAF2*(LHAF2)

C** ABSTRACT ***

C PURPOSE split line into two parts separated by a comma

C AUDIT HISTORY
MEMutchler 27-may-83 AUTHOR

C TYPE character utility

C FORMAL PARAMETERS
Cin line string to be split
Cin llin length of string
Cou haf1 put first part here
Cou lhaf1 length of haf1
Cou haf2 put second part here
Cou lhaf2 length of haf2

C COMMON BLOCKS
Cin charcon character constants

C METHOD
C find a comma and split line by that

C LOCAL VARIABLES
C i index of comma
SUBROUTINE PAR3LN(LINE, LLIN, PRT1, LPRT1, PRT2, LPRT2, PRT3, LPRT3, ERR)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER LLIN, LPRT1, LPRT2, LPRT3
LOGICAL ERR
CHARACTER LINE*(LLIN), PRT1*(LPRT1), PRT2*(LPRT2), PRT3*(LPRT3)

*** ABSTRACT ***

PURPOSE split line into three parts separated by a comma

MEMutchler 27-may-83 AUTHOR

TYPE character utility

FORMAL PARAMETERS

line string to be split
LLIN length of string
PRT1 put first part here
LPRT1 length of PRT1
PRT2 put second part here
LPRT2 length of PRT2
PRT3 put third part here
LPRT3 length of PRT3

COMMON BLOCKS

charcon character constants

METHOD
find a comma and split line by that

LOCAL VARIABLES

I index of comma

//
SUBROUTINE REAOBG (INFILE, IKEY, LINE, LENLINE, ERR, EOF)

C* *** FORMAL PARAMETER DECLARATIONS ***

%INCLUDE INCPAR
%INCLUDE FLPMTR
INTEGER INFILE, IKEY, LENLINE
LOGICAL EOF, ERR
CHARACTER LINE*LENLINE

C* *** ABSTRACT ***
C#PURPOSE reads next non-comment line from file infile, and
C parses line for firstword <= ikey characters and
C the rest of the line. Returns eof=true iff end of
C file has been read. If key=stop is read all lines
C are ignored untill key=start is read
C#AUDIT HISTORY
C MEMutchler 27-JUN-83 AUTHO
C#TYPE read from battlegroup force level report input file
C#FORMAL PARAMETERS
C INFILE file from which to read
C IKEY first word of line read
C LEKEY length of key
C LINE rest of line read
C LENLINE length of line read
C EOF end of file flag
C#COMMON BLOCKS
C CHARCON global character constants
C PARSOC
C#METHOD
C read a line. If eof then return. If comment line, read again.
C set key to first non-blank word of line, line to rest and get their
C lengths
C#
*CONTROL check=3,segment=FLRP

SUBROUTINE READFL (INFILE, IKEY, LINE, LENLINE, ERR, EOF)

*** FORMAL PARAMETER DECLARATIONS ***

%INCLUDE INCPAR
%INCLUDE FLPMTR

INTEGER INFILE, IKEY, LENLINE
LOGICAL EOF, ERR
CHARACTER LINE*LLINE

*** ABSTRACT ***

C** PURPOSE reads next non-comment line from file infile, and
C parses line for firstword=Ikey characters and
C the rest of the line. Returns eof=true IFF end of
C file has been read. If key=stop is read all lines
C are ignored untill key=start is read

C** AUDIT HISTORY
C MEnutchler 16-may-83 AUTHOR
C** TYPE read from input file

C** FORMAL PARAMETERS
C in infile file from which to read
C ou key first word of line read
C ou lekey length of key
C ou line rest of line read
C ou lenline length of line read
C ou eof end of file flag

C** COMMON BLOCKS
C in charcon global character constants
C** CALER parsoc

C** METHOD
C read a line. If eof then return. If comment line, read again.
C set key to first non-blank word of line, line to rest and get their
C lengths
C**
SUBROUTINE SKIPFL(INFILE,ERR,EOF)
C* *** FORMAL PARAMETER DECLARATIONS ***
INTEGER INFILE
LOGICAL EOF,ERR
C* *** ABSTRACT ***
C* PURPOSE reads and ignores all lines until a start line
C* is read
C* AUDIT HISTORY
C MEMutchler 16-may-83 AUTHOR
C TYPE read from input file
C FORMAL PARAMETERS
Cou err an error was found
Cou eof end of file flag
C COMMON BLOCKS
Cin charcon global character constants
C CALLER parsoc
C METHOD
C read a line. If eof then return. If comment line, read again.
C set key to first non-blank word of line, line to rest and get their
C lengths
C**
12.0 MANUAL ASSIGNER MODULE

12.1 PURPOSE

The assigner provides the user with a high-level facility for creating and modifying ship construction schedules. A typical five-year shipbuilding program projection contains more than 100 ship schedules; a typical schedule record from one of the ncjodat relations contains perhaps 20 data fields. This represents a fairly large volume of data.

One of the principal activities of program analysis is program redesign, which involves changing the number of ships to be built, their timing, and/or which shipyards will perform the jobs. Creating a new program or making changes is very time consuming if done on a schedule-by-schedule basis given the amount of data the raw schedules contain. Also, analysts often prefer to perform this activity by outlining the broad pattern of the program, e.g. the number of ships of each class to be awarded each year, rather than by working with the detail of the schedules.

The assigner is a productivity tool designed to support this usage pattern. It is a specialized editor which presents the user with shipyard assignments by yard, ship class, job type, and period in a tabular fashion, and which accepts changes to the assignments. At the close of a session it will create a new set of schedule records as implied by the assignments (using construction job descriptions for each class in its computation of schedule record fields), and will write them into the ncjodat.projj relation as the current schedules for the user's scenario. Shipbuilding program schedule creation and modification is thus a quick and intuitively natural process, rather than a tedious one in which the analyst can become lost in detail.
A number of features flesh out this basic capability. The user may choose to edit only a subset of the schedules, with the subset being defined by the yard, class, and job type code names that are "on" in the assigner's Command System list menus. Individual schedules changed using the DBU may be marked as unchangeable by the assigner so that the field values the user specified are not arbitrarily overwritten. The assigner checks for the existence of appropriate job description data whenever assignments are added or modified; if they are not found, the user may put the assigner on hold and return to the Command System and the DBU to enter the job description, and then come back to finish his assigner session. Command system parameters give the user the capability to configure the assigner in various ways. For example, the algorithm which computes new schedules from the rather sketchy data on the display page may be "tuned" by setting parameter values.

12.2 SUMMARY OF STRUCTURE

The assigner is structured in three major parts, as shown in Figure 12-1. This corresponds to the three fairly separable tasks which it must perform. When invoked, the assigner must first read the schedules in the data base for the current scenario and convert these into a form usable during the editing phase. Then it must support user editing by offering a variety of interactive command options. When the user is finished, it must compute new schedules and save them in the data base.

This division into parts is implicit, showing up only in the flow of execution of the asgn.prog FORTRAN program which implements all three. This program is run by the Core as a son process; its handling somewhat resembles the DBU in that the user may return to the Command System from this process without terminating it, coming back to reactivate the process and resume his in-progress editing session later.
Figure 12-1. Assigner Flow of Control Structure
Because of the variety of its functions the assigner has a particularly rich data structure. The center of this data structure is two direct-access binary files which the assigner creates in the user's log-on group (called bufasn and cmnasn) in which all assignment records (an assignment record corresponds to a single row appearing on the display screen) and the values of important common blocks are saved. This buffering of the data both conserves data memory (crucial on the HP) and provides abort protection. If an abort occurs for any reason during an editing session the user can always recover to that point, since the assigner automatically looks for and offers the user the option of using an existing bufasn/cmnasn during its initialization. The user need only re-run the assigner to effect recovery.

Bufasn and cmnasn and the /asgn/ and /asnv1d/ common blocks are the primary means of communication between the three parts of the program. The next sections will discuss the structure of each part in more detail.

12.2.1 Terminology

Before continuing it is necessary to define some terms and concepts which will be used throughout this section. They are:

1) ASSIGNMENT: An assignment is a count of 1 appearing in any row and column of the assigner display. It is the fact that a given ship will have a given job done at a given yard in a given period. A bufasn record element or a display page cell (a row/column location) of "3", for example, denotes 3 assignments.

2) ASSIGNMENT RECORD: A row of assignments, or a bufasn record.

3) SCHEDULE: A record with specific milestone dates for a given ship job, in the form used in the ncjodat and rejodat relations. A schedule is a detailed version of an assignment.

4) TUPLE: A schedule record that is resident in one of the ncjodat or rejodat relations.
5) CLASS-JOB: Any assignment will be to perform a given job on a ship of a given class in a given yard. An assignment record is all assignments for that job on that class in that yard. On the display page, assignment rows are labeled by the class name and a single-character code indicating the job type (blank indicated new construction). A class-job is such a row or the label on the row.

6) JOB SERIES TYPE: Assignments for a given class-job can be characterized as lead-ship jobs, first-in-yard jobs, follow ship jobs, etc. What variety, or job series type, a given assignment belongs to is represented on the display page by a single-character code appearing at the location of the assignment display cell. An "L2" indicates 2 assignments in the given period, the first of which is a lead ship.

7) INBOUND/OUTBOUND: The initialization phase of assigner execution is sometimes referred to as the "inbound leg", while the DB update phase is sometimes called the "outbound leg". The intuition of the terms is based on the direction of flow of data between the assigner and the data base.

8) HARD-WIRED TUPLES: The user may specify that a given schedule tuple not be changed during the assigner's DB update step by setting the AUTOOD field of the tuple to a "NO" value in the DBU. This is a no-assigner-modify or hard-wired tuple, one which the DB update logic must not change in any way (except that it can be deleted if the user deletes all the assignments for its class job in its period).

12.2.2 Initialization Structure

Figure 12-2 diagrams the structure of the initialization phase, with an emphasis on data flow. Initialization is triggered on asgn process creation and first activation. It is not repeated if the user puts the assigner on hold during the editing phase and then returns to it later; the user is just back where he was when he left in that case.

Figure 12-3 summarizes the flow-of-control of the initialization step. Although several data structures contribute to initialization, the central goal and activity of the process is the read of data base schedules and conversion of them into assignment records.
Figure 12-2. Assigner Initialization Structure
Figure 12-3. Narrative Summary of Assigner Initialization

1. Call iniprc to swap in Core data.
2. Check to make sure user has write privileges to ncjodat.proj
3. Set flags.
4. Open ncjdat.descj to support dynamic checking of job description availability as user enters new assignments.
5. Open and read iniasn.sysro
6. Open bufasn if it exists in the log-on group, or create it.
7. Open the help text file hlpasn.sysro
8. Open cmnasn if it exists in the log-on group, or create it.
9. If bufasn and cmnasn existed, see if the user wants to use them or start fresh. If use, initialize system to its old state from cmnasn's contents, bring in the lists of valid yards/classes/jobtypes for this invocation, and flush any invalid assignments records from the old bufasn. Write the display screen and we're done.
10. If bufasn/cmnsn didn't exist or the user want to start fresh, bring in the lists of valid yards/classes/jobtypes, and read the data base for schedules, converting them into assignments records. Write the display screen.
On Figure 12-2 note in particular the read of the iniasn.sysro configuration file. The assigner is very much a data-driven system, with many important elements appearing as variables rather than being hard-wired into the code. Many of these variables are set by reading iniasn, making it easy to change them as appropriate.

12.2.3 Editor Structure

As befits an editor, the second part of the assigner is fundamentally organized around obtaining and responding to user commands. The commands are summarized in Figure 12-4.

They can be divided into three types: paging commands which let the user look at a different time frame or set of shipyards/classes, assignment-modification commands which implement the basic editing functions of add, delete, modify, and copy, and service commands such as help requests and exit requests.

Figure 12-5 presents a typical assigner display page. The page is effectively a window on the assignments records held in the bufasn file. The window is up to 15 lines long and twenty columns (periods) wide. The position of the window is changed by the page up/down (+, -, ++, --) and the page right/left (>, <, >>, <<) commands. Paging never changes the assignments; the paging algorithm merely causes a different part of the buffer to be extracted and printed to the screen.

The modification commands (A, I, D, M, R, and RC and their permutations) do alter the assignments records by addition, deletion, modification, or copying. Most require the user to specify by number both a yard and a class-job to be changed. In figure 12-5, the LSD-49 assignments at Avondale would be indicated by the number 1.2. Most prompt the user for the new or changed assignments, perform basic data validation on the
Figure 12-4. Summary of Assigner Editing Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>Obtain help from a menu</td>
<td>? #</td>
<td>Print help subject number #</td>
</tr>
<tr>
<td></td>
<td>Refresh assign display</td>
<td>#</td>
<td>EXIT assignments module</td>
</tr>
<tr>
<td>[</td>
<td>Display previous page</td>
<td>[]</td>
<td>Display topmost page</td>
</tr>
<tr>
<td>]</td>
<td>Display next page</td>
<td>[]</td>
<td>Display last page</td>
</tr>
<tr>
<td>&lt;</td>
<td>Display left neighbor</td>
<td>&lt;&lt;</td>
<td>Display leftmost page</td>
</tr>
<tr>
<td>&gt;</td>
<td>Display right neighbor</td>
<td>&gt;&gt;</td>
<td>Display rightmost page</td>
</tr>
<tr>
<td>A</td>
<td>Add a new yard</td>
<td>A #</td>
<td>Add new shipclass to yard #</td>
</tr>
<tr>
<td>I #</td>
<td>Add new yard before #</td>
<td>I #.</td>
<td>Add new class before #.</td>
</tr>
<tr>
<td>D #</td>
<td>Delete an entire yard</td>
<td>D #.</td>
<td>Delete class from yard #</td>
</tr>
<tr>
<td>MN #</td>
<td>Modify Name of yard</td>
<td>MN #.</td>
<td>Modify Name of class #.</td>
</tr>
<tr>
<td>P</td>
<td>Display to line printer</td>
<td>P #</td>
<td>Print from yard # to on LP</td>
</tr>
<tr>
<td>M #.</td>
<td>Modify assignments for class in yard #</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R #</td>
<td>Relocate yard numbered # to end of list</td>
<td>R #,##</td>
<td>Relocate yard # to before yard number ##</td>
</tr>
<tr>
<td>R #,##</td>
<td>Relocate yard # to end of yard ##'s classes</td>
<td>R #,##.</td>
<td>Relocate class #. to before class ##.</td>
</tr>
<tr>
<td>RC #</td>
<td>Like R, except copy yard instead of move</td>
<td>RC #,##</td>
<td>Again like R; copy class instead of move</td>
</tr>
<tr>
<td>RC #,##</td>
<td>Again like R; copy class instead of move</td>
<td>RC #,##.</td>
<td></td>
</tr>
</tbody>
</table>
Figure 12-5. A Typical Assigner Display Page

<table>
<thead>
<tr>
<th>Scenario: DEMO</th>
<th><em>SHIP ASSIGNMENTS</em></th>
<th>Page 1A</th>
<th>Time in: FISCYR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yard Period:</td>
<td>1 2 3 4 5 6 7 8 9</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>Shipclass T</td>
<td>T86 T87 T88 T89 T90 T91 T92 T93 T94</td>
<td>TOT</td>
<td></td>
</tr>
<tr>
<td>AVONDALE</td>
<td>#01</td>
<td>--------</td>
<td>----------------</td>
</tr>
<tr>
<td>1 LSD-41</td>
<td>2 1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2 LSD-49</td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3 T-AC-187</td>
<td>2 2 2 2 2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>BIW</td>
<td>#02</td>
<td>--------</td>
<td>----------------</td>
</tr>
<tr>
<td>1 CG-47</td>
<td>1 1 1 1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2 DDG-51</td>
<td>Y1 F2 1 2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>EB GROT</td>
<td>#03</td>
<td>--------</td>
<td>----------------</td>
</tr>
<tr>
<td>1 SSBN-726</td>
<td>1 1 1 1 1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2 SSN-21</td>
<td>L1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3 SSN-688</td>
<td>2 2 2 1 2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>GDQ</td>
<td>#04</td>
<td>--------</td>
<td>----------------</td>
</tr>
<tr>
<td>1 AE</td>
<td>Y1 1 1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2 AG</td>
<td>L1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3 AO-187</td>
<td>c 1 1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>14 33 TOTALS</td>
<td>29 24 33 30 27</td>
<td>1</td>
<td>144</td>
</tr>
</tbody>
</table>

(?=help) >
response, and alter the contents of bufasn and/or the /asgn/
common block in response.

The service commands provide miscellaneous functions such
as help, module exit, and sending of the assigner display pages
to a printer.

At its highest level the structure of the editor is ex-
tremely simple, as indicated in Figure 12-6. The asgn program
unit routine calls a routine which prompts the user for a command
and which decode the response, and then calls the executive
routine for the given command. Complexities in the editor imple-
mentation involve the details of executing particular commands;
the complexities are "pushed down" into subsidiary routines
(which will be discussed in Section 12.5).

12.2.4 Data Base Update Structure.

The task of the data base update logic is conversion of a
summary description of program schedules, the assignments, into
detailed schedules in the ncjodat.projj relation. This involves
generation of more detailed than is explicitly contained on the
assigner display page. The detail is reconstructed using inform-
ation from the new construction job schedule descriptions, or
planning factors, found in the ncjdat.descj relation, and by
applying rules of thumb.

The principal computational task is generation of the
schedule dates. A single date can be inferred for an assignment
from the column on the assigner display in which it appears; the
rest must be calculated from this "basis date" using the
milestone-to-milestone time intervals given in ncjdat.descj. The
user may specify use of various date-spreading algorithms (e.g.,
compute the schedules such that all starts in a given yard for a
given class are evenly spaced over time) by setting parameters in
the Command System menu.
Figure 12-6. Assigner Editor Structure
The overall structure of the update step is shown in Figure 12-7. The user may have assignments for ships whose schedules are in the historical and current schedule relations displayed by setting parameters in the Command System menu; these must be removed before new projected schedules are generated or too many will be created.

It is fairly common for users to enter assignments for which no job description is available in ncjdat.descj. When the schedule creation logic detects this problem it tells the user about it and returns him to the Command System, putting the assigner process on "Hold" so he can enter the required data using the DBU and come back to the assigner to finish the data base update.

Schedule records must be given hull numbers only after the write of the new schedules to ncjodat.projj is complete because of the option which lets the user mark schedules as unchangeable by the assigner (AUTOMOD="NO"). Ncjodat.projj records must have unique values for the key SCENARIO, CLASS, HULL, COMNUM; aborts can occur if the update logic assigns final hull numbers before the write since a no-assigner-mod record might have the same hull number as one the assigner attempts to add (RELATE unary key index violation results). Records are written with negative hull numbers, and these are then changed by a logic which takes the presence of no-assigner-mod records into account.

The structure for the actual schedule generation and update step is pictured in more detail in Figure 12-8. This complex task is organized around the requirements of the date-spreading algorithm, which requires as input an ordered list of the candidate ship schedule dates (one per ship) as generated from assignments' column position on the display screen. The user may specify no spreading, spreading within a class-job, or spreading within a complexity-group. For example, even intervals between starts of DDG-51 construction jobs at BATH might be desired
OPEN RELATIONS

REMOVE SHIPS FROM HISTORICAL AND CURRENT EPOCH FROM BUFASN

GENERATE NEW PROJECTED SCHEDULES FROM UPDATE NCJODAT.PROJ

GIVE THE SCHEDULES APPROPRIATE HULL NUMBERS IN NCJODAT.PROJ

TERMINATE ASSIGNER PROCESS

EXIT TO COMMAND SYSTEM, HOLDING ASSIGNER PROCESS, IF REQUIRED JOB DESCRIPTION NOT FOUND

Figure 12-7. Assigner Data Base Update Overall Structure
Figure 12-8. Schedule Creation and Update Structure
(within class-job), or perhaps even intervals between starts for the combination of DDG-51 and CG-47 jobs (complexity-group option). A complexity group is defined as any set of class-jobs in the same yard with the same value in the COMPLXGRP field in their ncjdat.descj job description records.

Once the dates are spread, producing final schedule basis dates for each ship, the "hard-wired tuple removal process" look through ncjdat.projj for tuples in the current scenario with AUTOMOD field values of "NO", finds the corresponding ship-record for each one found, and removes that ship record so no double counting occurs. The match-up between tuples and ship-records is done according to schedule basis date.

Then complete schedule records are constructed for each ship-record and are placed in a temporary direct-access file used as a buffer. The ncjdat.projj update logic then takes records from this file and either updates them over corresponding existing tuples in ncjdat, adds them to ncjdat, or deletes ncjdat tuples when there are more of those than there are schedule records.
12.3 INPUTS AND OUTPUTS

The assigner's principal outputs are the display screens presented to the user during editing (such as the sample presented in Figure 12-5), and updated schedules in the ncjdat.projj relation. The primary purpose of the screens is to support interactive editing, but they also can serve as "final" outputs when the user causes them to be printed by giving the "P" command.

The schedules which are output have the following characteristics:

1) The milestone dates they contain are consistent with both the time pattern of assignments which was showing on the display screens when the 'Q'/'W' command was given, and with the schedule planning factors read from the job description relation ncjdat.descj. When the display time units are years and the date spreading basis date is not appropriation or award, the award date will be on the month and day specified in the DEFLTAWDAY field in ncjdat.descj. When DEFLTAWDAY is used, the schedule typically cannot agree completely with both it and the milestone-to-milestone intervals. The emphasis in these cases is on computing appropriation/award dates such that subsequent runs of the assigner will show the same time pattern of assignments.

2) The YARD, CLASS, NCJOB, and JSTYP fields all have values consonant with the names and code characters which were showing on the display. COMNUM is always set to 1.

3) DATADATE and ENTRY_DATE are both set to the current date, ENTRY_BY to the current user, and DATASOURCE to 'ASSIGNER'.

4) HULL is set by the complex algorithm implemented in the newhul.rprocs RELATE execute file. The algorithm tries to create hull numbers which continue the sequences found in the historical or current schedule relations.

5) All other fields are set according to the specifications of the job description records.

An example of some of the schedules created by the assigner from the AVONDALE assignments shown in Figure 12-5 is shown in Figure 12-9.
Figure 12-9. Sample Schedules From Ncjodat.projj Relation

<table>
<thead>
<tr>
<th>$LINE</th>
<th>SCENARIO</th>
<th>CLASS</th>
<th>HULL COMNUM</th>
<th>YARD</th>
<th>NCJOBT</th>
<th>JSTYP</th>
<th>CUSTOMER</th>
<th>SHIPNAME</th>
<th>CMETHD</th>
<th>APPROP</th>
<th>AWARD</th>
<th>START</th>
<th>KEEL</th>
<th>LAUNCH</th>
<th>DELIVERY</th>
<th>COMMISSION</th>
<th>DAYSADDED</th>
<th>ASNORDER</th>
<th>DATADATE</th>
<th>DATASOURCE</th>
<th>ENTRY_BY</th>
<th>ENTRYDATE</th>
<th>AUTO</th>
<th>PROGVAR1</th>
<th>PROGVAR2</th>
<th>SUBRELUMAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>334</td>
<td>DEMO</td>
<td>LSD-41</td>
<td>42</td>
<td>1</td>
<td>AVONDALE</td>
<td>NEWCON</td>
<td>ORDFO</td>
<td>USN</td>
<td>10/01/1985</td>
<td>11/01/1985</td>
<td>5/01/1987</td>
<td>1/01/1989</td>
<td>5/01/1990</td>
<td>6/01/1990</td>
<td>0</td>
<td>57553221</td>
<td>10/28/1984</td>
<td>908/SHAPM</td>
<td>MARK</td>
<td>10/28/1984</td>
<td>YES</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>335</td>
<td>DEMO</td>
<td>LSD-41</td>
<td>43</td>
<td>1</td>
<td>AVONDALE</td>
<td>NEWCON</td>
<td>ORDFO</td>
<td>USN</td>
<td>10/01/1984</td>
<td>11/30/1984</td>
<td>6/30/1986</td>
<td>1/30/1989</td>
<td>0</td>
<td>9/01/1984</td>
<td>908/SHAPM</td>
<td>DBA</td>
<td>9/25/1984</td>
<td>0</td>
<td>41</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>338</td>
<td>DEMO</td>
<td>LSD-49</td>
<td>49</td>
<td>1</td>
<td>AVONDALE</td>
<td>NEWCON</td>
<td>LEAD</td>
<td>USN</td>
<td>10/01/1987</td>
<td>11/01/1987</td>
<td>7/01/1989</td>
<td>6/01/1990</td>
<td>6/01/1993</td>
<td>7/01/1993</td>
<td>7/01/1993</td>
<td>0</td>
<td>57553222</td>
<td>10/28/1984</td>
<td>908/SHAPM</td>
<td>MARK</td>
<td>10/28/1984</td>
<td>YES</td>
<td>49</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>339</td>
<td>DEMO</td>
<td>LSD-49</td>
<td>50</td>
<td>1</td>
<td>AVONDALE</td>
<td>NEWCON</td>
<td>ORDFO</td>
<td>USN</td>
<td>10/01/1987</td>
<td>11/01/1987</td>
<td>5/02/1989</td>
<td>1/02/1990</td>
<td>12/02/1991</td>
<td>12/02/1993</td>
<td>1/02/1994</td>
<td>0</td>
<td>57553222</td>
<td>10/28/1984</td>
<td>908/SHAPM</td>
<td>MARK</td>
<td>10/28/1984</td>
<td>YES</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>341</td>
<td>DEMO</td>
<td>LSD-49</td>
<td>52</td>
<td>1</td>
<td>AVONDALE</td>
<td>NEWCON</td>
<td>ORDFO</td>
<td>USN</td>
<td>10/01/1988</td>
<td>11/01/1988</td>
<td>5/03/1990</td>
<td>1/03/1991</td>
<td>12/03/1992</td>
<td>12/03/1994</td>
<td>1/03/1995</td>
<td>0</td>
<td>57553222</td>
<td>10/28/1984</td>
<td>908/SHAPM</td>
<td>MARK</td>
<td>10/28/1984</td>
<td>YES</td>
<td>52</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Inputs the assigner requires include the iniasn.sysro configuration file, the Core data swap segment (with its current values from the assigner's parameter menu), the lists of yards, classes, and job types of interest for the run from the assigner's Command System list menus, the schedules found in the data base for the given scenario at the time of execution, the job description relation's contents for the given scenario, scenario key field values from the scenario system's extra data segment, and (of course) editing-session commands and inputs from the user.

12.3.1 The Configuration File and the Help File

Figure 12-10 shows a copy of the assigner configuration file stored in iniasn.sysro. This file is read during initialization to discover the values of certain assigner operating parameters. The file contains six lines, read sequentially using FORTRAN formatted i/o. The formats are typically fixed, so that column position of the data is important. The contents of the lines are as follows:

**LINE 1:** Six integers. The first may be 0 or 1, specifying whether the main initialization and editing lprnt (number 7) will be off or on. The second is the FORTRAN unit number for diagnostic output, typically the same as standard output. If changed, be sure to give the proper FILE equations prior to assigner execution so that diagnostics go to the proper device. The third number may be 0 or 1, specifying whether subsequent reads from iniasn will not or will be echoed. The fourth number specifies the FORTRAN unit number the echo will be sent to (same warnings apply as for the diagnostic unit). The fifth and sixth numbers specify the unit numbers for normal interactive input and output. The text string to the right of the 6 I5 fields is a brief reminder of the purpose of each field.

**LINE 2:** An integer, a *2 character string, and an up to *16 character string. The *2 string is the short interactive command prompt, set to "> " in the example. The large string is the long interactive prompt, to which the short prompt is concatenated.
Figure 12-10. Text of INIASN.SYSRO Configuration File

0  6  0  6  5  6  (6I5) iprint,ioutp,iecho,uecho,in,out
9½ (?=help)  "?" short, "(?=help)" long
10   LF  ncrs
BUFFASN   1  500
HLPASN.SYSRO  2
CMNASN   3
The number, given before the strings, is the number of characters in the long string. The command prompt is thus very easily configurable.

LINE 3: A single number specifying user input uppercasing rules for yard and class names. The remainder of the fields on the line are obsolete and ignored. The number specifies a bit map where bits 1 and 3 on indicate uppercasing only of the first characters of names, bits 2 and 4 uppercasing of all characters in names (the job type code character at the end of a class-job name is lowercased later in the current logic). The setting of 10 invokes complete uppercasing.

LINE 4: The name, FORTRAN i/o unit number, and maximum number of records allowed in the bufasn assignments holding buffer file. The file will be created in the log-on group and i/o will take place through the given unit number. Should 500 class-jobs ever be insufficient the limit can easily be expanded by changing the third parameter of this line.

LINE 5: The name and FORTRAN i/o unit number of the prthlp-readable assigner help text file, currently hlpasn.sysro.

LINE 6: The name and FORTRAN i/o unit number for the cmnasn file which is companion to bufasn. It always has only two records, since it contains the contents of two of the common blocks in the /asgn/ include file. It's primary function is to support recovery after an abort by holding the system status at the time of the abort.

The great majority of on-line help for the assigner is stored in the hlpasn.sysro file. This is a standard EDITOR-type ASCII file divided into sections by the %BEGIN statements recognized by the prthlp utility. This file is opened during initialization and read as necessary in response to user help requests.

12.3.2 Variables From the Assigner's Command System Parameter Menu

The assigner actually uses a good deal of the data in the swap segment provided by the Core and read during initialization by a call to the iniprc utility. Of interest are lprnts settings and the contents of the /scenar/ and /uzrprv/ common blocks. Of
principal interest are the setting of variables on the assigner's parameter menu. These are read by asnlbs, which transfers the values from their storage locations in /pvalue/ to variables in /asgn/. A sample of the assigner parameter menu is shown in Figure 12-11. The meaning and use of each parameter is:

1) TIME UNIT: Specifies the amount of time that each display page column represents. When combined with STARTING DATE, also determines the first day of the period each column represents.

2) STARTING DATE: First day of the period the user wants assignments displayed for. It is permissible to work with a subset or superset of the periods represented by schedules currently in ncjodat.projj. If the date given is not the first day of the period type specified it is moved back to that day (i.e. if fiscal years is the TIME UNIT and the STARTING DATE is given as 1/1/1986, it will be moved back to the first day of fiscal 1986, 10/1/1985).

3) ENDING DATE: Last day of the time span of interest. Determines the number of columns on the display page in combination with the first two parameters.

4) CANDIDATE SHIP YARDS: A gate to a list menu with the names of all the shipyards ALIAS knows about. The user can work with assignments for only a subset of these yards by setting some of the list names to 'OFF' status. No assignments are loaded for the 'off' yards, and their schedule records in ncjodat.projj (if any) are not updated.

5) CANDIDATE SHIP CLASSES: Like candidate yards, lets the user work with a subset of ship classes only.

6) CANDIDATE JOB TYPES: Like the previous two, lets the user see and update only class-jobs of certain job types.

7) DISPLAY BASIS: The column any given assignment is placed in depends on which schedule milestone date is being used as the basis for making assignments. For any given schedule, a different column will typically be chosen if the basis is DELIVERY rather than AWARD. The setting of this parameter determines which milestone is used as the basis date.

8) ADJUST BASIS: The schedule milestone date being used as the basis for date-spreading. Though the user may specify assignments in terms of AWARDs (display basis), he may want, e.g., the start dates of the resulting schedule evenly spaced over time.
Figure 12-11. Sample Assigner Parameter Menu

Menu is ASNPRM

* ALIAS COMMAND SYSTEM *

Scenario is DEMO

MANUAL ASSIGENER MODULE INITIALIZATION PARAMETERS

1. TIME UNIT = FISCYR (FISCYR,CALYR,QTR,MONTH,WEEK,DAY)
2. STARTING DATE = 1/1/1980 (MM/DD/YYYY)
3. ENDING DATE = 12/31/1999 (MM/DD/YYYY)
4. CANDIDATE SHIP YARDS = LIST (ALL/LIST)
5. CANDIDATE SHIP CLASSES = LIST (ALL/LIST)
6. CANDIDATE JOB TYPES = LIST (ALL/LIST)
7. DISPLAY BASIS = AWARD (APPROP,AWD,START,KEEL,LNCH,DELIV)
8. ADJUST BASIS = START (APPROP,AWD,START,KEEL,LNCH,DELIV)
9. ADJUST MODE = PROGRAM (NONE,PROGRAM,COMPLX-GROUP)
10. JOBS EPOCH OPTION = PROJ (ALL,CURR/PROJ,PROJ)
11. SHIPCLASS SORT ORDER = ALPHABETIC (ALPHABETIC,INPUT ORDER)
12. SHIPEYARD SORT ORDER = INPUT ORDER (ALPHABETIC,INPUT ORDER)
13. AUTO REFRESH = OFF (ON,OFF)

COMMAND:
9) **ADJUST MODE:** This setting controls the operation of the
date-spreading algorithm employed during schedule
generation. If NONE, then aspred is simply never
called. If PROGRAM, aspred is fed only the ships for a
single class-job in a single yard when it is called. If
COMPLX-GROUP, aspred is fed all class-jobs in a yard in
the same complexity group, where complexity group is
specified by the COMPLXGRP field value in the
ncjdat.descj relation. The latter case might be
desirable when a yard is building similar ships of
different classes, e.g. DDG's and CG's.

10) **JOBS EPOCH OPTION:** Controls which relations are read
for schedules during initialization. PROJ is the normal
setting since only projected schedules can be updated
anyway. Note that if the setting is not PROJ then
ascen must be called during the outbound leg, at a
substantial processing penalty.

11) **SHIPCLASS SORT ORDER:** The user may specify that
class-jobs be listed alphabetically within a given yard
on the display screen, or in the order in which they
were input or displayed during the last session.

12) **SHIPYARD SORT ORDER:** Similar to the previous
parameter, but its setting has no effect at this time.
The display order is always alphabetic.

13) **AUTO REFRESH:** If ON, the assigner display will be
refreshed (rewritten) every time the user gives a
command which changes its contents or writes substantial
output to the page. If OFF, the user must always
request a refresh via the command.

12.3.3 **List Menus**

The assigner's list menus were alluded to in the last
section. Figure 12-12 shows the third of them, the CANDIDATE JOB
TYPES menu. The user can restrict the types of job for which
assignments will be read from the data base and displayed; this
restriction also prevents the user from entering any new assign-
ments of the "off" job types. Note that new job type codes added
to the system must be explicitly added to a user's scenario with
the NC_JOB_TYPES and RE_JOB_TYPES screens of the DBU before they
will appear on this list menu, and thus before their assignments
can be displayed.
Figure 12-12. Sample Assigner Valid Job Types List Menu

<table>
<thead>
<tr>
<th>Menu is CHJ...</th>
<th>* ALIAS COMMAND SYSTEM * Scenario is DEMO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td><strong>CHOOSE THE SET OF VALID JOBS WHICH MAY BE ASSIGNED</strong></td>
<td></td>
</tr>
<tr>
<td>1. * CONV</td>
<td>5. * REPAIR</td>
</tr>
<tr>
<td>3. * REACT</td>
<td>7. * SLPCNV</td>
</tr>
<tr>
<td>4. * REFUEL</td>
<td></td>
</tr>
</tbody>
</table>

**COMMAND:**
The lists are stored in the valcls, valyds, and vljtyp relations in the .mnurel group (.makmenu for the development system).

12.3.4 Sample Schedules

A sample of ncjodat.proj schedule records was shown in Figure 12-9. For input purposes the assigner is only interested in a restricted set of the fields: SCENARIO, YARD, CLASS, HULL, COMNUM, NCJOBT, JSTYP, DATADATE, ENTRY_DATE, and the particular milestone date field being used as display basis (e.g. AWARD) are the only ones read. DATADATE and ENTRY_DATE are consulted during reads of the historical and current relations to ensure that no double-counting occurs due to multiple reads of the same schedule for different data dates.

12.3.5 Job Description Records

Figure 12-13 shows sample job schedule description records from ncjdat.descj, which are read during the DB update phase in order to gather information necessary to construct complete schedule records from the assignments. Note particularly that the YARD field may take on the name of a specific yard or ANY; the assigner always searches for a match on the name of the yard an assignment is in first (along with matches on the other keys, of course), but will take any ANY record if the first search fails.

12.3.6 Scenario Key Field Values

The assigner makes use of the scenario system via the DBIF in the usual fashion of ALIAS modules. Relations opened via the DBIF have the proper scenario key field values for the current scenario placed in the cursen array of the /scenar/ block; these values are then used to construct keys for searches.

12.3.7 User Inputs

User inputs during the editing phase are in two forms: commands and assignments. The commands must be from the list
Figure 12-13. Sample Job Description Records From Ncjdat.descj

<table>
<thead>
<tr>
<th>$LINE</th>
<th>SCENARIO</th>
<th>CLASS</th>
<th>NCJOBT</th>
<th>YARD</th>
<th>JSTYP</th>
<th>COMNUM</th>
<th>CMETHD</th>
<th>CUSTOMER</th>
<th>COMPLEXGRP</th>
<th>DEFLT</th>
<th>DAYSADDED</th>
<th>APPROP_AWD</th>
<th>AWD_ST</th>
<th>ST_KL</th>
<th>KL_LN</th>
<th>LN_DL</th>
<th>DL_COM</th>
<th>TIMUNT</th>
<th>DATASOURCE</th>
<th>DATADATE</th>
<th>ENTRY</th>
<th>ENTRY_BY</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>DEMO</td>
<td>LSD-41</td>
<td>NEWCON</td>
<td>ANY</td>
<td>ORDFOL</td>
<td>1</td>
<td>MODULZ</td>
<td>USN</td>
<td>11/01 0</td>
<td>1</td>
<td>12</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8/01/1984</td>
<td>8/02/1984</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/01</td>
<td>DEMO</td>
<td>LSD-49</td>
<td>NEWCON</td>
<td>ANY</td>
<td>ORDFOL</td>
<td>1</td>
<td>MODULZ</td>
<td>USN</td>
<td>11/01 0</td>
<td>1</td>
<td>12</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8/01/1984</td>
<td>8/02/1984</td>
<td></td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>DEMO</td>
<td>LSD-49</td>
<td>NEWCON</td>
<td>ANY</td>
<td>LEAD</td>
<td>1</td>
<td>MODULZ</td>
<td>USN</td>
<td>11/01 0</td>
<td>1</td>
<td>12</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8/01/1984</td>
<td>8/02/1984</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
given in Figure 12-4. The assignments are display-page lines in the general form shown on the display page: the user enters a class name and job type character code, and numbers of ships in each period for that class-job. See Section 3 of the User's Guide for examples of the formats of prompts and responses.

12.4 DATA STRUCTURES

The emphasis in this section will be on data structures internal to an assigner run, though many of the input and output structures discussed in the previous section can be fruitfully thought of as assigner data structures. Examples are the schedule and job description relations, and the parameter and list menus.

The internal data structures will be discussed according to function rather than type. As a preliminary, Table 12-1 presents an annotated listing of the include files used by the assigner. The common blocks in these files range over all the ALIAS block subtypes: ordinary common blocks, record structures, and linked lists.

12.4.1 System Status Data Structure

In some sense the most important include file is the asgn.incl file, which contains four common blocks holding most of the system status information maintained during the initialization and editing phases. The values in two of these blocks, /casgn/ and /nasgn/, are continually stored into the cmnasn file so that system status information is recoverable in the event of an abort. These blocks contain such data items as the names of shipyards for which assignments are loaded, the arrays with row and column assignment totals, etc.

Cmnasn is created along with bufasn during initialization in the log-on group.
### TABLE 12-1. Include Files Used By the Assigner

<table>
<thead>
<tr>
<th>FILENAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASGN</td>
<td>This voluminous include file contains four common blocks and several FORTRAN PARAMETER statements. It is the most important in-memory data structure for the initialization and editing phases of execution. In addition to flags, operating variables, display page totaling arrays, unit numbers, etc., the file contains the record structure (block /basgn/) which is used in communicating with the bufasn file.</td>
</tr>
<tr>
<td>ASHLDR</td>
<td>A record transfer structure used during the DB update phase to move assignments records (in their bufasn form) between the process data stack and an extra data segment used as a holding area. All assignments records in the complexity group currently being processed are maintained in the &quot;hldbuf&quot; data segment.</td>
</tr>
<tr>
<td>ASJD</td>
<td>ASSigner Job Descriptions. Used during the DB update phase to hold all the job description records for a particular class-job retrieved from the ncjdat.descj relation. There can be several such records, e.g. one for a LEAD job series type, one for an ORDPOL series type, etc. /ASJC/ is NOT a record structure (i.e. it is not used in the retrievals from the relation), but rather is a storage area consulted by the ncjodat record construction logic as necessary.</td>
</tr>
<tr>
<td>ASNOCR</td>
<td>ASSigner Outbound CuRsors. A common block of integer variables in which cursor indexes returned by the DBIF during the DB update relation-opening phase are stored. Used only by the update phase.</td>
</tr>
<tr>
<td>ASNVLD</td>
<td>The lists of valid (of-interest) ship class names, yard names, and job type code names as read from the assigner's three Command System list menus. Also, a list of valid job series type character codes for insertion in display page cells to indicate things like &quot;lead ship in this period (L)&quot;. Since both job type and job series type are specified on the screen using single-character codes, this block has matching arrays of names and character codes to facilitate searching and retrieval.</td>
</tr>
<tr>
<td>ASOPRM</td>
<td>ASSigner Outbound PaRaMeters. FORTRAN parameter statements and variables with Command System parameter menu parameter settings of interest to the DB update phase.</td>
</tr>
</tbody>
</table>
TABLE 12-1. Include Files Used By the Assigner

<table>
<thead>
<tr>
<th>FILENAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASRBUF</td>
<td>Assigner ship Record Buffer. Holds the first form of schedule records generated during the DB update phase. Managed as a linked list which holds records only for the current complexity group. The records consist only of class name, job series type, the &quot;display date&quot; (the schedule date derived from the given job's column position on the display page), and the adjust date (the milestone used as the basis date during date spreading---need not be the same as the display milestone). Schedule records need only contain this information through the &quot;hard-wire&quot; tuple removal phase of the update process.</td>
</tr>
<tr>
<td>ASTFR</td>
<td>Assigner Tuple File Record buffer. This include file contains two common blocks which function as record buffers for use in RELATE queries of and updates to the ncjodat.projj relation, and for similar operations performed on the schedule record holding file. During the actual relation update step it is necessary to have a record from each source current in memory at all times.</td>
</tr>
<tr>
<td>PVALUE</td>
<td>The System Core data structure which holds the current values for all command system parameter menus. Consulted during both the initialization phase and the DB update phase. In both cases the values are read by a service routine located in recomp.src and moved into common blocks dedicated to the assigner.</td>
</tr>
<tr>
<td>CONST</td>
<td>A block of commonly used constant values, e.g. the largest 32-bit integer number.</td>
</tr>
<tr>
<td>FLD05</td>
<td>Field list for reads and updates of the ncjodat.projj relation. The astfr include file must appear above it in any routine in which it is used.</td>
</tr>
<tr>
<td>FLD06</td>
<td>Field list for reads of the ncjdat.descj job description relation. Associated with the rcrd06 include file.</td>
</tr>
<tr>
<td>IOC</td>
<td>The standard ALIAS common block of key FORTRAN i/o unit numbers, e.g. those for terminal input and output.</td>
</tr>
</tbody>
</table>
TABLE 12-1. Include Files Used By the Assigner

<table>
<thead>
<tr>
<th>FILENAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPRNTS</td>
<td>The ALIAS array of logical variables (switches) controlling the operation of diagnostic prints.</td>
</tr>
<tr>
<td>PRMCRS</td>
<td>Permanently open ALIAS cursors (in each process). Used only by the iniprc and liston routines.</td>
</tr>
<tr>
<td>RCRD06</td>
<td>Record buffer which receives tuples from the ncjdat.descj relation. Used only as word-aligned temporary storage—the job descriptions are always moved into /asjd/.</td>
</tr>
<tr>
<td>SCENAR</td>
<td>Information about the current scenario and about cursors opened through the DBIF. Initial values for the block are swapped into the assigner process memory during the call to iniprc.</td>
</tr>
<tr>
<td>SCCHR</td>
<td>FORTRAN parameter statements defining the command characters the assigner will recognize and a code number for each one. This block is actually used by several modules, so the assigner does not have a function it will perform for each command character found here.</td>
</tr>
<tr>
<td>SENPRM</td>
<td>Scenario system parameters. Required by the /scenar/ block.</td>
</tr>
<tr>
<td>SNRREF</td>
<td>A block of declarations supporting the scenario system's low-level utilities which search the scenario system extra data segment for scenario key values. Direct use of these utilities is made in the assigner when SELECTs are given.</td>
</tr>
<tr>
<td>TDDATE</td>
<td>The file of declarations and statement functions which supports full use of the ALIAS date manipulation utility subsystem by a routine.</td>
</tr>
</tbody>
</table>
During the DB update phase two additional common blocks hold status information, /asoprm/ and /asnocr/.

12.4.2 Valid Names Data Structure

The /asnvld/ common block contains lists of all the of-interest (valid) yard, class, and job type names turned "on" by the user in the assigner's Command System list menus. Job series type names are also read from the jstyp.legals relation and stored here.

The job type and job series type lists are maintained in dual form: the names are needed during initialization and DB update phases for comparison with field values in relations, but the single-character code values used on the display screen to represent these names are needed during the editing phase. The names are maintained in an array, while the code values are maintained in corresponding elements of character*1 arrays. A match on an element of one array thus automatically yields an index number for the element of the corresponding array.

The code characters are read from the jobtyp.legals and jstyp.legals relations. Additional job type code names and series type names must be assigned unique character codes in these relations when they are added to the system.

12.4.3 Assignments Record Data Structure

A third block in asgn.incl (/basgn/) is the transfer record used to communicate with the bufasn file. Only a single assignment record is ever in memory at any given time, and it is stored in this block. Bufasn is a direct-access binary file with one record for each assignments record (class-job within a yard) displayable on the screen. Bufasn records are managed as a set of linked lists, one list per yard. The firstp array in /nasgn/ holds the record number of the first assignments record for each yard; subsequent records are pointed to using the first word of the bufasn record. The remainder of the record contains the
class name, a storage location for ASNORDER (which holds a time stamp of when the record was first entered), the row-total of assignments in the record, and two arrays giving the number of assignments in each cell (display column) and a code indicating the character code to appear in each cell.

This data structure conserves on memory to the maximum extent possible, is efficient in terms of retrieval time during display generation, and provides abort protection since all assignments are maintained on disk at all times.

Display records are generated dynamically from this data structure during the refresh process, rather than being held in memory.

An additional assignments record structure is used during the DB update phase. This consists of the /ashldr/ common block, which is a record containing the equivalent of the two arrays from a bufasn record, and an extra data segment which can hold several of these records. The segment stores the records for all ships in the current complexity-group during the new-tuple generation process. The segment is used to conserve on memory; each assignments record requires 520 words, since 260 periods is the configured capacity of the assigner.

12.4.4 The Job Descriptions Data Structure

New tuples are produced by the DB update phase one class-job at a time after date-spreading has been completed. Several job description records may be required for any given class-job since each individual ship may be of a different job series type (e.g. LEAD, ORDFOL). All the descriptions for a given class-job at a given yard are thus maintained in memory simultaneously during schedule generation (to avoid time-consuming multiple searches of ncjdat.descj) in the /asjd/ common block. This block is a series of arrays dimensioned by the maximum memory capacity for job descriptions. The routines which use the block's con-
tents first call the asgpf routine to get an index to a 'row' in /asjd/, and then just supply this index in any assignment statements using /asjd/ variables as the source.

/asjd/ is loaded by reads from ncjdat.descj using /rcrd06/ as the transfer record. The aspftr service routine is called after each read to transfer the given description to a 'row' in /asjd/.

12.4.5 The Schedule Tuple Data Structure

Schedule tuples are handled differently by the initialization and DB update phases. During initialization, a subset of tuple fields are read into a six-tuple array locally static in the asndbi routine. This supports the ordered-retrieval algorithm discussed in Section 12.5.1.

Several schedule record data structures are used during the DB update phase. The algorithm first generates one record per assignment in the /asrbuf/ common block, which is managed as a linked list with a capacity of 200 ships. Records are placed in the block in order of ship adjustment milestone in order to support the date spreading algorithm.

After dates are spread complete tuple images are generated and placed in a temporary holding file in the log-on group called tupfil. The /astfr/ block in the astfr.incl include file is the record structure used during read/writes from/to this file.

During the actual update of ncjodat.projj, records are read simultaneously from tupfil (into /astfr/) and from ncjodat.projj (into /astup/ in the astfr.incl file) and compared, with the /astfr/ image eventually being written into ncjodat.

12.4.6 Command Processing Data Structure

A command given during the editing phase may consist of up to three parts: its first or main-command character, its second
or subcommand character, and one or more numbers separated by commas. The main and subcommand characters are compared with the list of known command codes in the scrchr.incl file of FORTRAN parameter statements and converted into index numbers. These indexes and the user-supplied numbers are then stored in variables declared in the asgn program unit (thus effectively global variables), and are passed to the proper main-command processing routine.

The search for a command character match is done on the scrchr parameter; the location of the match serves as the index. Note that the index values formally assigned to the characters in the rest of the scrchr.incl file therefore depend on the position of the character in the scrchr string.

12.5 PROCESSING LOGIC

The assigner is a very large program replete with logic and algorithms. In this section only the major algorithms whose structure and operation are not fairly obvious in the code will be discussed. To thoroughly understand how part or all of the program works it is necessary to consult the in-line documentation and the code itself.

Table 12-2 contains an annotated list of all of the FORTRAN routines in the assigner, not including utilities.

The calling tree diagrams which appear below show only those non-utility routines maintained as part of the ASGNxxx.src source code libraries. In addition, middle-level routines' subsidiary trees are typically shown on only the first diagram in which they appear in order to save space.

12.5.1 Initialization Phase Logic

Figure 12-14 is a calling tree diagram for the initialization phase of assigner execution. Also consult Figure 12-2 (initialization structure) for a pictorial display of the logic.
Table 12-2. Annotated List of Assigner Routines

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCDAY</td>
<td>Low-level DB update logic utility which computes schedule milestone dates given a basis date and the index of a job-description in /asjd/. The job description gives time intervals between milestones; ascday just determines the appropriate number of intervals to increment the basis date by. An integer*4 function.</td>
</tr>
<tr>
<td>ASCDSP</td>
<td>A DB update schedule data modification routine which ensures that the award date in a schedule conforms to the default award day (DFLTAWDAY) field value in the appropriate job description relation. The routine is operative only when the display periods are years. An integer*4 function.</td>
</tr>
<tr>
<td>ASCMPG</td>
<td>DB update routine, &quot;Assigner-Complexity Group.&quot; Finds out what complexity group each class-job of interest in a given shipyard falls in. Output is a series of code numbers corresponding to class-job names, with each group having the same code number (the numbers are arbitrary).</td>
</tr>
<tr>
<td>ASDWRN</td>
<td>Prints a warning to the effect that a hard-wire (no-assigner-modification) schedule has been deleted because there was no assignment left for it on the assigner display page. Part of the DB update logic.</td>
</tr>
<tr>
<td>ASGN</td>
<td>The main program unit for the assigner module. Both supervises the three main phases of execution by making appropriate calls and is the executive for the editing phase.</td>
</tr>
<tr>
<td>ASGNXT</td>
<td>Moves all the assignments records for class-jobs in a given complexity group into the hldbuf data structure (extra data segment) in preparation for date spreading and schedule generation. Part of the DB update logic.</td>
</tr>
<tr>
<td>ASGPF</td>
<td>Integer function returning the index in /asjd/ of the job description most appropriate for a given class-job in a given yard of a given series type. Presumes that a description for the class-job is available; this routine's task is to find the closest match on series type. Asgpf assumes an appropriate call to aspfld has been made to load the job descriptions for the given yard and class-job into</td>
</tr>
<tr>
<td>ROUTINE</td>
<td>PURPOSE</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ASJ /asj/</td>
<td>DB update routine which removes ship records &quot;associated&quot; with &quot;hard-wired&quot; or no-assigner-modify tuples in ncjodat.projj. The method is to read ncjodat (via a selection) and, for every hard-wire tuple found, locate the ship record of the same class/job type with the closest display date, and remove it from the linked list in /asrbuf/. If there is no ship record in the same period, it is assumed the user wants the hard-wire tuple deleted, which is done.</td>
</tr>
<tr>
<td>ASHARD</td>
<td></td>
</tr>
<tr>
<td>ASHTRB</td>
<td>DB update routine which converts the hldbuf representation of assignments records for a given ship-job complexity group to the /asrbuf/ linked list of ship records. The ship records include both a display date and an adjustment date estimate; these are used by the date spreading logic.</td>
</tr>
<tr>
<td>ASJOI</td>
<td>Part of the DB update logic. Logical function which decides whether a given class-job needs to have schedule records generated for posting to ncjodat.projj. &quot;ASSigner Jobs Of Interest.&quot;</td>
</tr>
<tr>
<td>ASN1ST</td>
<td>An assigner version of the fddate date utility, which returns the first day of a given period. An integer function.</td>
</tr>
<tr>
<td>ASNADD</td>
<td>Contains entry point asnins. Adds a new yard and/or class-job assignment to the assignments record buffer and the display page. Prompts the user for names and assignments, does error checking, and puts the response into the data structures. Implements the &quot;A&quot; and &quot;I&quot; commands.</td>
</tr>
<tr>
<td>ASNALO</td>
<td>Low-level editing phase routine which allocates a new assignment record (bufasn record) onto the free chain (linked list) of such records.</td>
</tr>
<tr>
<td>ASNALT</td>
<td>Implements the &quot;M&quot; command at a low level by writing the modified assignments record as given by the user to the record buffer.</td>
</tr>
<tr>
<td>ASNAMM</td>
<td>Implements the yard/class-job name changing capability. Prompt the user for the new name and</td>
</tr>
<tr>
<td>ROUTINE</td>
<td>PURPOSE</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ASNCLN</td>
<td>Removes assignments based on a load from the relations with schedules for the historical and current epochs by re-reading those relations and decrementing the assignments buffer (bufasn) for every one found. Clumsy and time-consuming but the only way short of marking each bufasn element, also clumsy.</td>
</tr>
<tr>
<td>ASNCLR</td>
<td>Does a screen clear or formfeed, depending on whether output is to the screen or the printer.</td>
</tr>
<tr>
<td>ASNCMD</td>
<td>This routine prompts for, reads, and decodes user command input during the editing phase. Commands are broken into the main and sub command characters and any numeric specifications which follow them.</td>
</tr>
<tr>
<td>ASNCNV</td>
<td>Executive which supervises conversion of the bufasn assignments records into ncjodat.projj tuples. See Figure 12-8 for a summary of its flow of control.</td>
</tr>
<tr>
<td>ASNCOD</td>
<td>Servant of asndbi, used to set the job series type letter code value for a particular cell of the buffer. Ensures that the code shown on the page end up being the one attached to the &quot;highest&quot; ship &quot;in&quot; the cell, where the order from highest to lowest is, e.g., lead ship, first follow, lead in yard, ordinary follow. If there are 8 ships in the cell (i.e. ad 8 is displayed there on the screen) and two of them are a lead ship and a first follow, the cell will show an 'L'.</td>
</tr>
<tr>
<td>ASNCPY</td>
<td>Makes a new copy of a yard or a class-job within a yard, prompting user for the names for the new copy,</td>
</tr>
</tbody>
</table>
### Table 12-2. Annotated List of Assigner Routines

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASNDBI</td>
<td>Executive for the load of schedules from the database and their conversion into assignments records in(bufasn) and /asgn/. Reads records in key order from all six schedule relations simultaneously (see text), constructing assignment records as it goes. Creates new yards and classes within yards as necessary. On completion of the DB read, reorders assignment records according to input order if user has asked for that.</td>
</tr>
<tr>
<td>ASNDBR</td>
<td>The DB update phase routine which actually posts the newly created schedules to the ncjodat.projj relation. Must ensure that at the close of the assigner session the schedules in ncjodat are completely in consonance with the assignments which were showing on the display screen. Must also take into account the fact that the user could have been working with a limited period of time or a limited list of valid yards/classes/job types. Operates in a fashion basically similar to the asndbi routine: a tuple from the relation and a record from the new tuple holding file are always kept constant in buffers; action decisions are made on the basis of a comparison of their key values. The actions possible are to get the next tuple, update the existing one, delete it, or add a new tuple.</td>
</tr>
<tr>
<td>ASNDEL</td>
<td>Deletes one or more assignments records from the data structure, up to an entire yard. Implements the &quot;D&quot; command.</td>
</tr>
<tr>
<td>ASNDOT</td>
<td>Mid-level utility which prints the prompt (&quot;dots&quot;) for assignment record addition or modification. Also reads, checks, and decodes the input.</td>
</tr>
<tr>
<td>ASNDWN</td>
<td>Does the computations for a next-vertical-page command.</td>
</tr>
<tr>
<td>ASNEC</td>
<td>Reports a command input error or some other status condition to the user, pausing to let the user read the message.</td>
</tr>
<tr>
<td>ASNEND</td>
<td>An obsolete close-relations and finish up routine.</td>
</tr>
<tr>
<td>ROUTINE</td>
<td>PURPOSE</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>ASNEOI</td>
<td>Resets i/o unit numbers back to the terminal when an end-of-file is encountered on the current input unit number. Useful for detecting and resetting after the end of execution of the &quot;P&quot; command, for example.</td>
</tr>
<tr>
<td>ASNFND</td>
<td>Low-level routine which locates a particular class-job's assignments in a particular yard and brings it into the assignment buffer in /asgn/.</td>
</tr>
<tr>
<td>ASNFOL</td>
<td>Implements the vertical page-forward command (+ and ++). Causes the index numbers of the top yard and class-job (used by the display refresh logic) to be recomputed.</td>
</tr>
<tr>
<td>ASNGCQ</td>
<td>Logical function which asnout uses to ask the user if he wants to skip the DB update step.</td>
</tr>
<tr>
<td>ASNHLD</td>
<td>Implements the &quot;H&quot; (hold) command; suspends the assigner process and reactivates the Core process.</td>
</tr>
<tr>
<td>ASNHLP</td>
<td>Assigner help subsystem executive. Responds to the &quot;?&quot; command. Prompts the user with a menu of help choices. Accepts and implements the response.</td>
</tr>
<tr>
<td>ASNHUL</td>
<td>Responsible for updating the hull numbers in schedules newly posted to ncjodat.projj during the DB update phase. As posted the schedules have negative hull numbers to ensure that no unary key violations occur as a result of collisions with &quot;hard-wired&quot; schedules. The routine carries out its task by having RELATE execute the newhul.rprocs EXECUTE file, which contains the actual logic. However, asnhul must write proper scenario key field values into newhul.rprocs before it is executed so that scenario security is maintained.</td>
</tr>
<tr>
<td>ASNINI</td>
<td>Executive for the initialization phase. Does or supervises completion of everything necessary before user assignments editing can begin. Major steps include reading the iniasn.sysro configuration file, creation of the bufasn and cmnasn working files, load of valid lists from the list type relations, and loading of schedules from the data base and conversion of these into assignments records in bufasn.</td>
</tr>
</tbody>
</table>
Table 12-2. Annotated List of Assigner Routines

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASNINS</td>
<td>An entry point in asnadd which lets the user specify where on the display page the new yard/assignments are to be placed.</td>
</tr>
<tr>
<td>ASNLBL</td>
<td>Prints the top two rows of the screen display, which give status information (i.e. name of the current scenario) and the period labels.</td>
</tr>
<tr>
<td>ASNLBS</td>
<td>Performs part of the initialization of the /asgn/ common block, in particular for those variables whose values depend on setting in the Command System's assigner parameter menu, e.g. time units/period type option. Formats and stores relevant parts of the screen display.</td>
</tr>
<tr>
<td>ASNLEV</td>
<td>Closes all files and relations at the end of the DB update phase, preparatory to assigner process termination.</td>
</tr>
<tr>
<td>ASNLFT</td>
<td>Contains the asnrrot entry point as well. Implements the page-right and page-left horizontal (over-periods) paging commands (&gt;), &gt;&gt;, &lt;, &lt;&lt;). Recomputes column index specifications used by the display refresh logic.</td>
</tr>
<tr>
<td>ASNLPR</td>
<td>Implements the &quot;P&quot; command by redirecting display output to the user's default hard copy output device and by sending all available pages to this unit.</td>
</tr>
<tr>
<td>ASNMNP</td>
<td>Part of the assigner help subsystem; displays the values of selected Command System assigner parameter menu parameters for user inspection/reminder.</td>
</tr>
<tr>
<td>ASNMOD</td>
<td>Modifies an existing assignment record (i.e., implements the &quot;M&quot; command). Prompts the user with the existing assignments line and the dots and updates the data structures.</td>
</tr>
<tr>
<td>ASNMMOV</td>
<td>Implements the &quot;Move option of the &quot;Relocate&quot; command; moves a yard or a class-job's assignments to a different location on the display screen or to a different yard.</td>
</tr>
<tr>
<td>ASNOUI</td>
<td>Initialization routine for the DB update phase logic. Open relations and the tuple holding file and sets up</td>
</tr>
<tr>
<td>ROUTINE</td>
<td>PURPOSE</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>ASNOUT</td>
<td>Executive for the DB update phase of execution (the &quot;outbound leg&quot;). See Figure 12-7.</td>
</tr>
<tr>
<td>ASNPOP</td>
<td>Implements the &quot;'&quot; and &quot;Q&quot; commands; an interface routine between the editing and DB update code which calls the DB update executive.</td>
</tr>
<tr>
<td>ASNRPRN</td>
<td>Formats and prints an assignments buffer record, i.e. part of the contents of the valasn array in /asgn/.</td>
</tr>
<tr>
<td>ASNRPRO</td>
<td>Conditionally prints prompt text, based on the setting of the prompt flag in /asgn/. Prompt is set to true when operation is interactive, false when responses are taken from a file or some other source. Routines which prompt through asnpro are thus appropriate for use as processing utilities as well as for user interaction.</td>
</tr>
<tr>
<td>ASNRPRV</td>
<td>Implements the vertical page-up command (&quot;- and --&quot;). Computes the index numbers of the new top yard and class and stores them for reference by the display-refresh logic.</td>
</tr>
<tr>
<td>ASNRDC</td>
<td>A sophisticated terminal prompt-and-read utility. Takes prompt text and directives and returns the user response. Optionally takes response as input also and just runs it through its check logic. Checks for pop (undo) character and for help requests, and prints help from the hlpasn file if a '&quot;?'&quot; is given. Upper- or lowercases the input.</td>
</tr>
<tr>
<td>ASNREF</td>
<td>Conditionally calls asnrfh for a screen refresh: does so if prompt is true (we're interactive) and if the user has requested auto-refresh.</td>
</tr>
<tr>
<td>ASNREO</td>
<td>Re-orders the display order of assignments within a yard according the their input order, as obtained from the values in the ASNORDER field of the schedule relations. Called only when user has chosen INPUT ORDER rather than ALPHABETIC on the parameter menu.</td>
</tr>
<tr>
<td>ASNRFH</td>
<td>Performs a screen refresh, i.e. prints the current display page of the assignments buffer to the screen.</td>
</tr>
</tbody>
</table>
Table 12-2. Annotated List of Assigner Routines

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A mid-level executive which does lots of retrievals.</td>
</tr>
<tr>
<td>ASNRGT</td>
<td>Page-right. An entry point in asnlft; see its description above.</td>
</tr>
<tr>
<td>ASNRLC</td>
<td>Implements the relocate (&quot;R&quot;) command and its permutations. Either repositions a yard or class-job on the display page or make a copy of a class-job assignment record under another class-job name.</td>
</tr>
<tr>
<td>ASNSEE</td>
<td>Executive for the &quot;??&quot; (diagnostic assistance) help option.</td>
</tr>
<tr>
<td>ASNSWP</td>
<td>Inoperative.</td>
</tr>
<tr>
<td>ASNTPE</td>
<td>These three entry points of the asntpx routine form the schedule-tuple retrieval subsystem serving asndbi during the initialization phase. The routines manage a static (local) buffer which holds six schedule tuples, one per relation. The tuple images in this buffer are the next valid tuple in sequence from each relation. Asntpi loads this buffer. A call to asntpx returns the image which has the lowest key value of the six available; asntpx fills in the &quot;empty&quot; location with a new image before it returns. Asntpe just closes the schedule relations when the read is complete.</td>
</tr>
<tr>
<td>ASNTPI</td>
<td></td>
</tr>
<tr>
<td>ASNTPX</td>
<td></td>
</tr>
<tr>
<td>ASNTPU</td>
<td>A DB update utility routine used by asndbr to retrieve the next tuple from ncjodat.projj into the tuple holding buffer. Checks for both actual end-of-file and for end-of-scenario.</td>
</tr>
<tr>
<td>ASNUNL</td>
<td>Takes a schedule relation tuple image and unloads its fields into individual variables, passing their values back.</td>
</tr>
<tr>
<td>ASNVAL</td>
<td>Called when an old bufasn/cmnasn exists and user wants to use it. Flushes yards and classes which are not valid under the current invocation (i.e. not turned on in the list menus, or not even appearing if this is a different scenario).</td>
</tr>
<tr>
<td>ASNWID</td>
<td>Given a start period for screen display (i.e. the period number of the leftmost column, returns the number of columns to print and the index of the last</td>
</tr>
<tr>
<td>ROUTINE</td>
<td>PURPOSE</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ASNYRD</td>
<td>Used during display generation, prints a yard name and the &quot;-----&quot; grid lines.</td>
</tr>
<tr>
<td>ASOCMP</td>
<td>A character string comparison utility used by asndbr to decide if a given schedule key value is greater or less than another. An integer function returning -1 (key less), 0 (key equal), or +1 (key greater).</td>
</tr>
<tr>
<td>ASODEL</td>
<td>A servant of asndbr, called when asndbr thinks it has an ncjodat.projj tuple requiring deletion. This routine decides if the deletion is appropriate (might not be a valid job for this invocation, might be a no-assigner-modify tuple already processed by ashard) and does it if necessary. Asubdl is called to delete tuples in subsidiary relations.</td>
</tr>
<tr>
<td>ASPFLD</td>
<td>Reads the job description tuples for a given class-job in a given yard into the /asjd/ storage block for use by other DB update routines.</td>
</tr>
<tr>
<td>ASPFTR</td>
<td>A slave of aspfl d which just copies a job description from the /rcrd06/ buffer in which RELATE placed it into an index location in /asjd/. A simple xmit is not feasible due to the structure of /asjd/, which is in turn mandated by the requirements of the data calculation logic.</td>
</tr>
<tr>
<td>ASPRD2</td>
<td>These two routines implement the schedule date-spreading logic of the DB update phase. They operate on the /asrbuf/ linked list of ship records, changing only the adjustment-basis dates. Asprd2 was the original algorithm; it is not in the calling tree, having been replaced by the modified version now called aspred, but is functional. It was replaced as a matter of taste and might be offered as a parametrically invoked option in the future.</td>
</tr>
<tr>
<td>ASPRED</td>
<td></td>
</tr>
<tr>
<td>ASTUPF</td>
<td>Part of the DB update phase, astupf converts /asrbuf/ ship record to tuple images in the tupfil direct access holding file. It follows date spreading and precedes the actual update of ncjodat.projj.</td>
</tr>
<tr>
<td>ASUBDL</td>
<td>When a no-assigner-modify (hard-wired) tuple is deleted from the ncjodat.projj relation by the</td>
</tr>
</tbody>
</table>
Table 12-2. Annotated List of Assigner Routines

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>assigner</td>
<td>(might happen when its whole yard was deleted, for example) then any tuples dependent on it in subsidiary relations must also be deleted, much as the DBU deletes subsidiaries automatically (this is not required when assigner-modifiable tuples are deleted because it is assumed that the DBU marks any &quot;father&quot; schedules as AUTOMOD=&quot;NO&quot; when son tuples are added in the subsidiary relations). This routine does the extra deletions, learning which relations have something to be deleted by the status of the bit map in the SUBRELUMAP field of the schedule tuple about to be deleted (the DBU is assumed to maintain this field as well).</td>
</tr>
<tr>
<td>ASYCLS</td>
<td>Part of the DB update logic. Constructs a sorted list of the class-jobs for which update must be done in a given yard. Note that repair jobs can be ignored.</td>
</tr>
<tr>
<td>CKPF</td>
<td>Checks to see if a job description is available when the user adds or modifies assignments, thus providing advance warning of the necessity to go add the description using the DBU during or before the DB update phase in cases where no description has been entered. Logical function.</td>
</tr>
<tr>
<td>CMGET</td>
<td>These two entry points in the cmget routine retrieve and save /asgn/ status variables from/to the cmnasn file.</td>
</tr>
<tr>
<td>CMNSAV</td>
<td></td>
</tr>
<tr>
<td>DBASIS</td>
<td>A character function used during the initialization phase to set the name of the KEEL/DYDOCK field in field lists for schedule relation reads depending on whether the relation holds repair or new construction job data.</td>
</tr>
<tr>
<td>GETASN</td>
<td>This routine and its putasn entry point save and return records from the bufasn assignment record holding file.</td>
</tr>
<tr>
<td>INICLS</td>
<td>Does the necessary setup to establish a new class in a yard. Doesn't create the bufasn record, just sets /asgn/ values.</td>
</tr>
<tr>
<td>INIYRD</td>
<td>Does the necessary setup to establish a new yard.</td>
</tr>
</tbody>
</table>
Table 12-2. Annotated List of Assigner Routines

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCYRD</td>
<td>Does a binary search of the (sorted) array of existing yard names for a match, returning the element location of the match. An integer function.</td>
</tr>
<tr>
<td>NEWYRD</td>
<td>Adds a new yard to the list for which assignments can be made, prompting the user for the name. Ensures that the list of yards remains sorted.</td>
</tr>
<tr>
<td>NXTCLS</td>
<td>Retrieves the next tuple of interest from the schedule relation open on the given cursor. Reads records (points under some circumstances) until one matching all retrieval criteria (valid display basis date, yard name, class name, etc. are found).</td>
</tr>
<tr>
<td>NXTCLZ</td>
<td>A debugging support routine which prints a tuple and other data to ioutp. Used mainly by nxtcls.</td>
</tr>
<tr>
<td>PUTASN</td>
<td>An entry point in getasn, saves an assignments record to the given location in the bufasn file.</td>
</tr>
<tr>
<td>REMCLS</td>
<td>Removes a class (i.e. an assignment record) from a yard completely. Pulls it out of the bufasn holding file linked list and makes the necessary /asgn/ changes.</td>
</tr>
<tr>
<td>REMYRD</td>
<td>Removes an entire yard and all its assignments records.</td>
</tr>
<tr>
<td>RESTAT</td>
<td>Entry point in svstat; see below.</td>
</tr>
<tr>
<td>SCNGET</td>
<td>An entry point in CMNGET which read the first record of an existing cmnam file to see what scenario it was created under.</td>
</tr>
<tr>
<td>SVSTAT</td>
<td>This routine and its entry point restat save and restore the current values of assigner control variables before and after the &quot;P&quot; command is executed. &quot;P&quot; is implemented by using the standard logic but with alternative control settings; thus the setting must be kept and restored if the user is to be left in the same state as before &quot;P&quot; was given.</td>
</tr>
<tr>
<td>TMSTMP</td>
<td>An integer*4 function which computes a time-stamp for placement in the ASNORDER field of newly generated schedule records so that on next initialization the assigner will be able to retrieve them in the order</td>
</tr>
</tbody>
</table>
Table 12-2. Annotated List of Assigner Routines

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>TUPFRD</td>
<td>A low level routine used by asndbr to fetch the next schedule record from the tuple holding file.</td>
</tr>
<tr>
<td>VALCLS</td>
<td>Given a class name (including job type definition character, returns whether it is valid (usable) under this scenario, i.e. whether class and job type are &quot;on&quot; in list menus. A logical function.</td>
</tr>
<tr>
<td>VALYRD</td>
<td>Logical function returning .true. if the given yard name is on the valid list for this scenario and assigner invocation.</td>
</tr>
<tr>
<td>VLDLST</td>
<td>Initializes the /asnvld/ common block's lists of yards, classes, and job types that are valid for this invocation, i.e. that it's ok to work with. Does this by calls to the liston and qsortc utilities, primarily.</td>
</tr>
<tr>
<td>VLDLSZ</td>
<td>A diagnostic utility for vldlst which prints out the lists of valid names and codes after they're set up.</td>
</tr>
<tr>
<td>YCASN</td>
<td>Service routines which implement a &quot;control-Y&quot; capability to abort printout of unwanted screen refreshes. Ycasn is called when the user invokes the interrupt by pressing the control and Y keys, ycasnr can be called later to detect that the interrupt occurred, and ycasni resets the flag which remembers the interrupt. The routines do not issue the ON statement; the routine(s) using the capability must issue the ON.</td>
</tr>
</tbody>
</table>
Figure 12-14. Assigner Initialization Phase Calling Tree
Initialization is overseen almost entirely by the ASNINI routine. The operation of this routine is fairly straightforward: it reads the iniasn.sysro files, opens and/or creates other files which will be accessed via FORTRAN i/o, sets up the lists of valid class, yard, and job type names via a call to vldlst (which in turn just uses the liston utility), and reads the schedule relations to construct assignment records. At the close of initialization a refresh is done (by a call to asnr fh) in order to present the first screen page.

The non-obvious parts of the logic have to do with the schedule read, conducted by asndbi, and with what happens when there is an existing assignments buffer (bufasn file).

This latter condition will occur only when the last user to execute the assigner in the log-on group aborted during the editing or DB update phases. The user must be prompted for a desire to recover from the abort rather than starting fresh. If recovery is desired, then no DB read is required; the system is returned to its state at the time of the abort by reading the contents of the cmnasn file into memory and using the contents of the bufasn file.

There is a catch, however, embodied in the call to asnval. Since the /asnvld/ common block is not saved on disk as is /asgn/, the valid name lists must be re-initialized via a call to vldlst after an abort. However, there is no guarantee that the user has not changed the settings in the valid lists since the abort. It is therefore possible that bufasn may contain assignments records with invalid names. These are flushed by asnval so everything is consistent.

The read of data base schedules, conducted by asndbi, centers around use of asntpx and its subsidiaries to retrieve schedules from the relations. In order to make initialization
efficient, it was desirable to have the schedule tuples be read in order of yard and class-job, so that assignments records could be constructed one at a time with no necessity to go back and work on them again. At first blush this seems no problem, since one can just read from the relations on an index consisting of SCENARIO, YARD, CLASS, (NC RE) JOBT. However, up to six relations are involved, each open on a separate cursor, and a typical assignments record might have schedules resident in several relations.

It is thus necessary to construct a single virtual relation. Unlike the horizontal construction effected by a SELECT, this needs to be a vertical construction in which, apparently, the contents of all six relations are copied into a single temporary file which is then indexed by the given fields and read sequentially.

The tremendous inefficiency involved in creation of a temporary file is avoided by the following algorithm:

1) Open the relations and retrieve the first tuple from each for the given scenario on the appropriate index into a holding buffer (thus a buffer with a six-tuple capacity). This is done by asndbi.

2) Have the main read routine (asndbi) call a utility (asntpx) which does an in-memory sort of the six tuples in the buffer according to the given keys, returning the one with the lowest key value.

3) This utility in turn calls a service routine (nxtcls) to get the next tuple from the relation whose tuple was just selected, placing it in the 'vacated' location of the holding buffer. NXTCLS also ensures that the tuple is still for the given scenario, is for a valid class, yard, job type, etc., and that it is not an earlier-data-date-representation of the tuple just used. NXTCLS places a high ascii-collating sequence character in the most-significant key location of the buffer when it encounters end-of-file or end-of-scenario in a given relation, thus ensuring that the given buffer location will not emerge at the top of the in-memory sort.
This algorithm is extremely efficient and produces the desired read-ordered behavior. It employs record reads on the relations in preference to record points, doing points only when it finds it has read into a new yard (it does one point for each valid yard name).

When the algorithm detects that a given assignment record is complete (by asntpx returning a schedule for a different class-job), the given record is processed into the /asgn/-bufasn data structure just as though the user had entered it interactively.

At the close of DB reading the assignments records are reordered according to ASNORDER field value (stored in bufasn record) if the user has specified the INPUT ORDER parameter option for class-job ordering (this is done by asnreo).

12.5.2 Editing Phase Logic

Figure 12-15 presents the calling tree diagram for the editing phase. In spite of the very large number of routines mentioned on the diagram, the logic of the editing phase is fairly straightforward.

The logic is organized around response to specific user commands, with command prompting and response overseen by the asgn program unit. With the exception of the asncmd command retrieval utility, every other routine called by asgn is an executive for the processing of a particular command.

Note that asnc, called almost everywhere, is an error-reporting utility.

The paging and display-generation algorithm can be somewhat obscure because it is highly data-driven and is distributed among several routines. When the user requests a page up/down or right/left, the only processing involved is recalculation of the
Figure 12-15. Assigner Editing Phase Calling Tree Diagram
Figure 12-15. Assigner Editing Phase Calling Tree Diagram (Continued)
index numbers of the topmost yard/class-job and the leftmost column to show at the next refresh. These computations are complicated by a desire not to have yard names "hanging" at the bottom of the display with their class-jobs all appearing on the next page.

The refresh logic (asnrfh) takes these index settings as input. It prints the first few lines of the display, the header, and then retrieves bufasn records as necessary to print the assignments records. It must pay attention to proper placement of the grid lines which make it easier to read the columns, and to placement of row and column totals.

Note that most system calls for a refresh are done through the asnref routine, which consults the setting of the AUTO REFRESH parameter and just returns if the user does not want an automatic refresh after processing of each command.

Processing of the assignment-modification commands is more concentrated and linear, typically involving error checking to ensure the user has asked for something sensible, a prompt for the new assignment or modification, more error checking on the response, and posting of the result to bufasn.

12.5.3 Data Base Update Phase Logic

Figure 12-16 is a calling tree diagram for the DB update phase. Note also that Figures 12-7 and 12-8 and Section 12.1.3 summarized the structure of the phase. This section will concentrate on subtle parts of the algorithm.

The basic idea of the update is to make the bufasn assignments record structure reflect only the assignments to be updated in ncjodat.proj, generate the corresponding schedule records and place them in the relation, give the schedules as realistic a set of hull numbers as possible, and clean up by getting rid of bufasn and cmnasn.
Figure 12-16. Assigner DB Update Phase Calling Tree Diagram
12.5.3.1 Bufasn Preprocessing

This brute-force step removes any assignments in bufasn which were placed there during initialization as a result of reads of schedule records from ncjodat.histj and ncjodat.currj. Assignments originating in the rejobt relations are no problem because all repair-type jobs are ignored during the update anyway (they must be ignored because there is no way for the user to specify in limited screen space exactly which ship (class-hull-comnum) a given repair job is to be done on, and this is a crucial piece of information about a repair job).

The removal is done by re-reading the historical and current relations, using a selection rather than the asntpx-based logic of the initialization phase. For each schedule returned the corresponding bufasn assignments record is found and the proper column decremented.

12.5.3.2 Schedule Creation

The schedule creation algorithm must perform four tasks: it must generate detailed information from the summary data provided by the assignments records, it must (optionally) spread jobs over time intelligently, it must be sure not to touch any hard-wired schedule tuples in ncjodat.projj, and it must ensure that the schedules it generates are placed in ncjodat efficiently and in such a way that the relation's contents accurately represent the user's expressed desires.

Generation of the detailed schedules is done by creating a set of schedule records of number equal to the grand total showing at the bottom right of the assignments display screen MINUS all historical/current jobs and all repair-type jobs. Each record is associated with a given yard name, class name, job type name, job series type name, and contains a display date which is the first day of the period represented by the display page.
column the assignment appeared in. This is the total amount of information supplied by the user.

The rest of the information comes from the job description relation ncjdat.descj, which is queried for matches on the scenario, yard, class, job type, and job series type fields.

The full schedule records are generated by a two-pass query of ncjdat for each yard and complexity group. The first query is used only to compute the adjustment date for each schedule (i.e. the milestone named in the ADJUST BASIS parameter by the user) based on the display date. Records in this limited-information form (in the /asrbuf/ linked list) are then processed through the date spreading algorithm.

The date spreader takes all the schedule records in a given complexity group (may be limited to a given class-job), finds the first date and the last date of the interval they span, computes an average interval between ships, and recomputes their adjustment dates such they are the average interval apart. The algorithm is constrained to produce adjustment dates that fall within a single display-period duration later than the original adjustment date, thus ensuring that when a new display date is computed from the new adjustment date the given assignment would still appear in the same display page column.

The output of the date spreader is then passed through the filter of the hard-wire tuple compensator ashard. This routine reads ncjodat.projj for tuples in the current scenario with AUTOMOD field values of "NO", finds which new schedule record most closely corresponds to each such tuple, and removes the schedule record from the /asrbuf/ linked list. Hard-wired tuples can thus be ignored during the later relation update pass, since they are already compensated for.

12-57
At this point /asrbuf/ holds a number of schedule records equal to the adds/updates which will be done on ncjodat.projj for the given yard and ship complexity group. Full-scale schedule tuples are generated by a second query of ncjdat.descj and by combining its information with that in /asrbuf/, with output to tupfil.

Asndbr then supervises the update of ncjodat.projj, using tupfil's contents. It reads tupfil and asndbr concurrently, performing either a skip, delete, or update for each ncjodat tuple and an add or update for each tupfil record. Hard-wired tuples and those not in a class, yard, job type, or period of interest are skipped. Those which match the keys of the current tupfil record are updated with that record. Those with smaller keys are deleted. When the ncjodat tuple keys are larger, tupfil records are added.

12.5.3.3 Hull Number Assignment

Figure 12-17 displays the text of the newhul.rprocs RELATE EXECUTE procedure file. This file is processed by the asnhul FORTRAN routine and written to a temporary with the proper scenario key field values replacing each instance of "IMAGINATION" in the original. It is then executed.

Its goal is to assign the most realistic hull number possible to each schedule in the current scenario (not only those on the valid lists for this run) given the state of the data base. It first reads the ncjodat.histj and ncjodat.currj relations to determine the maximum hull number in each class. It then assigns this hull number to the first ship of each corresponding class in ncjodat.projj, making the assignment in the PROGVAR1 working variable, however. There are usually some projected classes which do not appear in the current and historical relations; the procedure next attempts to extract a first-ship hull number from the class name itself, e.g. "51" from DDG-51. Where even this fails, the earliest-delivery ship of the class is assigned "1" as a hull number.

12-58
Figure 12-17. Text of NEWHUL.RPROCS Execute File

1 NOTE 23
2 NOTE 28
3 NOTE 47,49,51,53,61,68,71,76,82,103,105
4 NOTE first lines of file must contain posn of lines
5 NOTE where alias will substitute in name of current scen
6 NOTE
7 NOTE THIS RELATE PROCEDURE FILE IS TO BE CALLED BY
8 NOTE ALIAS MODULE ASSIGNER.
9 NOTE
10 NOTE ITS PURPOSE IS THE UPDATING OF SHIP HULL NUMBERS TO
11 NOTE FORM A CONSISTENT, UNIQUE SERIES AFTER ASSIGNMENTS HAVE
12 NOTE BEEN CHANGED
13 NOTE
14 OPEN FILE NCJODAT.PROJJ;MODE-SHARED;PATH-PROJX
15 OPEN FILE NCJODAT.CURRJ;MODE-SHARED;PATH-CURRX
16 OPEN FILE NCJODAT.HISTJ;MODE-SHARED;PATH-HISTX
17 NOTE
18 NOTE GET MAX HULL NUMBERS FOR EACH CLASS
19 NOTE
20 SET PATH HISTX
21 SELECT SCENARIO,CLASS,HULL-SMAX(HULL BY SCENARIO,CLASS) &
22 UNIQUE BY SCENARIO,CLASS WHERE SCENARIO= &
23 "IMAGINATION"
24 COPY TO HLTMP;ERASE;RETENTION-TEMPORARY
25 SET PATH CURRX
26 SELECT SCENARIO,CLASS,HULL-SMAX(HULL BY SCENARIO,CLASS) &
27 UNIQUE BY SCENARIO,CLASS WHERE SCENARIO= &
28 "IMAGINATION"
29 COPY TO HLTMP;RETENTION-TEMPORARY
30 NOTE
31 NOTE INDEX MAX HULL NUMBERS; UNARY INDEX REQUIRED FOR
32 NOTE LET COMMANDS GIVEN THROUGH A SELECTION
33 NOTE
34 OPEN FILE HLTMP;RETENTION-TEMPORARY
35 SELECT CLASS,HULL UNIQUE BY HULL WHERE HULL-SMAX(HULL BY CLASS)
36 COPY TO HLTMP2;ERASE;RETENTION-TEMPORARY
37 OPEN FILE HLTMP2;RETENTION-TEMPORARY
38 LET HULL-HULL+1
39 CREATE INDEX BY CLASS;UNARY
40 NOTE
41 NOTE GET THE PROJ. SCHEDULE RELATION SET TO MOVE THE MAX
42 NOTE HIST/CURR
43 NOTE
44 SET PATH PROJX
45 SET INDEX SCENARIO,CLASS,DELIVERY
46 LET PROGVAR1=1 FOR SCENARIO=&
47 "IMAGINATION"
48 LET PROGVAR1-Last(PROGVAR1,CLASS) FOR SCENARIO=&
49 "IMAGINATION"
50 LET PROGVAR2=0 FOR SCENARIO=&
51 "IMAGINATION"
LET PROGVAR2=1 FOR PROGVARI=0 AND SCENARIO="IMAGINATION"

NOTE
NOTE MOVE MAX HIST/CURR HULL NUMBERS INTO FIRST TUPLE EACH CLASS
NOTE WHERE NO MAX/CURR HULL NUMBER, SET TO NUMBER IN CLASS NAME
NOTE IF THERE IS ONE
NOTE
SELECT PROJX.@,MHULL=HLTMP2.HULL WHERE PROJX.CLASS=HLTMP2.CLASS AND &
PROJX.PROGVAR2=1 AND PROJX.SCENARIO="IMAGINATION"

LET PROGVARI=MHULL
SELECT
NOTE PARSE FOR THE FIRST NUMERIC SUBSTRING OF THE CLASS FIELD
LET PROGVARI=SUBSTR(CLASS,$MATCH(CLASS,"[1-9]"), &
$MATCH($SUBSTR(CLASS,$MATCH(CLASS,"[1-9]"), "/0-9")-1) &
FOR PROGVARI=0 AND SCENARIO="IMAGINATION"
NOTE CATCH ANY LEFT AND SET THEM TO 1
LET PROGVARI=1 FOR PROGVARI=0 AND SCENARIO="IMAGINATION"
NOTE
NOTE NOW SET HULLS FOR EACH CLASS IN INCREASING ORDER
NOTE
LET PROGVARI=SRTOTAL(PROGVARI,SCENARIO,CLASS) FOR SCENARIO = &
"IMAGINATION"
NOTE
NOTE NOW MAKE SURE THAT NO SHIPS FLAGGED BY THE SCHED EDITOR
NOTE AS UNCHANGEABLE WILL HAVE THE SAME HULL AS AN UNFLAGGED SHIP
NOTE
IF PROJX.AUTOMOD="NO" AND PROJX.SCENARIO="IMAGINATION"
OPEN FILE NCJODAT.PROJJ;MODE=SHARED;PATH=PROJ2
SELECT LINE=PROJ2.$LINE,AJUNK=PROJ2.PROGVAR1, &
PROJX.HULL,PROJX.AUTOMOD &
WHERE PROJ2.PROGVAR1=PROJX.HULL AND PROJ2.SCENARIO=PROJX.SCENARIO &
AND PROJ2.CLASS=PROJX.CLASS AND PROJX.AUTOMOD="NO" &
AND PROJ2.COMNUM=PROJX.COMNUM
COPY TO HLTMP3;RET=TEMP
OPEN FILE HLTMP3;RETENTION=TEMP
CR IN BY LINE;U
SELECT PROJX.PROGVAR1,HLTMP3.LINE,FROM=HLTMP3.PROGVAR1 &
WHERE PROJX.$LINE=HLTMP3.LINE
LET PROGVARI=FROM
CLOSE PATH PROJ2
PURGE FILE HLTMP3
ENDIF
NOTE
NOTE NOW TRANSFER THE HULL NUMBERS TO THE HULL FIELD
NOTE
SET PATH PROJX
LET HULL=1000+$SRTOTAL(-1) FOR AUTOMOD="YES" AND SCENARIO="IMAGINATION"
Figure 12-17. Text of NEWHUL.RPROCS Execute File

104 LET HULL=PROGVAR1 FOR AUTOMOD="YES" AND SCENARIO= &
105 "IMAGINATION"
106 NOTE
107 NOTE CLEAN UP
108 NOTE
109 CLOSE PATH PROJX
110 CLOSE PATH CURRX
111 CLOSE PATH HISTX
112 PURGE FILE HLTMP
113 PURGE FILE HLTMP2
**
The remaining ships of each class are then assigned hull numbers in increasing order starting with the first hull for that class, still in the PROGVAR1 variable. Next the hull numbers are revised, so that there are no unary-key conflicts with hard-wired tuples, by swapping the PROGVAR1 hull numbers of hard-wired tuples with those of tuples whose PROGVAR1 value matches the actual HULL number in the hard-wired tuple. Finally the HULL field can be assigned the value of the PROGVAR1 field for all non-hard-wired tuples.

12.6 FILES USED BY THE ASSIGNER

The assigner uses a large number of files and relations. The iniasn.sysro and hlpasn.sysro permanent files are consulted. Two permanent files named bufasn and cmnasn are created in the executing user's log-on group, and purged only on successful assigner process completion. Temporary files tupfil and extmpxyz are created as working areas during DB update execution.

The ncjodat.histj, ncjodat.currj, ncjodat.projj, rejodat.histj, rejodat.currj, rejodat.projj, ncjdat.descj, valcls.mnurel, valyds.mnurel, vljtyp.mnurel, jobtyp.legals, and jstyp.legals relations are all consulted; ncjodat.projj is altered.

Assigner source code is in the .src group, in files asgna, asgnan, asgnan, asgnanc, asgnand, asgnane, asgnani, asgnanm, asgnanr, asgnant, asgnao, asgnat, asgnc, asgnr, asgny, and recomp. Object code is in corresponding files in the .obj group. Linkable object code is in asgn.obj. Program files are in asgn.prog and tasgn.prog. The GLUE, LINK, and MAKE procedure files are in asgn.merge, asgn.link, and tasgn.link respectively.

12.7 INTERFACES

The assigner depends on the integrity of the data base in several important respects. First, it assumes that the yard, class, and job type names appearing in all schedule records are
represented on the candidate lists in valyds.mnurel, valcls.mnurel, and valjtp.mnurel. This will be true as long as only the DBU is used for updating schedule relations and for updating the shdesc.miscj relation. Where it is not true, it will be as if the schedules without valid names did not exist as far as the assigner is concerned. These schedules will not be tampered with, but are unretrievable.

The assigner assumes that appropriate job description records will appear in ncjdat.descj, but is forgiving when this is not the case, allowing the user to go make needed additions in the DBU and then come back to finish his assigner run.

The assigner assumes that the DBU maintains the SUBRELUMAP field in ncjodat records, indicating the presence of data in subsidiary relations which will require deletion in the event of primary schedule deletion. Since there are no subsidiary relations supported at this time, this feature is inoperative. Care must be made in constructing future DBU screens, however. See the asubdl routine for more information.

The assigner makes the usual use of the DBIF, scenario system, and Core data swap facilities through the usual utilities.

12.8 SUBROUTINE ABSTRACTS

12-63
ASSIGNER ABSTRACTS

C*** ASSIGN---------------------------------------------
$CONTROL segment=asgn,check=3
  PROGRAM asgn

C*  *** ABSTRACT ***
C#PURPOSE executive for the Manual Assigner Module
C#AUDIT HISTORY
C  Densmore  15-Mar-83  AUTHOR
C  Densmore  19-Jul-83  Made Son Process of module
C#TYPE manual assigner routine
C#COMMON BLOCKS
Cin  scrchr  screen characters
C#CALLER Menu system through process create/activation (ASSIGN)
C#METHOD
C  executive
C#LOCAL VARIABLES
C  icmd  command index
C  isub  subcommand index
C  val  integer array giving input numeric values
C  nval  length of val array
C##
$CONTROL segment=ASGNO
C$TRACE ascday:
   INTEGER*4 FUNCTION ascday(srcday,srcod,tgtcod,pfindx)
C*
   *** FORMAL PARAMETER DECLARATIONS ***
   integer*4 srcday
   integer srcod,tgtcod,pfindx
C*
   *** ABSTRACT ***
C#PURPOSE ASsigner Calculate DAY. Use one date and
C schedule planning factors to calculate another date.
C Expects and outputs a clarified ddate.
C#AUDIT HISTORY
C MSCarey 27-jun-83 AUTHOR
C#FORMAL PARAMETERS
Cin srcday date to use as basis
Cin srcod code indicating which milestone srcday is
Cin tgtcod code indicating which milestone is desired
Cin pfindx index of job description to use in /asjd/
C#COMMON BLOCKS
Cin asoprm outbound parameters
Cin asjd job descriptions
C#CALLER various assigner outbound
C#METHOD
C Sum up the intervals between src milestone and tgt milestone
C and add that number of days to get the output.
C##
ASSIGNER ABSTRACTS

C ASCDSP

$CONTROL segment=ASGNO
C$TRACE ascdsp:

INTEGER*4 FUNCTION ascdsp(date,start,pfindx)

C* *** FORMAL PARAMETER DECLARATIONS ***

integer*4 date,start
integer pfindx

C* *** ABSTRACT ***
C#PURPOSE ASsigner Change DiSPlay date. Alters an award date
C so that its MM/DD are according to planning factors.
C#AUDIT HISTORY
C MSCarey 27-jun-83 AUTHOR
C#FORMAL PARAMETERS
Cin date current date
Cin start start date for ship; new award date must be less
Cin pfindx location of proper planning factors in /adjd/
C#COMMON BLOCKS
Cin asoprm outbound parameters
Cin asjd job description tuple images
C#CALLER asntrb
C#METHOD
C Move the desired mm and dd into the output variable from the
C planning factors. The year will be the later of the year
C of date or the year after date; ascdsp must be within
C 12 months of date.
C##
C ASCMPG ************************************************************
$CONTROL segment=ASGNO
C$TRACE asycis:
    SUBROUTINE ascmpg(mcyd3,yard,nclas,lstchr,clist,cgcode)
C *
*** FORMAL PARAMETER DECLARATIONS ***
    integer iyard,nclas,lstlen,cgcode(nclas)
    character*(lstchr) clist(nclas),yard*(mcyd3)
C *
*** ABSTRACT ***
C#PURPOSE Assigner CoMPlexity Group identifier. Takes C a list of classes and a yard reference and identifies C what complexity group each class belongs to.
C#AUDIT HISTORY
C MSCarey 19-jun-83 AUTHOR
C#FORMAL PARAMETERS
Cin iyard location of yard name in /asgn/ 
Cin nclas number of classes on list 
Cin lstchr number of chars in each class name 
Cin clist list of class names 
Cout cgcode list of complexity-group codes for each class 
C codes are arbitrary, but same for every member 
C of the same group 
C#COMMON BLOCKS 
Cin asjd job descriptions for each class 
C#CALLER asncnv 
C#METHOD
C Make up a key for each yard-class-jobtype combination, and call 
C asgdsc to get a job description tuple into the /asjd/ buffer 
C for each. Assign complexity-group codes based on the tuples. 
C If spreading mode is none or class, assign each class to a 
C separate complexity group and do not retrieve job desc tuples. 
C#
ASSIGNER ABSTRACTS

C    ASDWRN ************************************************************
$CONTROL segment=ASGNO
SUBROUTINE asdwrn(yard, class, hull)

C*    *** FORMAL PARAMETER DECLARATIONS ***

    integer hull
    character yard*8, class*10

C*    *** ABSTRACT ***

C$PURPOSE  ASsigner hardwire tuple Deletion WaRNING.

C$AUDIT HISTORY
C    MSCarey  01-jul-83 AUTHOR

C$FORMAL PARAMETERS
Cin  yard   name of yard for tuple about to be deleted
Cin  class  name of class of tuple about to be deleted
Cin  hull   hull number for tuple about to be deleted

C$COMMON BLOCKS
Cin  ioc    io unit numbers

C$CALLER various assigner outbound

C$METHOD
C    Write a message
C$
C ASGNXT **********************************************************************
$CONTROL segment=ASGNO
C$TRACE asycls:
  SUBROUTINE asgnxt(nclas,cgcode,cptr,mclchr,numpds,cused,
  1  hldcls,hldnum,nclash,nomore)
C*  *** FORMAL PARAMETER DECLARATIONS ***
  integer nclas,cgcode(nclas),cptr(nclas),nclash,mclchr,numpds
  integer hldnum(nclas)
  logical cused(nclas),nomore
  character*(mclchr) hldcls(nclas)
C*  *** ABSTRACT ***
C$PURPOSE ASsigner outbound Get Next complexity group. Moves
  bufasn lines for a related set of classes into holding buffers
C$AUIDT HISTORY
C  MSCarey  18-jun-83 AUTHOR
C$FORMAL PARAMETERS
  Cin  nclas  number of classes in yard
  Cin  cgcode  complexity-group idcode of each class
  Cin  cptr  pointer to class record in bufasn
  Cin  mclchr  max characters in class name
  Cin  numpds  max periods: length of bufasn record in effect
  Cio  cused  true if class(i) has already been processed
  Cout  hldcls  name of class(i) with job type char
  Cout  nclash  number of classes placed in hld buffers this call
  Cout  hldnum  location on clist of class i in hld
  Cout  nomore  true if all classes in this yard have been processed
C$COMMON BLOCKS
  Cio  asgn  bufasn and edit stage blocks
C$CALLER asncnv
C$METHOD
  C Look through the class list for the first unprocessed class.
  C Retrieve the record for this class and place it in hld
  C Look through the rest of the list for classes with the same
  C complexity group code and store their bufasn records.
Asigner Abstracts

C ASGPF

$CONTROL segment=ASGN0
C$TRACE asgpf:

INTEGER FUNCTION asgpf(sercod)

*** FORMAL PARAMETER DECLARATIONS ***

integer sercod

*** ABSTRACT ***

C#PURPOSE Asigner outbound Get Planning Factor index.
C Returns an index in /asjd/ of the job description
C which most closely matches the requested job series type.

C#AUDIT HISTORY
C MSCarey 27-jun-83 AUTHOR

C#FORMAL PARAMETERS
Cin sercod index indicating series type of ship (e.g. LEAD)

C#COMMON BLOCKS
Cin asnvld job type and series type reference
Cin asjd job description tuple images

C#CALLER astrb

C#METHOD
C Convert the code to a string value. Look for an exact match
C in /asjd/. If none, warn and use general purpose description.

C\"
C ASHARD

$CONTROL segment=ASGNO
C$TRACE asycls:
   SUBROUTINE ashard(mxclcr,nclash,hldcls,mxcyrd,yard)
C*
   *** FORMAL PARAMETER DECLARATIONS ***
   character*(mxclcr) hldcls(nclash), yard*(mxcyrd)
   integer mxclcr,nclash,mxcyrd
C*
   *** ABSTRACT ***
C$PURPOSE ASsigner HARDwire tuple integration routine.
C 'Deletes' one record from /rbuf/ for each hardwire tuple.
C$AUDIT HISTORY
C MSCarey 19-jun-83 AUTHOR
C$FORMAL PARAMETERS
Cin mxclcr number of characters in class name
Cin nclash number of classes being processed now
Cin hldcls names of classes being processed now
Cin mxcyrd max number of characters in yard name
Cin yard name of current yard
C$COMMON BLOCKS
Cio asrbuf 1-record buffer
Cout astfr tuple and tupfil record buffers
C$CALLER asncnv
C$METHOD
C For each class-jobt, point to the first flagged tuple in
C ncjodat using the cursor with the flags-only selection.
C Return tuples for this class until there are no more.
C For each tuple, find the record with the closest rdispd
C in rbuf which has an exact match on the job type of the
C tuple. Delete this record by removing it from the pointer
C chain. If no record can be found with a match on job type
C or with rfirst <= tupadj <= rlast, delete the tuple.
C Search for subsidiary tuples and delete them also.
ASSIGNER ABSTRACTS

C **ASSTRB** .................................................................
$CONTROL segment=ASGNO
CTRACE asycls:
SUBROUTINE ashtrb(yard,ydchr,clchr,numpds,hldcls,
    1 hldnum,nclash,mxcls,clist)

C* *** FORMAL PARAMETER DECLARATIONS ***
character*(ydchr) yard
character*(clchr) hldcls(nclash),clist(mxcls)
integer clchr,numpds,nclash
integer mxcls,hldnum(nclash),ydchr

C* *** ABSTRACT ***
C$PURPOSE ASsigner To tuple Record Buffer. Converts hld__ form of assignments to 1-line-per-ship representation useful to the data spreader and tupfil producer.
C$AUDIT HISTORY
C MSCarey 19-jun-83 AUTHOR
C$FORMAL PARAMETERS
Cin ydchr max chars in yard
Cin yard name of yard being processed
Cin clchr max chars in a class name
Cin hldcls name of class(i)
Cin nclash number of classes in hld__
Cin numpds dimension of hld__ Same as numper
Cin hldnum position of hclass on clist
Cin mxcls dimension of clist
Cin clist alphabetic list of classes in yard
C$COMMON BLOCKS
Cin asoprm general outbound params and variables
Cout asrbuf 1-ship 1-record structure: output of this routine
C$CALLER asncnv
CMETHOD
C First, run through the hld__ structure and construct asrbuf and associated pointers. Assume that the display date is to be the first day of its period in each case.
C Then process one class at a time.
C First, load planning factors for the class.
C If the display basis is award and time unit years, then convert display dates from the first to the proper date.
C Then arrive at adjust-basis dates for each ship.
C$
ASSIGNER ABSTRACTS

C ASJOI ********************************************
$CONTROL segment=ASGNO
C$TRACE asjoi;
   LOGICAL FUNCTION asjoi(string)
C* *** FORMAL PARAMETER DECLARATIONS ***
   character*12 string
C* *** ABSTRACT ***
C$PURPOSE Decides if the job type implied by the character
C in position 11 of string is one of outbound interest.
C$AUDIT HISTORY
C MSCarey 27-jun-83 AUTHOR
C$FORMAL PARAMETERS
Cin string class name with job type recog char attached
C$COMMON BLOCKS
Cin asnvlid recognition character conversion
C$CALLER asycls
C$METHOD
C Only new construction job types are of interest.
C Job types are listed as new construction or repair
C in a companion char variable to jtvld. Indexing
C first on jtvld and then on jttype gives the desired value
C**
ASSIGNER ABSTRACTS

```
C
*$CONTROL segment=asgni,check=3
   INTEGER*4 FUNCTION asnlst(fyear,fdate,idurat)
C
   INTEGER fyear,fdate,idurat
C*** ABSTRACT ***
C* PURPOSE Return equivalent of FDDATE(DPFRST,IDURAT) given /asgn/ data
C* AUDIT HISTORY
C  Densmore  28-Jul-83 AUTHOR
C* TYPE assigner date routine
C* FORMAL PARAMETERS
Cin  fyear /asgn/ fiyear - the year of the first period
Cin  fdate /asgn/ fidate - the number in year of first period
C   value depends on the value of idurat (including its
C   being undefined when idurat=1 or 2)
Cin  idurat /asgn/idurat - 1=Fyr,2=Cyr,r,3=qtr,4=month,5=week,6=day
C* COMMON BLOCKS
Cin  tddate date data type block
C* CALLER asnmnp, assigner outbound
C* METHOD
C   simple case statement. An FDDATE is performed on the result
C   as insurance.
C* LOCAL VARIABLES
C   date the result before fddate call
C   i... j... day, month numbers
C**
```
ASSIGNER ABSTRACTS

C ASNADD********asnins*********asgnd,check=3
$CONTROL segment=asgnd,check=3
SUBROUTINE asnadd(isub,val,nval)
C* *** FORMAL PARAMETER DECLARATIONS ***
C* INTEGER isub,nval,val(nval)
C* *** ABSTRACT ***
PURPOSE implements manual assigner add assignment command
HISTORY Densmore 15-Mar-83 AUTHOR
TYPE manual assigner routine
FORMAL PARAMETERS
isub subcommand index
val user-input values array
nval length of val
COMMON BLOCKS
scrchr screen characters
asgn manual assigner blocks
CALLER assign
METHOD
Checks for user and system errors. Determines if a yard is
to be added, and calls newyrd if so. Loops over period pages
and obtains input from user defining new assignment. Resets
assignment record pointers so that the new record is inserted
in proper order by ship-class name.
LOCAL VARIABLES
loc index for the new yard
start first period on this page
len number of periods on this page
last last period on this page
msg message buffer
class class buffer
before index of item before the one searched; 0 if first
item index of item searched; 0 if not present
after index of item after the one searched; 0 if last
valbuf values buffer
codbuf codes buffer
t,nil .true.,.false. -- easier to see
xsec cross section sum
ifree pointer to next item in free chain (after freptr)
look class index for which to look; 0 if Add command
mval maximum val index expected [1..2]
clip IF asnadd THEN t ELSE (nil & Assert asnins)
beyond name of max val index expected [yard,class]

12-75
SUBROUTINE asnalo

*** ABSTRACT ***
PURPOSE allocates an assignments buffer; places it on free chain

HISTORY
Densmore 21-Mar-83 AUTHOR

TYPE manual assigner routine

COMMON BLOCKS
asgn assigner data block

CALLER asnadd

METHOD
Two variables store the free buffer records status: nvruse and freptr. nvruse is the record index of the first never-used direct access buffer record. freptr is a pointer (a record index) to the head of a list of assignment buffer records which are free for use (and probably got there by being used and then freed).

LOCAL error true if an I/O error occurred


SUBROUTINE asnl(t(len,valbuf,codbuf,values,codes)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER len,valbuf(len),codbuf(len),values(len),codes(len)

*** ABSTRACT ***

C REPRESENTS ALTERS BUFFER VALUES/CODES ACCORDING TO BUFFER CODES
SO THAT MODIFICATION IS AFFECTED.

C AUDIT HISTORY
C Densmore 17-Mar-83 AUTHOR
C TYPE manual assigner routine
C FORMAL PARAMETERS
CIN len number of periods in question
CIN/OUT valbuf new values input...undefined if corresponding
C codbuf new codes input...cundef means change (valbuf,codbuf)
C to (0,0)...0 means change pair to (values,codes ),
C that is, the old values.
CIN values the old values
CIN codes the old codes
C CALLER asnmmod
C

ASSIGNER ABSTRACTS
ASSIGNER ABSTRACTS

C ASNNAMM*******************************************************************************
$CONTROL segment=asgnr,check=3
  SUBROUTINE asnnam(val,nval)
C*          *** FORMAL PARAMETER DECLARATIONS ***
C*          INTEGER nval, val(nval)
C*          *** ABSTRACT ***
P*PURPOSE Allows modification of a yard or class name
CAUDIT HISTORY
C Oensmore 11-Jul-83 AUTHOR
C*TYPE assigner routine
C*FORMAL PARAMETERS
Cin     val input numeric parameters
Cin     nval number of parameters
C*COMMON BLOCKS
Cin/out asgn assigner data block
C*CALLER asnmod
C*METHOD
C Determines whether yard or class name is being modified,
C checks that line is being displayed, solicits new name,
C checks it, and makes the change.
C*LOCAL VARIABLES
C buffer character variable storing new name
C old old value
C mcls true if class name being modified (else yard name)
C msg message buffer
C icsl class number
C iyrd yard number
C leno/b character lengths
C item pointer to appropriate class
C**
SUBROUTINE asncal(yard)
INTEGER yard

PURPOSE asks that next refresh center around mentioned yard

AUTHOR Densmore 1 April 1983
ASSIGNER ABSTRACTS

C SUBROUTINE asndch(bdate,nbd,tdate,prioty,ntd,ndate,nnd,mnd)

C* *** FORMAL PARAMETER DECLARATIONS ***

INTEGER nbd,ntd,nnd,mnd,prioty(ntd)

---DOATE bdate(nbd),tdate(ntd),ndate(mnd):

INTEGER*4 bdate(nbd),tdate(ntd),ndate(mnd)

C* *** ABSTRACT ***

C* PURPOSE Accepts Buffer_DATEs, Tuples_DATEs, outputs instructions
C* for converting tdates into new dates in New_DATEs.

C* AUDIT HISTORY
C* Densmore 27-Apr-83 AUTHOR

C* TYPE Manual Assigner routine

C* FORMAL PARAMETERS

Cin bdate Sorted Dates derived from the buffer assignment for a
C* particular yard/class/period; presumably these
dates are evenly spaced over the period.

Cin nbd number of bdates

Cin tdate Sorted Dates obtained from database corresponding to
C* the old assignments in this yard/class/period.
C* These are presumably obtained directly from the
C* tuples in the database.

Cin prioty A priority for the old dates. Currently, there are
C* only two values: 0="software", 1="hardwire"

Cin ntd The result array. If ndate(i) is nonzero, then
C* the tuple corresponding to i should be updated
to contain the date ndate(i). If ndate(i) is
C* zero, that tuple should be removed from the
C* database. If nnd>ntd, then the tuples for
C* which i>ntd should be added to the database. If
C* ntd>nnd, then there are exactly (ntd-nnd) ndate
C* values which are zero. Note result not sorted.
C* *** It is assumed that all valid dates are > 0.

Cout ndate Length of ndate

Cin mnd Maximum allowable length for ndate

C* COMMON BLOCKS

Cin tddate DOATE data type block

Cin lprnts diagnostics

C* METHOD

C* The method is a three step process. First, recall that the
C* total number of dates we wish to be left with is nbd. If ntd>nbd,
C* then the first step is to mark for deletion the (ntd-nbd) latest
C* software tuples. If more must be deleted than exist, then the
C* algorithm begins deleting the latest hardware tuples.

C* The second step is to loop through and mark all the hardware
C* tuples that remain to be kept. Let the number of such tuples be
C* given by nhard. Then at this point (nbd-nhard) dates remain to
Now, for each hardwire tuple being kept, exactly one softwire date in bdate must be ignored. The one ignored at each step is the one "closest" to the hardwire tuple, timewise. (Datatype DDATE function DCLOSR is used.) The dates remaining after the ones to be ignored are marked are placed in ndate at the appropriate spots, and processing is complete.

**C** LOCAL VARIABLES

- **unmark** an unmarked state variable of type DDATE
- **delete** a marked state meaning delete this tuple
- **tooqrt** a diagnostic state meaning index > maximum
- **d** delete-ndate index
- **h** hardwire-ndate set index
- **a** arbitrary ndate index
- **hard** boolean indicating now doing hardwire deletions
- **hipri** stmt function true when hi-priority is on
- **nhd** number of hardwire dates to be kept
- **ignore** flags indicating that the corresponding Buffer-DATE should not be used to set NDATE values
- **closest** buffer-DATE index such that:
  
  \[ BDATE(closest) \geq NDATE(h), \]
  
  but
  
  \[ BDATE(closest-1) < NDATE(h); \]
  
  Overflow condition indicated by closest=0 or nbd+1
- **low** nearest lower unignored BDATE to BDATE(closest)
- **high** nearest higher unignored BDATE to BDATE(closest)
- **set** either low or high value
**SECTION**

*CONTROL segment=asgnd, check=3
SUBROUTINE asnc6k(ival)

C* *** FORMAL PARAMETER DECLARATIONS ***
C* INTEGRAL ival

C* *** ABSTRACT ***
C* PURPOSE EXECUTES DIAGNOSTICT ACTION NUMERED ival
C* AUDIT HISTORY
C* Densmore 28-Mar-83 AUTHOR
C* TYPE manual assigner utility
C* FORMAL PARAMETERS
C in ival diagnostic action index
C COMMON BLOCKS
C in asgn assigner data block
C CALLER asnee
C**
SUBROUTINE asncln

*** FORMAL PARAMETER DECLARATIONS ***

*** ABSTRACT ***

ASigNer CLeaN.

Removes any instances of historical or current jobs from bufasn to ensure that no duplication in the data base occurs.

C* PURPOSE

C* AUDIT HISTORY

MSCarey 16-jun-83 AUTHOR

C* FORMAL PARAMETERS

C* none

C* CALLER

asnout

C* METHOD

Perform a selection which will cause only those tuples of interest to be returned. For each such tuple, figure out which cell of bufasn it was put in and decrement that cell. Check the cell value to see if it is now < 0, and warn the user that current/historical assignments may not be changed with the assigner.

C* LOCAL VARIABLES

C* cursor (1) for ncjodat.histj, (2) for .currj selections

C**

12-83
C     ASNL**R-----------------------------------------------
$CONTROL segment=asgnd,check=3
   SUBROUTINE asncnr
C*
*** ABSTRACT ***
C* PURPOSE clears screen for assigner module
C* AUDIT HISTORY
C Densmore 17-Mar-83 AUTHOR
C*TYPE manual assigner routine
C*COMMON BLOCKS
Cin/out  asign assigner data block
C*CALLER Several ASN routines
C##
SUBROUTINE asncmd(icmd,isub,val,nval,mval)
      INTEGER icmd,isub,nval,mval,val(mval)

C*     *** ABSTRACT ***

C* PURPOSE Read user command and decode it

C* AUDIT HISTORY
C  Den3mor  17-Mar-83  AUTHOR
C* TYPE manual assigner routine
C* FORMAL PARAMETERS
Cout icmd major command index (tied to /scrchr/)
Cout isub subcommand index (/scrchr/)
Cout val values array -- set of integers separated
  on input by periods here.
Cout nval length of val
Cin mval maximum length of val
C* COMMON BLOCKS
Cin/out asgn assigner data block
Cin scrchr screen characters
C* CALLER assign
C* METHOD
C <Command-string> ::= <Command> <Subcommand> [ <Num> <Delim> <Num> ]
C <Command> ::= (one of the /scrchr/ characters)
C <Subcommand> ::= <Command> ; <Null>
C <Num> ::= (an integer)
C
C 200

CBLANK BLANK / BLANK DIGIT ,->DOT or COMMA-, BLANK
C ^ ^ ,->EOL< ^ ^ ^ ^ ^ ^ / BLANK C ::
^ / ^ / ^ / ^ / ^ / ^ / ^ /
C v : v / v /
C 100-->CMD-->110-->SubCMD-->120-->DIGIT-->130-->BLANK-->140-->DOT-->150
C  ^ ^ ^ ^ ^ ^ ^ ^ ^
C 110:---DIGIT------------------>' ; C

C Out of every state is an implicit "ANY "<--------DIGIT<--------
C OTHER CHARACTER" whose vector leads to
C an error state, which returns to $10.

C* LOCAL VARIABLES
C csave command indices corresponding to commands
C for which saves (CALL cmnsav) must be done
C
C##
SUBROUTINE asncnv(numpds)

C*** FORMAL PARAMETER DECLARATIONS ***

*** ABSTRACT ***

C* PURPOSE
ASSigner CoNVert. Converts bufasn into tupfil.
C* Basically, input is as displayed by assigner, output
C* is tuples as found in the data base.

C* AUDIT HISTORY
MSCarey 16-jun-83 AUTHOR

C* FORMAL PARAMETERS

Cin_numpds number of periods; same as numper
C* CALLER

C* METHOD

C* Loop over yards in bufasn.
C (1) For each yard, make a list of classes and bufasn record
C* numbers for those which have non-zero assignments. Sort the list.
C (2) Search the list for classes belonging to the same spreading-group
C* (menu option complexity-group). Load all classes belonging to
C* the same group into a holding buffer, and mark them as processed
C* on the list. Unless the user has chosen the complexity-group
C* spreading option, this function will return one class at a time
C* until a yard is exhausted. Load schedule planning factors
C* for each class of interest into a companion buffer.
C (3) For each group, loop over columns in the holding buffer.
C* For each column,
C loop over rows, decrementing each cell by one and constructing
C a record for the tuple buffer. Attention is paid to job series
C type special characters here. The dates put into the tuple
C buffer should be adjustment-basis dates, which are arrived at
C from the display-basis dates using the planning factors. These
C dates are therefore the earliest dates allowable after the
C date-spreading process if display-date integrity
C is to be maintained.
C NOTE EXCEPTION: if display basis is awards, and time units are
C years, then take the day of award in a given year from the
C job desc record, rather than assuming it to be the first day
C of the year. Then calculate the adjdat's as usual.
C (4) If the user has specified date-spreading in his calling
C parameters, conduct the date spreading here.
C First spread the dates in the tuple buffer completely evenly,
C also assigning a first allowable and last allowable date for each
C Then go through EACH CLASS sequentially: on finding a date
C earlier than its
C limit, add the amount needed to bring it up to the limit to
C every item in the class. On finding a date later than its
C limit, look backwards and forwards for all dates with the same
limits (i.e. from the same display period) and spread
within that period evenly. Now recombine the classes and check
for instances of identical adjdat's. If found, calculate the
mean interval for the nearest # ships and add/subtract half this
amount to each IF this will not violate the period limits.
Decide which to add or subtract to depending on which is closer
to upper/lower period limits.

(5) Now integrate in any hardwire tuples in the data base.
SELECT @ BY yard, class WHERE scenario=_curen_ and flag="up"
For each class now in tupbuf, calc to first matching tuple;
if any, then find tupbuf records with closest adjdat, and mark
them gone;

(6) Now get the rest of the dates for softwires, based on the spread
adjdat's. Also construct the rest of each tuple and put it into
tupfil. Use standard planning factors except for award:
if adjbasis is award, use as calculated;
IF tunit is years then calc from factors and set to next-earliest
desc-date
ELSE calc from factors and set to first of period it's in, based
on time units

(7) On end of busasn for this yard, update data base.
LOCAL VARIABLES
clist list of classes in current yard
cgcode complexity-group each class is in
cptr pointer to bufasn record for each class
nclas number of classes in current yard
cused true if a class has been processed
hldcls class name of each class in hld__
hldval per-period assignments for each class in a comp-grp
hldcod per-period codes for each class in a comp-grp
nclash number of classes in the hld__ buffers now
SUBROUTINE asncod(newcod, oldcod)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER oldcod, newcod

*** ABSTRACT ***

PURPOSE conditionally sets old job series code id for asndbi

AUDIT HISTORY

Densmore 24-Jun-83 AUTHOR

FORMAL PARAMETERS

Cin newcod tcode from the latest tuple
Cio oldcod the relevant code from this scen/yr/cld/period

12-88
SUBROUTINE asncpy(val, nval, move, succes)

INTEGER nval, val(nval)
LOGICAL succes, move

CSPURPOSE Makes new copies of yards or classes

C$COMMON BLOCKS
Cin/out asgn assigner data block

C$CALLER asnrlc

C$METHOD
C Split into two parts: a copy-yard part and a copy-class part.
C The copy-yard part is implemented in yrdcpy.
C The copy-class part does routine checks, loops down to the
C fromclass data, and then holds it. Then asnfnd is called; the
C remainder is similar to the code in asnadd -- hard to make into
C a subroutine because variables must be held in limbo around
C sections of code which are different for the two applications.

C$LOCAL VARIABLES
C buffer holds old assignment
C i do index
C from old or from yard location
C loc new or to yard location
C ifree pointer to free chain after class copy
C class holds old assignment class name
C msg holds messages to be sent to user via asnpro-mpt
C firstm n/o /l/o/n/g/e/r/ /u/s/e/d

12-89
**ASNDBI**

$CONTROL segment=asgni,check=3

SUBROUTINE asndbi

*** ABSTRACT ***

**C**\*PURPOSE** Recovers ASN data from relations

**C**\*AUDIT HISTORY

C  Densmore  07-Apr-83  AUTHOR
C  Densmore  06-May-83  To begin looping thru DB

**C**\*TYPE  manual assigner utility

**C**\*COMMON BLOCKS

Cin  asgn  assigner data block

**C**\*CALLER asmini

**C**\*METHOD

**C**\*
ASSIGNER ABSTRACTS

C ASNDBR

$CONTROL segment=ASGNO
C$TRACE asndbr:
C

SUBROUTINE asndbr(numyds,mcyds,yard,nclas,tupfst,lstcal,
clist,mccls)
C* *** FORMAL PARAMETER DECLARATIONS ***
.
C* integer numyds,mcyds,yard,nclas,tupfst(nclas)
C* character*(mccls) yard
C* character*(mccls) clist(nclas)
C* logical lstcal
C* *** ABSTRACT ***
C

C$PURPOSE ASsigNer Data Base tuple Replacement routine.
C Updates the tuples in the data base for the given yard
C using the assignments implied by the current state of
C bufasn.
C$AUDIT HISTORY
C
MSCarey 03-jul-83 AUTHOR
C$FORMAL PARAMETERS

Cin numyds number of yards in bufasn; if zero, clean bufasn
Cin mcyds length of yard
Cin yard yard to be updated
Cin nclas number of class-jobtypes in tupfil
Cin tupfst pointer to first tupfil record for each clas-jobt
Cin lstcal true if this is the last call to asndbr; in this
C case, processing should go on until the end of tuples
C for this scenario so that any trailing deleted
C assignments are caught.
Cin clist list of classes found in bufasn for this yard
Cin mccls length of clist class names.
C$COMMON BLOCKS
Cin asoprm outbound parameters
Cin asnocr outbound cursors
Clo astfr buffer for relation and tupfil records
C$CALLER asncnv
C$METHOD
C

Basically, series of cases. There is always a current tuple
and a current tupfil record. The actions which may be taken
are to update the tuple using the record, to add the record
to the relation, to skip to the next tuple, and to delete the
tuple. Which is appropriate depends on a comparison of the
values of the yard, class, and jobtype fields in the tuple and
tupfil holding buffers. Both tuples and tupfil records are
assumed to arrive in their holding areas sorted by yard, class,
and jobtype.
C$LOCAL VARIABLES
C
eofil true if no more records in tupfil
C lstyrd yard name of tupfil record prev to current record
C lstcls class "

12-91
C lstjob job typ "
C next record number in tupfil of next record
C iclas class-job type on clist now being processed
SUBROUTINE asndel(isub,val,nval)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER isub,nval,val(nval)

*** ABSTRACT ***

PURPOSE implements manual assigner delete assignment command

HISTORY

Densmore 15-Mar-83 AUTHOR

TYPE manual assigner routine

FORMAL PARAMETERS

isub subcommand index

val user-input values array

nval length of val

COMMON BLOCKS

scrchr screen characters

asgn manual assigner blocks

CALLER assign

METHOD

If an assignment index is given, deletes that assignment. Checks
first to see if it is the last. If so or if no assignment given,
the yard is deleted after prompting to make sure.

LOCAL VARIABLES

msg message buffer for asnpro/asnec

loc yard index

before pointer to assignment record before delete item

item pointer to delete item

after pointer to item's successor

next do index
ASSIGNER ABSTRACTS

C ASNDOT******************************************
$CONTROL segment=asgn,check=3
SUBROUTINE asndot(getcls,alwdel,len,class,values,codes)
C*** FORMAL PARAMETER DECLARATIONS ***
LOGICAL getcls,alwdel
INTEGER len,values(len),codes(len)
C*** CHARACTER*12 class -- 12==mccls (given in /asgn/)
C*** ABSTRACT ***
PURPOSE Types the dots for prompting of assignments input,
and accepts and verifies the input. Output to values
and codes array in decoded form.
AUDIT HISTORY
Densmore 17-Mar-83 AUTHOR
TYPE manual assigner routine
FORMAL PARAMETERS
Cin getcls .T. if shipclass should be prompted for and
Cin alwdel .T. if DELetion of assignments should be allowed
Cin len number of periods over which input is expected
Cout class output shipclass, char=mccls...not output unless
Cout values output values
Cout codes output codes
COMMON BLOCKS
Cin/out asgn assigner data block
CALLER asndadd,asndmod
METHOD
Prompts for add and modify commands, via dots which delineate
proper placement of each number/code sequence (and also the
class name, if requested). for each period, the following
sequence of characters (3 for each period) are valid: the
first character must always be a blank (CU means cundef):

<table>
<thead>
<tr>
<th>Char-1</th>
<th>Char-2</th>
<th>Char-3</th>
<th>Value</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>blank</td>
<td>blank</td>
<td>blank</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>*</td>
<td>zero</td>
<td>Digit3</td>
<td>D3</td>
<td>cdidef</td>
</tr>
<tr>
<td>*</td>
<td>blank</td>
<td>0</td>
<td>CU IF alwdel ELSE 0</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>zero</td>
<td>0</td>
<td>CU IF alwdel ELSE 0</td>
<td></td>
</tr>
<tr>
<td>Code2</td>
<td>Digit3</td>
<td>D3</td>
<td>C2</td>
<td></td>
</tr>
<tr>
<td>Digit2</td>
<td>blank</td>
<td>D2</td>
<td>cdidef</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>Digit3</td>
<td>10*D2+D3</td>
<td>cdidef</td>
<td></td>
</tr>
</tbody>
</table>

LOCAL VARIABLES
m/c/c/l/s/l -- variable now in common
ccdots prompt for class name (either blank, or dots)
dot2 prompt for each period assignment input ("...")
number the string of digits 0 through 9
buffer location where user's input is accepted
b1,b2,b3 each character of a given assignment entry
iper the current period index (1..len)
nchar the number of blanks before first prompt dot
C SUBROUTINE asndwn(toploc,topind)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER toploc,topind

*** ABSTRACT ***

C* PURPOSE performs "DOWN" (following page) command
C* AUDIT HISTORY
C Densmore 17-Mar-83 AUTHOR
C TYPE manual assigner routine
C* COMMON BLOCKS
C in/out asgn assigner data
C CALLER asnfol,asnrfh
C METHOD
C If BOTTOM, sets low limits to last assignment index, then
C computes upper limits. If DOWN, sets upper limits to former
C lower limits and recomputes lower limits; if it crosses the
C bottom then BOTTOM command is performed.
C
C Also, this routine always recomputes the page number npagev.
C
ASSIGNER ABSTRACTS

C    ASSIGNER*************************************************************************
$CONTROL segment=asgned,check=3
    SUBROUTINE as nec(what,value,type,text)
C*    *** FORMAL PARAMETER DECLARATIONS ***
    INTEGER what,value,type
    CHARACTER*255 text
C*
C* PURPOSE Assigner Error in Command reporter
C* AUDIT HISTORY
C    Densmore 17-Mar-83 AUTHOR
C* TYPE manual assigner routine
C* FORMAL PARAMETERS
Cin  what  what was in error: refers to explicit errors in the input user command, rather than to logic errors caused by faulty command order or intent.
C    0=none
C    1=major command index
C    2=subcommand index
C    -N=val array element
Cin  value Value code for the item in error: the value of the index or the value of the val array element.
C    Undefined if what=0.
Cin  type Error type: 0=some external system error or error about which no further information should be printed.
C    1=explicit user-input command error. 2=user stressed assigner up against some limitation (array bound,etc.)
Cin  text Delimited text string which should be printed to assist in describing the cause of the problem.
C*COMMON BLOCKS
Cin/out asgn assigner data block
C*CALLER various ASN routines
C##
SUBROUTINE asnend

C*    *** ABSTRACT ***
C*PURPOSE Prepares ASGN for STOP (End of process)
C*AUDIT HISTORY
C    Densmore    19-Aug-83 AUTHOR
C*TYPE    assigner
C*COMMON BLOCKS
Cin      asgn
C*CALLER  asgn
C*METHOD
C    Currently just closes the files which are open.
C**
SUBROUTINE asneoi

*** ABSTRACT ***

C* PURPOSE reverts I/O switches upon EOF on 'inasn' in /asgn/

C* AUDIT HISTORY
C Densmore 17-Mar-83 AUTHOR

C* TYPE manual assigner route

C* COMMON BLOCKS
C in/out asgn assigner data block
C in/out loc input/output variables
C* CALLER asncmd,asndot,newyrd,remyrd

C*
SUBROUTINE asnfnd(loc, class, look, before, item, after)

**FORMAL PARAMETER DECLARATIONS**

INTEGER loc, look, before, item, after

CHARACTER*12 class

ABSTRACT

Purpose: Locates a particular class within a given yard

History: Densmore 17-Mar-83

Type: manual assigner routine

Formal Parameters:

- **loc**: yard index
- **class**: name of class, char*12
- **look**: if location of new cls is by index, look is nonzero and gives that index; means "ins before class LOOK"
- **before**: pointer to assignment record preceding item
- **item**: pointer to assignment record with given class
- **after**: pointer to assignment record succeeding item

Common Blocks:

- **asgn**: assigner data block

Callers:

- **asnadd**
SUBROUTINE asnfol(isub,val,nval)

* *** FORMAL PARAMETER DECLARATIONS ***

INTEGER isub,nval,val(nval)

* *** ABSTRACT ***

C* PURPOSE implements manual assigner follow (down) page command

C* AUDIT HISTORY

C* TYPE manual assigner routine

C*PARAMETERS

Cin isub subcommand index

Cin val user-input values array

Cin nval length of val

C*COMMON BLOCKS

Cin scrcrn screen characters

Cin/out asgn manual assigner blocks

C*CALLER assign

C**
C ASNGOQ .................................................................
$CONTROL segment=ASGNO
C$TRACE asngoq:
   LOGICAL FUNCTION asngoq(idum)
C* *** FORMAL PARAMETER DECLARATIONS ***
C*   integer idum
C* *** ABSTRACT ***
C$PURPOSE Assigner Get Outbound Quit response. Prompts
C  user before assigner outbound processing commences to
C  see if results of session should just be thrown away.
C$AUDIT HISTORY
C   MSCarey  27-jun-83  AUTHOR
C$FORMAL PARAMETERS
Cin   idum    dummy required by FORTRAN
C$COMMON BLOCKS
Cin   io    io unit assignments
C$CALLER asnout
C$METHOD
C   Print an explanatory message and call yesno
C$LOCAL VARIABLES
C   none
C**
SUBROUTINE asnhlp(isub,val,nval)

C* *** FORMAL PARAMETER DECLARATIONS ***
  INTEGER isub,nval,val(nval)

*** ABSTRACT ***
C* PURPOSE retrieves help text for ASN
C* AUDIT HISTORY
C   Densmore  17-Mar-83 AUTHOR
C*TYPE manual assigner routine
C*FORMAL PARAMETERS
Cin  isub   subcommand index
Cin  val    values array
Cin  nval   length of val
C*COMMON BLOCKS
Cin/out asgn  assigner data block
C*CALLER assign
C**
SUBROUTINE asnhul

*** FORMAL PARAMETER DECLARATIONS ***

*** ABSTRACT ***

ASsigner HULL number reset. Makes sure each ship in the current scenario has a unique and reasonable hull number.

MSCarey 20-jun-83 AUTHOR

none

none

asnout

Get the maximum hull number for each class in nzjoprj from the historical/current relations. Execute command file NEWHUL.RPROCS to do this, first writing the current scenario key into it for each of the three schedule files.

none
**Abstract**

Manual Assigner Initialization Routine

**Purpose**

Initialization is divided into three steps. The first is to set up hardwire values. Next, certain variables are input from a file (FILE04). Lastly, the DESC and JOB relations are consulted to obtain assigner data block initial values.
C ASNLBL

$CONTROL segment=asgnl,check=3

SUBROUTINE asnlbl(ydllbl,.clsllbl,total,start)

*** FORMAL PARAMETER DECLARATIONS ***

LOGICAL ydllbl,.clsllbl,total
INTEGER start

*** ABSTRACT ***

C#PURPOSE print top two rows of assigner display -- period rows
C#AUDIT HISTORY
C Densmore 17-Mar-83 AUTHOR
C#TYPE manual assigner routine
C#FORMAL PARAMETERS
Cin ydllbl .T. if "yard" should be printed
Cin clsllbl .T. if "shipclass" should be printed
Cin total .T. if total for the rows will be printed
Cin start index of first period to be printed
C#COMMON BLOCKS
Cin/out asgn assigner data block
C#CALLER asnadd,asnm,asnrfh
C#
C ASNLLEV *****************************************************
$CONTROL segment=ASGNO
   SUBROUTINE asnllev(quit)
C* *** FORMAL PARAMETER DECLARATIONS ***
   logical quit
C* *** ABSTRACT ***
C* PURPOSE  ASsigner LEaVe. Cleans up assigner files and
            relations before return to the menu system.
C* AUDIT HISTORY
C  MSCarey      22-jun-83 AUTHOR
C* COMMON BLOCKS
Cio  asncr   cursors for the assigner
Cin  asoprm  assigner outbound parameters
C* CALLER asnout
C* METHOD
C  Many calls to rvclso and filcls
C**
C ASNLFT-------------------------------------------------------------
$CONTROL segment=asgnd,check=3
   SUBROUTINE asnlft(isub,val,nval)
C*                      *** FORMAL PARAMETER DECLARATIONS ***
C*     INTEGER isub,nval,val(nval)
C*                      *** ABSTRACT ***
C#PURPOSE implements manual assigner left page command
C#AUDIT HISTORY
C Densmore 15-Mar-83 AUTHOR
C#TYPE manual assigner routine
C#FORMAL PARAMETERS
Cin  isub subcommand index
Cin  val user-input values array
Cin  nval length of val
C#COMMON BLOCKS
Cin  scrchr screen characters
Cin/out asgn manual assigner blocks
C#CALLER assign
C#METHOD
C Determines if edge command is needed; pages left/right
C accordingly.
C##
SUBROUTINE asnlpr(isub,val,nval)
C* *** FORMAL PARAMETER DECLARATIONS ***
C* INTEGER isub,val,nval
C* *** ABSTRACT ***
C* PURPOSE PRINT command executive
C* AUDIT HISTORY
C* Densmore 26-May-83 AUTHOR
C* TYPE Manual assigner routine
C* FORMAL PARAMETERS
C* isub subcommand index
C* val user-input values array
C* nval length of val
C* COMMON BLOCKS
C* asgn manual assigner blocks
C* CALLER assign
C* METHOD
C* Determines validity of input. Given valid values, it then
C* prints to a (reset) outasn via asnrff, manipulating the
C* top/low yards/indexes as required to get whole yards on
C* a page.
C* LOCAL VARIABLES
C* start first yard to print
C* stop last yard to print
C* ydfrst first yard to be printed on this page
C* ixfrst first index to be printed on this page
C  ASNMOD******************************************************
$CONTROL segment=asgn,d,check=3
   SUBROUTINE asnm(bsub,bsub,bsub(bsub))
C*   *** FORMAL PARAMETER DECLARATIONS ***
   INTEGER bsub,bsub,bsub(bsub)
C*   *** ABSTRACT ***
C* PURPOSE implements manual assigner modify assignment command
C* AUDIT HISTORY
C   Densmore 15-Mar-83 AUTHOR
C* TYPE manual assigner routine
C*FORMAL PARAMETERS
Cin  bsub    subcommand index
Cin  bsub    user-input values array
Cin  bsub    length of bsub
C*COMMON BLOCKS
Cin  scrcr    screen characters
Cin/out asgn    manual assigner blocks
C*CALLER assign
C*METHOD
C   Locates yard/assignment and loops over periods
C*LOCAL VARIABLES
C   msg     message buffer for prompts
C   valbuf  values buffer
C   codbuf  codes buffer
C   diff    difference between buffer and old valasn page
C   loc     yard location index
C   idx     assignment index
C   change  SUM(diff[i])
C   start   starting period on current page
C   len     number of periods on current page
C   last    last period on current page
C   item    pointer to assignments buffer being modified
C   tnil    easier to read than TRUE/FALSE
C   begin   T on first iteration of $60, F otherwise

12-110
SUBROUTINE asnmov(val, nval)

*** FORMAL PARAMETER DECLARATIONS ***
INTEGER nval, val(nval)

*** ABSTRACT ***

Implements Relocate Move command

Densmore 08-Jul-83 AUTHOR
assigner routine
numeric values
number of values
assigner data block

See inline comments
Implemented via copy followed by delete
SUBROUTINE asnout

*** FORMAL PARAMETER DECLARATIONS ***

*** ABSTRACT ***

PURPOSE
ASsigner OUTbound. Converts edit-able assigner data base to RELATE data base format.

AUDIT HISTORY
MSCarey 16-jun-83 AUTHOR

FORMAL PARAMETERS
none

COMMON BLOCKS
asoprm  general outbound

CALLER assign

METHOD

(1) Prompt the user to see if he wants to exit without saving the results of his assigner session. Intended to allow users to peruse assignments without having to pay the execution-time price of the outbound leg if they have made no changes. Remind the user at this point that no changes to repair or historical/current assignments will be saved.

(2) Remove from bufasn any instances of historical/current assignments, where an instance is defined as an increment in any bufasn cell.

(3) Convert bufasn into a set of tuples, one per ship, which is merged with any flagged ('hardwire') tuples from the RELATE data base. Take care of uniform spreading of construction schedule dates here.

Make an update pass on the data base so that the set of tuples in the data base is identical to that generated in step 3. Note that subsidiary tuples must be purged for purged hardwire tuples.

(5) Make another update pass at the data base to arrive at a set of consistent hull numbers, unique for each ship. 'Hardwire' tuples retain their own hull numbers.

(6) Clean up and exit.

LOCAL VARIABLES

quit true if user wants to throw away bufasn
ASSIGNER ABSTRACTS

C   ASNPOP************************************
$CONTROL segment=asgnd,check=3
   SUBROUTINE asnpop(isub,val,nval)
C*   *** FORMAL PARAMETER DECLARATIONS ***
    INTEGER isub,nval,val(nval)
C*   *** ABSTRACT ***
C&PURPOSE performs termination code when exit requested from ASN
C&AUDIT HISTORY
C    Densmore  17-Mar-83 AUTHOR
C#TYPE manual assigner routine
C#FORMAL PARAMETERS
Cin   isub    subcommand index
Cin   val     input values array
Cin   nval    length of val
C#COMMON BLOCKS
Cin/out asgn    assigner data block
C#CALLER assign
C#

12-113
SUBROUTINE asnprn(typcls, total, start, idx)

** FORMAL PARAMETER DECLARATIONS **

INTEGER start, idx
LOGICAL typcls, total

*** ABSTRACT ***

** PURPOSE Prints a line from the buffer during display formatting **

** AUDIT HISTORY **

Densmore 17-Mar-83 AUTHOR

** TYPE manual assigner routine **

** FORMAL PARAMETERS **

Typcls .T. if shipclass name should be printed
Total .T. if total for row should be printed
Start index of first period on the row
Idx assignment index (for printing purposes only)

** COMMON BLOCKS **

Asgn assigner data block

** CALLER asnmod, asnrsp **
ASSIGNER ABSTRACTS

C ASNPRO

*CONTROL segment=asgn,check=3
SUBROUTINE asnpro(text)

C* *** FORMAL PARAMETER DECLARATIONS ***
CHARACTER*255 text
C* *** ABSTRACT ***

C*PURPOSE prints prompt text conditionally
C*AUDIT HISTORY
C Densmore 17-Mar-83 AUTHOR
C*TYPE manual assigner routine
C*FORMAL PARAMETERS
Cin text prompt text
C*COMMON BLOCKS
Cin/out asgn assigner data block
C*CALLER various ASN routines
C**

12-115
C SUBROUTINE asnprv(isub,val,nval)
C
*** FORMAL PARAMETER DECLARATIONS ***
C
INTEGER isub,nval,val(nval)

*** ABSTRACT ***
C
C#PURPOSE implements manual assigner previous (up) page command
C
C#AUDIT HISTORY
C
Densmore 15-Mar-83 AUTHOR
C
C#TYPE manual assigner routine
C
C#FORMAL PARAMETERS
C
isub subcommand index
val user-input values array
nval length of val

C#COMMON BLOCKS
scrchr screen characters
asgn manual assigner blocks

C#CALLER assign

C#METHOD
C
If TOP then set top limits and locate lower limits.
C
If UP then set lower limits from former upper limits and locate
C
upper limits; if top limits are crossed, then a TOP command
C
done.
C
SUBROUTINE asnrdc(msg, hlptxt, case, proc, buffer, len, vld)

PARAMETER lcase=4, lproc=4

INTEGER len
LOGICAL vld
CHARACTER msg*2SS, hlptxt*B, case*lcase, proc*lproc, buffer*(len)

CSPURPOSE Semi-general purpose input routine for assigner
C Routine name means ASsigner ReaD Character string
C
C*AUDIT HISTORY
C Densmore 14-Jul-83 AUTHOR
C
CSTYPE assigner routine
C
CFORMAL PARAMETERS

Cin msg OIS message to print as a prompt before reading
Cin hlptxt Char*8 help text -- not needed if lproc=3
Cin case character case flag; first 4 characters significant
C 'CLASS ' : set case for class name
C 'YARD ' : set case for yard name
C 'UPPER ' : set case to uppercase
C 'LOWER ' : set case to lowercase
C 'NOCASE ' : do not change case of input
Cin proc Process flag; first 4 characters significant
C 'READ ' : Read buffer; check & process it
C 'NULLREAD' : Like READ, + allow null input
C 'CHECK ' : Check buffer, process it
C 'PROC ' : Only process buffer
Cin/out buffer "in" only if proc isn't READ; this is the buffer
C which is optionally input and checked
Cin len number of characters in buffer
Cout vld False if buffer not valid (like for EOF) or for POP

C#COMMON BLOCKS

Cin asgn assigner data block
Cin srcchr screen characters

C#METHOD

C#LOCAL VARIABLES

C cases character-case options
C proc procedural options
C alwrd stmt fn means case is READ or NULLREAD
C

SUBROUTINE asnref

C#PURPOSE asks for the regular screen refresh
C#AUTHOR Densmore 1 April 1983
C##
SUBROUTINE asnreo(ormode)

*** FORMAL PARAMETER DECLARATIONS ***
INTEGER ormode

*** ABSTRACT ***

CONDITIONALLY reorders classes in each yard WITH respect TO their input ORDER.

AUDIT HISTORY
Densmore 23-Jun-83 AUTHOR

TYPE assigner INBOUND routine

FORMAL PARAMETERS
ormode: Zero bit means ALPHABETIC ordering desired;
One bit means output tuples IN ORDER OF input;
Bit 1: for sort order OF shipclasses...
shipclass ORDER is ignored by asntpx, so ORDER is always input ALPHABETIC. This routine determines and accomplishes class reordering IF NEEDED.
Bit 2: for sort order OF shipyards... shipyard order is currently ignored.
If insufficient space EXISTS to order a yard's classes alphabetically, the ORDER mode IS changed to input-order-for-classes by force.

COMMON BLOCKS
asgn assigner data block
const for llarge

CALLER asndbi

METHOD
Recover ordering values from class buffer lines and save the pointers to each line. Use jhash to find new ordering. Set nextp values.

LOCAL VARIABLES
omin, omax smallest/largest orders value
ptrs the pointers FOR each YARD
orders the ORDER numbers FOR each YARD; FROM shpord;
used AS data in the jhash CALL
mbuf one more THAN the maximum number of classes POSSIBLE in a YARD -- the last one is used TO set the end class buffer line's nextp to zero
item pointer to current class buffer line
loc index TO current YARD
count number OF class buffer lines IN this YARD
jhl ith(i)
jh ith(i+1)
l do loop index FROM 1 TO count
ih output FROM jhash giving reordering
alpha true IF class sort mode IS alphabetic
notbig true IF insufficient space to sort a YD's classes
ASSIGNER ABSTRACTS

C  ASNRFH***********************************************
$CONTROL segment=asgn,check=3
    SUBROUTINE asnrfh(isub,val,nval)
C*  *** FORMAL PARAMETER DECLARATIONS ***
    INTEGER isub,nval,val(nval)
C*  *** ABSTRACT ***
C#PURPOSE Refresh display on screen for ASN assigner module
C#AUDIT HISTORY
C  Densmore  17-Mar-83  AUTHOR
C#TYPE manual assigner routine
C#FORMAL PARAMETERS
Cin  isub  subcommand index
Cin  val  values array
Cin  nval  length of val
C#COMMON BLOCKS
Cin/out asgn  assigner data block
C#CALLER various ASN routines
C#

12-120
SUBROUTINE asnrlc(isub,val,nval)
*** FORMAL PARAMETER DECLARATIONS ***
INTEGER isub,nval, val(nval)
*** ABSTRACT ***

C** PURPOSE Implements Relocate command

C** AUDIT HISTORY
C  Densmore  08-Jul-83 AUTHOR

C** TYPE assigner high level routine

C** FORMAL PARAMETERS
Cin  isub  subcommand code
Cin  val   numeric values
Cin  nval  number of values

C** COMMON BLOCKS
Cin  scrchr  screen characters
C** CALLER assign
C**
SUBROUTINE asnssee(val,nval)

*** FORMAL PARAMETER DECLARATIONS ***
INTEGER nval,val(nval)

*** ABSTRACT ***
PURPOSE interpret the '??' command -- diagnostic switching

AUDIT HISTORY
Densmore 28-Mar-83 AUTHOR

TYPE manual assigner routine

FORMAL PARAMETERS
val(nval) values input from user; interpreted as menu subcmds

COMMON BLOCKS
asn data block

CALLER asnhlp

ASIGNER ABSTRACTS
SUBROUTINE asnswp(val,nval)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER nval, val(nval)

*** ABSTRACT ***

PURPOSE Implements Relocate Swap command

AUDIT HISTORY

 TYPE assigner routine

FORMAL PARAMETERS

val numeric values
nval number of values

COMMON BLOCKS

asgn assigner data block

CALLER asncrc

ASSIGNER ABSTRACTS
SUBROUTINE asntup(cursor,eof)

*** FORMAL PARAMETER DECLARATIONS ***

integer cursor
logical eof

*** ABSTRACT ***

C#PURPOSE  Reads the next tuple on cursor, which must always
be open on ncjodat.projj

C#AUDIT HISTORY
MSCarey   01-jul-83  AUTHOR

C#FORMAL PARAMETERS
 Cin   cursor    relate cursor number
 Cout  eof       true if end of relation found, or if scenario
               in found tuple not same as current scenario

C#COMMON BLOCKS
 Cin  asoprm    outbound parameters
 Cin  astfr     tuple/record holding buffers
 Cin  fld05     field list for ncjodat.projj
 Cin  scenar    scenario info

C#CALLER  asndbr

C#METHOD
Read the tuple and check if scenario is different from the
current one; if so then eof.

C##
SUBROUTINE asnunl(tuple,yard,class,series,scnaro,date,order,type)
C** FORMAL PARAMETER DECLARATIONS **
CHARACTER scnaro*12
C*** CHARACTER yard*mcyds, class*mccls, series*mccd, type*mcjt
INTEGER tuple(1)
INTEGER*4 date,order
C** ABSTRACT **
C* PURPOSE Unloads the tuple, in char form, into constituent parts
C* AUDIT HISTORY
C Densmore 22-Jun-83 AUTHOR
C*TYPE assigner inbound routine
C*FORMAL PARAMETERS
Cin tuple integer version of the tuple
Cout yard,class,scnaro yard/class/scenario names
Cout series job series code name, like Lead, etc.
Cout type job type, like NewCon, etc.
Cout date RELATE date on which the job is marked (one of
C award,start,keel,launch,delivery dates)
Cout order the ASN-Order value which determines input order
C*COMMON BLOCKS
Cin asgn assigner data block, used for parameter values
Cin asnvld assigner valid lists, used for parameter values
C*CALLER asntpx,nxtcls
C*METHOD
C Unloads items into the output variables in the order they are
C found in the tuple to begin with. Note the type conversions
C that are often necessary. The parameters tup... indicate the
C lengths allotted the various values within the tuple itself.
C The lengths of the output variables are determined from common
C block parameter statements. See asntpi for input tuple order.
C*LOCAL VARIABLES
C tup... *2 lengths of various tuple element values
C cbuf,ibuf buffer to avoid assignment type conversion
C ifour,jfour buffer to avoid conversion
C**
SUBROUTINE asنزل

*** ABSTRACT ***

PURPOSE Removes invalid yards/classes under current scenario so that an old buffer is consistent with new constraints.

AUDIT HISTORY

Oensmore 03-June-83 AUTHOR

TYPE assigner routine

COMMON BLOCKS

Cin/out asign assigner data block

CALLER asnini

METHOD

First, loop over yards and call valyrd for each to check that its appearance in the buffer is allowed. If so, then go on to the next yard until EOY. If not, then remyrd effectively increments the yard index.

Next, perform a similar loop for classes in each yard. One caveat is the possibility that the last class in a yard might be deleted.

LOCAL VARIABLES

iyard yard index
old old yard or class index as displayed on last refresh
lency actual character count of yard name
valid true if there have been no yard/class deletions yet (ie. all present are valid...used for header print)
before pointer to buffer record before current record
item pointer to current record
after pointer to next record
C
ASNID

CONTROL segment=asgn,check=3
SUBROUTINE aswrid(start,len,last)

*** FORMAL PARAMETER DECLARATIONS ***
INTEGER start,len,last

*** ABSTRACT ***
PURPOSE accepts start period index, computes len and last

HISTORY
Densmore 17-Mar-83 AUTHOR

TYPE manual assigner routine

FORMAL PARAMETERS
Cin start first period index
Cout len last-start+1
Cout last last period index -- MIN(numper,start+mperh-1)

COMMON BLOCKS
Cin/out asgn assigner data block
CALLER various ASN routines
**SUBROUTINE** asnyrd(loc,name,len)

**INTEGER** loc,len

**CHARACTER** name*12   12--mcyds /asgn/

**AUTHOR** Densmore  17-Mar-83

**TYPE** manual assigner routine

**FORMAL PARAMETERS**

Cin   loc   yard index
Cin   name   yard name
Cin   len   number of periods

**CALLER** asnrfl

**ABSTRACT**

prints yard name/index and the stuff
$CONTROL segment=ASGNO
C$TRACE asocmp:
  INTEGER FUNCTION asocmp(rvalue,fvalue,len)
C*

*** FORMAL PARAMETER DECLARATIONS ***

  integer len
  character*(len) rvalue,fvalue

*** ABSTRACT ***

C#PURPOSE  ASsigner Outbound field CoMParison utility. Returns
C  -1 if rvalue > fvalue, 0 if they are equal, 1 if rvalue<fvalue
C#AUDIT HISTORY
C  MSCarey  03-jul-83  AUTHOR
C#FORMAL PARAMETERS
Cin  rvalue  character string
Cin  fvalue  character string
Cin  len    length of strings
C#COMMON BLOCKS
C  none
C#CALLER asndbr
C#METHOD
C  Comparison.
C#

12-129
SUBROUTINE asodel(mcyds,yard,nclas,mccls,clist)

*** FORMAL PARAMETER DECLARATIONS ***
integer mcyds,mccls,nclas
character*(mcyds) yard
character*(mccls) clist(nclas)

*** ABSTRACT ***
ASsigner Outbound tuple DELetion utility. Deletes a tuple from ncjodat.projj if appropriate.

MSCarey 03-jul-83 AUTHOR

C Find out if the current tuple is one of interest for this invocation. If not, return. If so, delete it unless it is flagged and get the next tuple. If it is flagged, then see if it is one that was already processed by ashard. That will be true if its (yard class jobt) is the same as that of the PREVIOUS tupfil record, indicating that this tuple is a trailer in the ordering sequence but is still of a type processed by ashard (since there is a tuple in ashard with the same key).
FORMAL PARAMETER DECLARATIONS

character*8  yard, class*12
logical  all

ABSTRACT

PURPOSE
ASsigner Planning Factor LoaDer. Reads one or all job description tuples for a given yard-class-job type from ncjdat.descj and stores them in /asjd/.

AUDIT HISTORY
MSCarey  27-jun-83  AUTHOR

PARAMETERS

yard  name of yard to find job desc for
class  class and job type to find desc for
howmny  true if caller wants all tuples matching
input key, false if only first one needed

COMMON BLOCKS

scenario  scenario field value information
asoprm  outbound parameters
asjd  job description holding buffer
rcrd06  interface buffer for relation ncjdat.descj

CALLER
various assigner outbound

METHOD
Convert the passed key values to the proper form for DB query.
Point to the first match and place in the holding area.
Read the rest of the matches if desired.
C ASPFTR ********************************************
$CONTROL segment=ASGNO
C$TRACE aspftr:
    SUBROUTINE aspftr(recnum)
C* *** FORMAL PARAMETER DECLARATIONS ***
    integer recnum
C* *** ABSTRACT ***
CIPURPOSE Assigner Planning Factor Transfer. Moves a job
C description record from rcrd06 to row recnum of /asjd/
C#AUDIT HISTORY
C MSCarey 29-jun-83 AUTHOR
C#FORMAL PARAMETERS
Cin recnum record number (row) to move data to in asjd
C#COMMON BLOCKS
Cin rcrd06 holds job desc record extracted from ncjdat
Cio asjd holds many job description records
C#CALLER aspfld
C#METHOD
C Lots of assignments
C
SUBROUTINE aspred(nclash)

integer nclash

*** ABSTRACT ***

Spreads out schedule dates reasonably evenly.

MSCarey 11-aug-83 AUTHOR

Cin nclash number of classes in record buffer

Cin asoprm assigner outbound parameters

Cio asrbuf buffer holding ships/dates to spread

Cin lprnts debug switches

The work area here is /asrbuf/, which is in order of
adjust or spreading date of the ships in the current
complexity group. There is also a linked-list ordering
by class.

Figure out the first and last allowable adjdat-s, and
from these and the number of ships get the size of the
interval with which to spread all the ships evenly. Do
so.

Then go through each class, looking for dates earlier or
later than their limits. On finding an earlier, add the
minimum number of days to bring it up to its lower limit, and
add this number to all subsequent ships of the class.
On finding one later than its limits, group all ships in the
class with the same limits and spread evenly within those limits,
doing nothing to ships in following periods.

12-133
SUBROUTINE ASPRD2(nclash)

*** FORMAL PARAMETER DECLARATIONS ***

integer nclash

*** ABSTRACT ***

Purpose: Spreads out schedule dates reasonably evenly.

This version is original, now is secondary.

MSCarey 11-aug-83 AUTHOR

Formal Parameters

cin nclash number of classes in record buffer

C#COMMON BLOCKS

Cin asoprm assigner outbound parameters

Cio asrbuf buffer holding ships/dates to spread

Cin lprnts debug switches

C#CALLER asncnv

Method

The work area here is /asrbuf/, which is in order of
adjust or spreading date of the ships in the current
complexity group. There is also a linked-list ordering
by class.

Figure out the first and last allowable adjdat-s, and
from these and the number of ships get the size of the
interval with which to spread all the ships evenly. Do
so.

Then go through each class, looking for dates earlier or
later than their limits. On finding an earlier, add the
minimum number of days to bring it up to its lower limit, and
add this number to all subsequent ships of the class.
On finding one later than its limits, group all ships in the
class with the same limits and spread evenly within those limits.

doing nothing to ships in following periods.
C ASTUPF

$CONTROL segment=ASGNO
C$TRACE asycls;

SUBROUTINE astupf(mccls,nclash,hldcls,mcyds,yard,timyd,orgpos,
   hldnum,nclas,tupfst,nxtupf)

C*** FORMAL PARAMETER DECLARATIONS ***
integer mccls,nclash,mcyds,nclas,nxtupf,tupfst(nclas)
integer orgpos(nclas),hldnum(nclas)
integer*4 timyd
character*(mccls) hldcIs(nclish), yard*(mcyds)

C*** ABSTRACT ***
C#PURPOSE  Converts /asrbuf/ records to tupfil tuple images.
C  When all classes in yard have been processed, these tuple
C  images will be used to update the data base.
C#AUDIT HISTORY
C  MSCarey 20-jun-83 AUTHOR
C#FORMAL PARAMETERS
Cin  hldcls name of each class
Cin  ydname name of yard holding these classes
Cin  timyd time stamp for this yard; used to get asnorder
Cin  orgpos position of clist i on assigner list (for asnorder)
Cin  hldnum position of hldcls i on clist
Cin  nclas number of classes in this yard
Cio  tupfst pointer to first tuple image for class i
Cio  nxtupf next free record in tupfil
C#COMMON BLOCKS
Cin  asrbuf holds 1-ship 1-record structure for this comp-grp
Cio  asjd job descriptions
Cout astfr record for reading/writing tupfil
C#CALLER asncnv
C#METHOD
C  Loop over the number of ships in each class. Construct a tuple
C  image for each of these ships (load planning factors for class
C  at the outset). Then add this image to tupfil, maintaining
C  the pointer structure.
C SUBDL ........................................................................................................

$CONTROL segment=asgno

SUBROUTINE asubdl(clchr,class,hull,comnum,map)

C* *** FORMAL PARAMETER DECLARATIONS ***

integer hull,comnum,clchr
integer*4 map
character*(clchr) class

C* *** ABSTRACT ***

C#PURPOSE Deletes tuples in relations subsidiary to schedule relation (ncjodat.projj). Hardwires only.

C#AUDIT HISTORY

C MSCarey 09-aug-83 AUTHOR

C#FORMAL PARAMETERS

Cin clchr chars in class
Cin class name of class to delete
Cin hull hull number of ship to delete
Cin comnum commissioning number of ship to delete
Cin map bit mapped variable indicating which relations
to delete tuples from. Mapping is: [0 --> 15]
C 15:ncjodol 14:ncjolbr 13:ncjoe mp 12:ncjomr
C 11:ncjomd 10:ncjocom

C#COMMON BLOCKS

Cin scenar holds current scenario information

C#CALLER ashard,asodel

C#METHOD

C Set up the point transfer buffer.
C Parse the bit map to see which relations to look in dynamically.
C If a bit is on, make sure that its relation is open.
C Point towards the target record, then delete it.
C Look for trailers with the same key value.

C#
SUBROUTINE asycls(iyard,lstchr,lstlen,clist,cptr,orgpos,nclas)

**FORMAL PARAMETER DECLARATIONS**

integer iyard,lstchr,lstlen
integer nclas,cptr(lstlen),orgpos(lstlen)
character*(lstchr) clist(lstlen)

**ABSTRACT**

ASigner Yard Classes. Constructs a sorted list of the non-repair job classes in yard iyard.

**PURPOSE**

ASigner Yard Classes. Constructs a sorted list of the non-repair job classes in yard iyard.

**FORMAL PARAMETERS**

Cin iyard index of the yard in /casgn/
Cin lstchr max length of any class name
Cin lstlen max number of classes to be returned
Cout clist the sorted list of classes (job type char attached)
Cout cptr bufasn record pointers for each class
Cout orgpos original position in display order of clist(i)
Cout nclas number of classes returned

**COMMON BLOCKS**

Cio asgn bufasn and editing-phase blocks

**METHOD**

Do gets from bufasn until no more in iyard. Check for repair job types and ignore them. Retain the pointer for each class gotten. Sort the final list and pointers.
SUBROUTINE ckpf(clsjob,yard,holdup,ok)

character clsjob*12,yard*8
logical holdup,ok

*** ABSTRACT ***

C#PURPOSE  Checks to make sure there are planning factors for
C  new class-jobs entered by user.

C#AUDIT HISTORY
C  MSCarey  29-mar-84  AUTHOR

C#FORMAL PARAMETERS
Cin  clsjob   name of ship class as entered, with job type char
Cin  yard     name of yard job assigned to
Cin  holdup   true if should stop execution to ensure msg seen
Cout ok      true if job desc found

C#COMMON BLOCKS
Cin  asnvld   validity info

C#CALLER  asnadd

C#METHOD
C  Decode clsjob into data base key values and do a point
C  on the job description cursor; success of point means ok.

C#LOCAL VARIABLES
C  jchar   job type, single-character representation
C  target dummy data destination
SUBROUTINE cmnget
CHARACTER*16 scn

*** ABSTRACT ***

C#PURPOSE Get-Save /casgn/ & /nasgn/ in asgn include - unit uasnc
C#AUDIT HISTORY
C Densmore 04-Apr-83 AUTHOR
C#TYPE Manual Assigner Utility
C#FORMAL PARAMETERS
Cout scn scenario name in disk buffer
C#COMMON BLOCKS
Cin/out asgn assigner data block
C#CALLER asnini, asncmd
C#METHOD
C Simply uses variable format read/write
C#LOCAL VARIABLES
FUNCTION dbasis(disbas,mode)

** FORMAL PARAMETER DECLARATIONS **

PARAMETER mdb=8
CHARACTER dbasis*mdb
CHARACTER disbas*mdb
CHARACTER mode*6

** ABSTRACT **

Returns appropriate date fieldname for the job relations given the menu display basis string and a character mode describing the job relation.

** AUDIT HISTORY **

Densmore 23-Jun-83 AUTHOR

** TYPE **

assigner routine

** FORMAL PARAMETERS **

Cin disbas display basis, from menu system; describes the date to be used in determining the period to which a job (tuple) belongs; one of: 'AWARD', 'START', 'KEEL', 'LAUNCH', 'DELIVERY'

Cin mode type of job relation; one of: 'NEWCON', 'REPAIR'

** CALLER **

asntpx, asntpi

** METHOD **

trivial; if mdb is changed from 8 note required code changes

12-140
SUBROUTINE getasn(ptr)

** FORMAL PARAMETER DECLARATIONS **

INTEGER ptr

*** ABSTRACT ***

PURPOSE Gets/Puts assignment record into position ptr

AUDIT HISTORY

C TYPE manual assigner routine

FORMAL PARAMETERS

Cin, ptr assignment record position

COMMON BLOCKS

Cin/out asgn assigner data block

CALLER various ASN routines

**
**INICLS**

```plaintext
$CONTROL segment=asgn, check=3
SUBROUTINE inicls(class, order, loc)

C* *** FORMAL PARAMETER DECLARATIONS ***
C *** CHARACTER*mccls class
   INTEGER*4 order
   INTEGER loc

C* *** ABSTRACT ***
C* PURPOSE Sets up sums and increments for a new class buffer
C* AUDIT HISTORY
C Densmore 10-May-83 AUTHOR
C* TYPE assigner routine
C* FORMAL PARAMETERS
C in class class name for new buffer
C in order ship order (asn order)
C in loc yard location in which new buffer resides
C* COMMON BLOCKS
C in asgn assigner data block
C* CALLER asnadd,asnins
C* METHOD
C straightforward
C#
```
SUBROUTINE iniyrd(loc,yardnm)

*** FORMAL PARAMETER DECLARATIONS ***
CHARACTER*mcyds yardnm
INTEGER loc

*** ABSTRACT ***
Purpose: Finishes the process of adding a yard

History:
Oensmore 10-May-83

Type: assigner routine

Formal Parameters:
loc  yard index
yardnm  yard name

Common Blocks:
assigner data block

Caller: newyrd

Method:
straightforward

12-143
ASSIGNER ABSTRACTS

C LOCYRD=================================
CONTROL segment=asgn, check=3
  INTEGER FUNCTION locyrd(yard,names,len)
  *** FORMAL PARAMETER DECLARATIONS ***
  INTEGER len
  *** CHARACTER*12 yard,names(len) -- 12==mcyds /asgn/
  *** ABSTRACT ***
PURPOSE Locates yard in names array via binary search
AUDIT HISTORY
Densmore          17-Mar-83 AUTHOR
TYPE manual assigner routine
FORMAL PARAMETERS
  cin yard        yard name to locate
  cin names       sorted array of yard names to search
  cin len         length of names
COMMON BLOCKS
  cin/out asgn    assigner data block
CALLER asnadd
METHOD
  Binary search. If the yard is not found, locyrd is returned
  such that if the yard were inserted it would become number
  locyrd and items locyrd on would be right-shifted. NOTE: locyrd
  assumes the dimension of NAMES to be at least len+1 on failure.
LOCAL VARIABLES
  left,right -- search positions
  locyrd, the returned value, is also used as the mid value

12-144
ASSIGNER ABSTRACTS

C NEWYRD-----------------------------
$CONTROL segment=asgnd,check=3
   SUBROUTINE newyrd(loc,defind)
C*       *** FORMAL PARAMETER DECLARATIONS ***
   INTEGER loc
   LOGICAL defind
C*       *** ABSTRACT ***
C* PURPOSE adds new yard
C* AUDIT HISTORY
C TYPE manual assigner routine
C FORMAL PARAMETERS
C locType loc  yard index; returned 0 if abort is desired
C defind (IF defind THEN locType="in/out" ELSE locType="out")
C COMMON BLOCKS
C asgn scrchr asgn assigner data
C scrchr screen characters
C CALLER asnadd
C METHOD
C Checks for input and system errors; obtains yard name input;
C Right shifts appropriate arrays to keep these arrays sorted
C by yard name.
C LOCAL VARIABLES
C yardnm local yard name input
C lenr length of right shift
C msg character buffer for asnpro message
C leny length of yard name
C

12-145
ASSIGNER ABSTRACTS

C NXTCLS******************************************************************************
$CONTROL segment=assigni,check=3
SUBROUTINE nxtcls(cursor,fields,vldydi,firstd,lastd,
  1 tupmax,tuple,len)
C*                      *** FORMAL PARAMETER DECLARATIONS ***
INTEGER cursor,vldydi,len
CHARACTER fields*255
INTEGER tupmax(len),tuple(len)
INTEGER*4 firstd,lastd
C*                      *** ABSTRACT ***
C$PURPOSE Grabs the next tuple from the relation given by cursor
C$AUDIT HISTORY
C  Densmore 22-Jun-83 AUTHOR
C$TYPE assigner inbound routine
C$FORMAL PARAMETERS
Cin  cursor cursor index to appropriate relation
Cin  fields DTS string giving names of fields to be returned
Cin/out vldydi index to the list of valid yards in /asnvld/
Cin  firstd RELATE representation of first date of first period
Cin  lastd RELATE rep of last date of last period
Cin  tupmax maximum value of a tuple...used when vldydi max
C  value is exceeded to set 'tuple', so that the
C  hash sorting done by the caller still works
Cout  tuple returned tuple value
Cin  len length of tuple...better be .GE. than fields implies
C$COMMON BLOCKS
Cin  asnvld assigner valid lists
C$CALLER asntpx,asntpx$asntpi
C$METHOD
C  Nxtcls is divided into two parts: a yard search section
C  and a class search section. The routine performs an initial-
C  ization part first if vldydi is zero on entry. Note that
C  NeXT-Tuple operations are attempted before CALc operations are
C  performed during searches for a next valid yard. Next ops are
C  much less expensive than are Calc ops.
C$LOCAL VARIABLES
C  eot True when no more tuples in that relation
C  or when a calc operation failed
C  yard,class,series, - tuple elements unloaded
C  scnaro,type,date,order /
C  clcomp *mcvcls version of class, for comparison purposes
C$
SUBROUTINE nxtclz(routin, where, tuple, string, ls, eot)

*** FORMAL PARAMETER DECLARATIONS ***

INTEGER ls, tuple(l)
CHARACTER routin*6, where*255, string*(ls)
LOGICAL eot

*** ABSTRACT ***

CSPURPOSE Prints diagnostics for NXTCLS
C#AUDIT HISTORY
C    Densmore  04-Jul-83 AUTHOR
C#TYPE assigner inbound diagnostic routine
C#FORMAL PARAMETERS
Cin routin  *6 caller name
Cin where  DTS description of where the call is being made
Cin tuple  integer version of tuple read, if any
Cin string  calc string, or whatever else, of length ls
Cin ls     length of string
Cin eot    True if no tuple was read because end-of...
C#COMMON BLOCKS
Cin lprnts diagnostic common block
C#CALLER nxtcls
C##
SUBROUTINE remcls(loc, before, item, after)

INTEGER loc, before, item, after

*** ABSTRACT ***

C#PURPOSE Performs decrements and repointering for removal of a class

C#AUDIT HISTORY

C Densmore 06-Jun-83 AUTHOR

C#TYPE assigner routine

C#FORMAL PARAMETERS

Cin loc yard index

Cin before pointer to class before the one to be deleted

Cin item pointer to the one scheduled for deletion

Cin after pointer to the next one after item

C#COMMON BLOCKS

Cin/out asgn assigner data block

C#CALLER asnval, asndel

C#METHOD

C Decrements asntot, numasn, sumper, grdtot. Re-route pointers.

C#
ASSIGNER ABSTRACTS

C REMYRD***SUBROUTINE remyrd(loc)

C* === FORMAL PARAMETER DECLARATIONS ===

C* INTEGER loc

C* === ABSTRACT ===

C#PURPOSE Removes an entire yard

C#AUDIT HISTORY

C Densmore 16-Mar-83 AUTHOR

C#TYPE manual assigner routine

C#FORMAL PARAMETERS

Cin loc yard index location

C#COMMON BLOCKS

Cin scrchr screen characters

Cin/out asgn assigner data

C#CALLER asndel

C#METHOD

C Prompts to make sure if Prompt=.True.; then runs down the C assignent buffer list and decrements all summary arrays. C The list is then CONSed to the free chain, and all approp- C riate arrays are left-shifted.

C#LOCAL VARIABLES

C msg asnpro message buffer

C verify input containing "" or "?

C item assignments buffer pointer

C leni length of left shift

C#
SUBROUTINE svstat

*** ABSTRACT ***

C#PURPOSE save/restore screen status for line printer outputs

C#AUDIT HISTORY

C#  Densmore  11-May-83  AUTHOR

C#TYPE assigner utility

C#CALLER asnlpr

C##
C TMSTMP

$CONTROL segment=ASGNO
   INTEGER*4 FUNCTION tmstmp(idum)
C* *** FORMAL PARAMETER DECLARATIONS ***
   integer idum
C* *** ABSTRACT ***
C#PURPOSE Returns current time in seconds since 1/1/1983
C#AUDIT HISTORY
C#MSCarey 03-jui-83 AUTHOR
C#FORMAL PARAMETERS
Cin idum dummy
C#COMMON BLOCKS
Cin tddate date manipulation functions
C#CALLER asncnv
C#METHOD
C Get the current date and convert it to seconds: then get
C the time of day and add it on.
C**
**ASSIGNER ABSTRACTS**

```c
C TUPFRD *****************************
$CONTROL segment=ASGNO
SUBROUTINE tupfrd(recnum,iclass, tupfst, nclas, eofil,
  lstyrd, lstcls, lstjob)
C
*** FORMAL PARAMETER DECLARATIONS ***
  integer recnum, iclas, nclas, tupfst(nclas)
  logical eofil
  character lstyrd*8, lstcls*10, lstjob*6
C
*** ABSTRACT ***
C$PURPOSE Read a record from direct access tuple holding file.
C$AUDIT HISTORY
C MSCarey 01-jul-83 AUTHOR
C$FORMAL PARAMETERS
  C io  recnum in: record to read; out: next record
  C io  iclas location on clist of current class-job type
  C in  tupfst pointers to first tupfil record each class-job
  C in  nclas number of class-job types this yard
  C io  eofil true if no more tupfil records
  C out lstyrd yard for tupfil record in memory on call
  C out lstcls class "
  C out lstjob job type "
C$COMMON BLOCKS
  C in  asoprm outbound parameters
  C out astfr tuple/record holding records
C$CALLER asdbr
C$METHOD
C Read and reset recnum
C$$
```

12-152
ASSIGNER ABSTRACTS

C VALCLS***************************************************
$CONTROL segment=asgn, check=3
   LOGICAL FUNCTION valcls(class)
C* *** FORMAL PARAMETER DECLARATIONS ***
C *** CHARACTER*mccls class
C* *** ABSTRACT ***
C#PURPOSE Determine if input class is allowed in this scenario
C#AUDIT HISTORY
C     Densmore  02-Jun-83 AUTHOR
C#TYPE assigner routine
C#FORMAL PARAMETERS
Cin class     input class name...character*mccls
C#COMMON BLOCKS
Cin asgn assigner data block
C#CALLER asnadd
C#METHOD
C Matches against legal list; if match then verifies validity
C**
VALYRD

**CONTROL** segment=asgn,check=3

**LOGICAL FUNCTION** valyrd(yard)

**C**

***** FORMAL PARAMETER DECLARATIONS *****

**C**

**CHARACTER** mcyds yard

**C**

***** ABSTRACT *****

**C**

**PURPOSE** Determines if input yard is valid in this scenario

**C**

**AUDIT HISTORY**

**C**

Densmore        02-Jun-83 AUTHOR

**C**

**TYPE** assigner routine

**C**

**FORMAL PARAMETERS**

**C**

yard input yard

**C**

**COMMON BLOCKS**

**C**

asgn assigner data block

**C**

**CALLER** newyrd

**C**

**METHOD**

**C**

Verifies that input yard is on the valid yards list

**C**

**C**
**ASSIGNER ABSTRACTS**

C

VLDSLST

$CONTROL segment=assign1,check=3
SUBROUTINE vldlst

C* ABSTRACT

C#PURPOSE Initializes valid lists for assigner (classes,yards)
C#AUDIT HISTORY
C Densmore 10-Jun-83 AUTHOR
C#TYPE assigner routine
C#COMMON BLOCKS
Cin scenar current scenario
Cout asnvld valid lists
C#CALLER asnini
C#METHOD
C simply uses liston for each list being initialized
C##
C SUBROUTINE vldlsz

*** ABSTRACT ***
PurPOSE Prints diagnostic information for vldlst on /asnvld/

AUDIT HISTORY

Densmore 05-Jul-83 AUTHOR

TYPE assigner inbound diagnostic

COMMON BLOCKS

asnvld assigner valid arrays

CALLER vldlst

METHOD just a set of prints and trecol calls

##
ASSIGNER ABSTRACTS

C YCASN

$CONTROL segment=asgni,check=3
SUBROUTINE ycasn
LOGICAL flag

*** ABSTRACT ***

C#PURPOSE Implements Control-Y in Refresh
C#AUDIT HISTORY
C Densmore 06-Apr-83 AUTHOR
C#TYPE Assigner utility
C#METHOD
C To use this routine with a specified module, one relies on an
C unspecified compiler specific mechanism to call YCASN on some
C user requested interrupt. For example, on the HP, the exec-
C ution of the Fortran statement " ON CONTROL Y CALL YCASN"
C performed this function. This routine assumes that following
C completion of the interrupt process (ie. the call to YCASN)
C control returns to where it was before the interrupt. Thus,
C to discover that the interrupt occurred, entry YCASNR may be
C called; it returns .TRUE. if so, and .FALSE. otherwise, while
C resetting the internal flag (save). YCASNI may be called to
C initialize this process. Note that the ON statement need not
C be reset, since calling YCASN is harmless, and only effective
C if YCASNR is being called in a loop or something like that.
C##
C  SUBROUTINE yrdcpy(from,loc,defind,succes)
  INTEGER from,loc
  LOGICAL defind,succes

C* *** ABSTRACT ***
CPURPOSE Implements yard copy
CAUDIT HISTORY
CDensmore 08-Jul-83 AUTHOR
CTYPE assigner routine
CFORMAL PARAMETERS
Cin from from yard
Cin loc to yard
Cin defind .TRUE. if loc is defined
Cout succes true if successful
CCOMMON BLOCKS
Cin/out asgn assigner data block
CCALLER asncpy
CMETHOD
C  This copy-yard part does the customary checks and then adds
C  the new yard via a newyrd call. Then it is assured that enough
C  class buffers are actually allocated on the free chain to permit
C  using it without changing any of the pointers within it. Finally,
C  the appropriate data is placed in each class buffer by looping
C  over the from buffers.
CLOCAL VARIABLES
C  ochain pointer to current part of old yard class chain
C  nchain pointer for new yard class chain
C  from old or from yard location
C  loc new or to yard location
C  new used as a do index
C  item pointer to current buffer
C  end pointer to last buffer in allocated chain

12-158
APPENDIX A

CROSS REFERENCE FROM AUTOMATED DATA SYSTEMS DOCUMENTATION STANDARDS CONTENTS TO ALIAS GUIDES CONTENTS

A.0 PURPOSE OF THE APPENDIX

The set of manuals which form the documentation for ALIAS do not conform strictly to the DoD 7935.1-S documentation standard. They contain all the information mandated by the standard (with the exception of Functional Description, Test Plan, and Test Report) and more, but are organized differently. The organization of the standard is not well suited to ALIAS, and would have resulted in much less useful documentation.

This Appendix lists sections in the ALIAS Guides which contain the information mandated in each section of the standard. It is organized according to the tables of contents for the standard manuals, with references to one or more sections in the documentation as written. A reader wishing to have information presented in the order given by the standard tables of contents may detach this Appendix and use it to order his reading of the documentation.

In a few cases, sections mandated by the standard were not relevant to ALIAS. Comments regarding this are included in this Appendix.

In order to conserve space, references to the various guides are made according to the following scheme: a section in a particular guide is designated as G-#.#, where G represents the code for the guide and #.#... is the actual section number within it. Codes for the Guides are:

U: Alias User's Guide
P: Alias Guide to System Maintenance and Expansion
D: Alias Data Base Reference Guide
X: Any ALIAS Guide

A-1
A.1 SYSTEM/SUBSYSTEM SPECIFICATION

1.0 GENERAL

1.1 Purpose of the System/Subsystem Specification
see mainly P-1.1

1.2 Project References
see P-1.1.3, P-1.2, P-1.3

1.3 Terms and Abbreviations
see U-2, P-2

2.0 SUMMARY OF REQUIREMENTS

2.1 System/Subsystem Description
see U-1.3, P-1.2, P-1.3

2.2 System/Subsystem Functions
see P-1.2, P-1.3, P-8, P-11 and onward

2.2.1 Accuracy and Validity
no reference

All ALIAS calculations must be carried out with a normal degree of accuracy; that is, the nature of the problems are not such that extraordinary mathematical precision is required, as it sometimes is for scientific problems.

2.2.2 Timing
see P-2.3.11, P-1.3.2.7

In general, response time should be minimized, and for functions requiring a great deal of time, off-line execution options should be available.

2.3 Flexibility
see P-1.3, P-1.4, P-2, P-6, P-8, P-9.1

3.0 ENVIRONMENT
see P-6

3.1 Equipment Environment
see P-4

3.2 Support Software Environment
see P-4, P-5
3.3 Interfaces
see P-1.3.2.5, P-2.3.8, P-2.3.9, P-2.2.5, P-2.3.2, P-8.2.5, P-8.3.5, P-8.4.5, P-9, P-10, P-11 and onward

3.4 Security and Privacy
see P-7, P-8.3, P-8.4, P-11 and onward

3.5 Controls
see P-7, P-8

4.0 DESIGN DETAILS

4.1 General Operating Procedures
see P-1.3, U-1.3, U-4, U-5

4.2 System Logical Flow
see P-1.3, P-3.2, P-8.1, P-11 and onward

4.3 System Data
see U-5, P-3.2, P-8.2.4 and onward

4.4 Program Descriptions
see P-8, P-10, P-11 and onward

A.2 PROGRAM SPECIFICATIONS

1.0 GENERAL

1.1 Purpose of the Program Specification
see P-1.1, P-8, P-11 and onward

1.2 Project References
see P-1.1.3, P-1.2, P-1.3

1.3 Terms and Abbreviations
see U-2, P-2

2.0 SUMMARY OF REQUIREMENTS
see P-1.2, P-2, P-8.2.1, P-8.3.1, P-8.4.1, P-11 and onward

General system requirements and standards are covered in the early sections of the Maintenance Guide, while specifics for each system module are covered in Section 8 and Sections 11-12. The remarks on accuracy and validity made above (A.2-2.2.1) apply.
3.0 ENVIRONMENT
see P-6

3.1 Equipment Environment
see P-4

3.2 Support Software Environment
see P-4, P-5

3.3 Interfaces
see P-1.3.2.5, P-2.3.8, P-2.3.9, P-2.2.5, P-2.3.2, P-8, P-9, P-10, P-11 and onward

3.4 Security and Privacy
see P-7, P-8.3, P-8.4, P-11 and onward

3.5 Controls
see P-7

4.0 DESIGN DETAILS
see P-8, P-11 and onward

A.3 DATA BASE SPECIFICATIONS

1.0 GENERAL

1.1 Purpose of the Data Base Specification
see D-1.1

1.2 Project References
see P-1.1.3, P-1.2, P-1.3, I-1

1.3 Terms and Abbreviations
see U-2, P-2

2.0 DATA BASE IDENTIFICATION AND DESCRIPTION

2.1 Data Base Identification
see D-1.4

2.1.1 System Using the Data Base
see D-1.3

2.1.2 Effective Dates
no reference
The ALIAS data base may be used with the ALIAS system for as long as the system is in existence. The data base is expected to expand and change continuously.
2.1.3 Storage Requirements
see D-2

2.1.4 Physical Description of Data Base Files
see D-2

2.2 Labeling/Tagging Conventions
see D-2

2.3 Organization of the Data Base
see D-2, D-5

2.4 Special Instructions
see D-3, D-4, D-5, D-6

2.5 Support Programs Available for Handling the Data Base
see D-4, D-6

3.0 DATA DEFINITIONS

3.1 Data Files
see D-2

3.2 Tables
same as files

3.3 Items
see D-2

3.4 Records and Entries
not applicable

4.0 INTEGRATED DATA BASE
See D-1.4, D-2.0, D-5, D-6

A.4 USERS MANUAL

1.0 GENERAL

1.1 Purpose of the Users Manual
see mainly U-1.1

1.2 Project References
see P-1.1.3, P-1.2, P-1.3

1.3 Terms and Abbreviations
see U-2, P-2

1.4 Security and Privacy
see U-7
2.0 SYSTEM SUMMARY

2.1 System Application
see U-1.2, U-1.3, U-2.0

2.2 System Operation
see U

2.3 System Configuration
see U-4

2.4 System Organization
see U-1.2, U-1.3, U-5

2.5 Performance
see U-1.2, U-1.3, U-4, U-5, U-7 and onward

2.6 Data Base
see U-6

2.7 General Description of Inputs, Processing, and Outputs
see U-1.3, U-5, U-7 and onward

3.0 STAFF FUNCTIONS RELATED TO TECHNICAL OPERATIONS

3.1 Initiation Procedures
see U-1.3, U-1.4, U-3, U-4

3.2 Staff Input Requirements
see U-1.4, U-3 and onward

3.3 Output Requirements
see U, P-8 and onward

4.0 FILE QUERY PROCEDURES

4.1 System Query Capabilities
see U-1.4, U-6, U-7, RELATE manuals

4.2 Data Base Format
see D-2

4.3 Query Preparation
see U-7, RELATE manuals

4.4 Control Instructions
not relevant
A.5 COMPUTER OPERATION MANUAL

Not provided for; structure of system does not call for separate class of operators. Program inventories, file inventories, processing and security descriptions may be found in the User's and Maintenance Guides.

A.6 PROGRAM MAINTENANCE MANUAL

1.0 GENERAL

1.1 Purpose of the Program Maintenance Manual
see mainly P-1.1

1.2 Project References
see P-1.1.3, P-1.2, P-1.3, I-1

1.3 Terms and Abbreviations
see U-2, P-2

2.0 SYSTEM DESCRIPTION

2.1 System Application
see P-1.2, P-1.3

2.2 Security and Privacy
see P-7

2.3 Program Description
see P-8, P-3, P-6, P-10, P-11 and onward

3.0 ENVIRONMENT

3.1 Equipment Environment
see P-4

3.2 Support Software
see P-5

3.3 Data Base
see D

4.0 PROGRAM MAINTENANCE PROCEDURES

4.1 Conventions
see P-2, P-6
4.2 Verification Procedures
   see P-9

4.3 Error Conditions
   see U-B

4.4 Special Maintenance Procedures
   see P-6

4.5 Special Maintenance Programs
   see P-6

4.6 Listings
   see P-Appendix C
This Appendix is meant as an introduction to the tasks which would need to be completed in order to convert ALIAS to run on a host computer other than an HP-3000. ALIAS is fundamentally a very host-dependent system, primarily because of its dependence on RELATE and BUILDER, but measures were taken during development to isolate dependency in order to minimize conversion costs.

This Appendix should not be construed as a complete listing of conversion requirements. Such a listing inevitably depends on the hardware and software of the target host as well as on the existing software.

The major host dependencies fall into four categories: RELATE, BUILDER, HP FORTRAN, and MPE dependencies.

B.1 RELATE DEPENDENCE

RELATE is the DBMS used to implement and access the ALIAS data base. A move to a new system will involve creation of the ALIAS data base structure using the DBMS on the new host, an unload of the data contained in RELATE files on the HP 3000, and a reload on the new host.

Although tedious and time consuming, these tasks are unlikely to present serious technical challenges or suprises. Converting ALIAS FORTRAN programs to access the new data base programmatically may be another matter. Such programmatic access must go through the equivalent of the RELATE Host Language Interface. Should the new DBMS fail to provide a cursor-oriented,
routine-call oriented interface, major revision of all ALIAS FORTRAN code would be required.

Given a cursor-oriented, general-purpose routine call interface structurally similar to the RELATE HLI (really the most common interface method among DBMSs) there may not be much problem. Although calling syntax and data structures may differ, (almost) all ALIAS programmatic DBMS usage is buffered through the DBIF, a library of interface routines written specifically to support future conversions. These are general-purpose routines whose formal parameters could remain unchanged while their guts were rewritten to work with the new DBMS. Should this be the case conversion changes would be isolated in perhaps two dozen routines and a few thousand lines of code.

There are two reasons why things might not be so rosy (in addition to the disaster of a non-cursor-oriented new DBMS). First, RELATE requires that data source and target buffers be word-aligned arrays or common blocks in which numeric and character variables are mixed according to data relation field data type. Many FORTRAN '77 compilers will not permit such constructs, making it possible that a different buffering method will be required. Since these buffers are just passed through DBIF calls, any changes to the buffering scheme would affect all ALIAS FORTRAN routines which use RELATE.

Second, most ALIAS modules are heavily dependent on the existence of RELATE's record-point query capability, as implemented in the rtpcal DBIF routine, and its particular idiosyncrasies. This dependency was necessitated by the large memory usage and execution time penalties imposed by trying to make queries using selections. Should the new DBMS fail to have a point capability, the design logic of many ALIAS query and update routines would have to be substantially changed. This might be desirable in any case, though, in order to take advantage of the efficiency features offered by a new DBMS.
B.2 BUILDER DEPENDENCY

The BUILDER screen application generator (a member of the RELATE family of software) was used to implement the Data Base Updating system and Data Dictionary, both central elements of the ALIAS system. It is likely that complete rewrites of both modules will be required on conversion, since BUILDER currently runs only on the HP 3000. BUILDER was used because the only alternative was to write a similar package from scratch; this package would have had so many host dependencies given the limitations imposed by the HP 3000 as to be no better than use of BUILDER for conversion purposes.

CRI, BUILDER's vendor, is considering conversion of BUILDER to run under UNIX with a variety of DBMSs. Should this occur the outlook in this area might improve substantially.

In considering alternative screen application packages for the new host system particular care should be taken to ensure that all the FUNCTIONAL features of the DBU can be implemented. Few packages offer the range and power of BUILDER.

B.3 HP 3000 FORTRAN DEPENDENCY

The HP 3000's FORTRAN compiler is a nonstandard extension of the ANSI '66 standard compiler which offers many ANSI '77 standard-equivalent features. Those converting ALIAS FORTRAN programs to run on a new machine will find that a moderate effort will be required to correct syntax and logic to conform to the ANSI standard.

The major problem will be with data structures which mix character and numeric data types, technically forbidden under the '77 standard. This mixing was necessary on the HP 3000 in order to support RELATE use, as noted above.
A number of syntax differences can probably be dealt with by writing a specialized editor to process all the code and make the necessary changes. HP FORTRAN uses 's' rather than 'a' as the variable-length character output descriptor in FORMAT statements, permits the use of quotes as character string delimiters, does not support in-line string concatenation, and limits character strings to 255 characters in length.

The OPEN and CLOSE statements are also not supported by HP FORTRAN, but all functionally equivalent calls in ALIAS have been isolated in calls to the filopn and filcls utilities, sharply restricting the extent of the resulting problem.

B.4 MPE DEPENDENCIES

ALIAS is dependent on the host operating system to a moderate extent. Due to the process memory limitation of the HP 3000 a good deal of process handling is done in running modules. Conversion personnel may elect to retain this multiprocessing capability if supported by the new machine, or may simply link all modules into a single large program as was originally planned. There are advantages to each approach. In any case, the number of routines in which process handling is done is sharply limited.

Likewise, ALIAS uses MPE extra data segments as extended data memory storage in cases where the 64K byte per-process data memory limitation is binding. This paging can be eliminated on a truly virtual machine. Again, the number of routines involved is rather limited.

Most other calls to operating system service routines are isolated in general-purpose FORTRAN utilities, minimizing the work required to move onto a new system.