LA-UR -81-3400

MASTER

TITLE DYNAMIC PARAMETER ARRAY SYSTEM (DPA

AUTHOR(S) G. T. Anderson
M. A. Oothoudt
J. F. Amann
T. Kozlowski

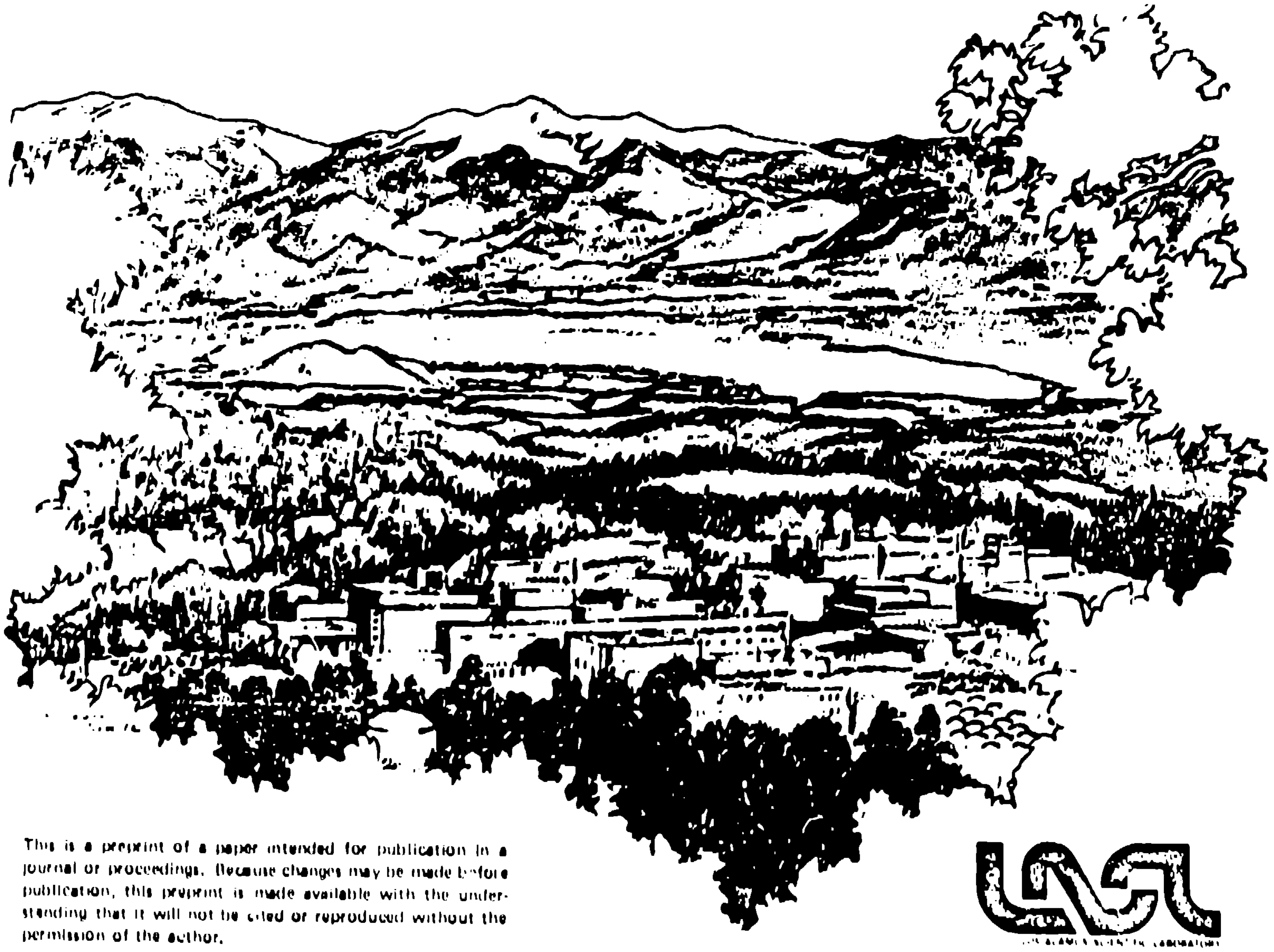
DISCLAIMER

SUBMITTED TO Digital Equipment Computer Users Society,
December 7-11, 1981, Los Angeles, CA

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

DYNAMIC PARAMETER ARRAY SYSTEM (DPA)

Gail T. Anderson
Michael A. Oothoudt
James F. Amann
Thomas Kozlowski



DYNAMIC PARAMETER ARRAY SYSTEM [*] (DPA)

Gail T. Anderson
Michael A. Oothoudt, James F. Amann, and Thomas Kozlowski
Los Alamos National Laboratory
Los Alamos, New Mexico

ABSTRACT

This paper describes software which provides a means to change dynamically (in real time) the values of parameters in an executing task. The parameters reside in an RSX-11M memory management region or a VMS shared global section. Values may be accessed and altered by any task attaching the region, including an interactive task. The interactive task may be run from the keyboard to allow user interaction with parameters stored in the region. Keyboard commands allow the user to make a disk file copy of values in the region, modify values in the region, and define meaningful synonyms to use in referring to parameters in the region.

OVERVIEW OF THE SOFTWARE

The purpose of DPA is to allow the user to change values of parameters in a specified user task while it is running. The software consists of a keyboard interface task named PRM and two subroutines to be called by user tasks to set up storage of the parameters. The interactive task PRM allows the user read/write access to the parameters from the terminal or from a command file. One subroutine PRMCRE may be called by the user's task to create the memory management region the parameters reside in; the other subroutine PRMMAP may be called by other tasks to attach and map the region.

The user's task may define any number of INTEGER*2, INTEGER*4, REAL*4, and REAL*8 variables (arrays I, II, R, and RR respectively) to be held in a memory management region and be accessible to other tasks. These four arrays are thought of as subsections of a parameter array. These variables are available for use by the user's tasks via a FORTRAN COMMON block.

This software is written in FLECS.[**]
This paper will discuss only the RSX-11M version; the VMS version will have identical functionality and user interface although somewhat different implementation where necessary.

Section II describes the history leading up to the writing of this software and illustrates the needs to be met. Section III describes the method of implementation of DPA and the relationship of DPA to other system routines. Section IV details the functionality provided from the point of view of the user.

HISTORY

Experimenters at Los Alamos Meson Physics Facility (LAMPF) frequently need to be able to change parameters in their data acquisition programs without going through the edit-compile-taskbuild cycle. Examples of such parameters are gains and pedestals for detectors, kinematics parameters (angles, energies, target masses, etc.), calibration coefficients and test limits. Note that such parameters may be changed several times an hour which means changing "hardwired" values in code is impractical.

A facility for changing the parameters in one task from another task was developed by several experimental groups. The simplest form used was a disk file containing the necessary constants. The user task would be notified of new data to read by front panel sense switches and would read in the file. This scheme has three major disadvantages:

1. The disk I/O is comparatively slow.
2. The disk I/O requires a large amount of code in the user's task.
3. The amount of effort required to change a single parameter is large, since the file must be edited and even finding the correct number in the file can be tedious.

---

[*] This work is supported by the U. S. Department of Energy.

[**] Developed by Terry Beyer, Department of Computer Science, University of Oregon. FLECS is a FORTRAN preprocessor allowing use of structured language constructs in FORTRAN.

A scheme used by James F. Amann under RSX-11D for two experimental groups was to use a Shared Global Area (SGA) linked to both a data analysis task and an input task. The input task could be used to set variables in the SGA by command, e.g., "R37=45.5." Such tasks normally kept the parameters stored in a disk file so that they could be restored after a reboot. Also, command file input and a line printer listing of values were allowed. Such a method provides fast updating of parameters and even allows simple multitask updating but does have two disadvantages:

1. The SGA required 4K words of virtual address space in the user's task even if only a few hundred parameters existed. The unused space was not easily available to the user.

2. The programs did little to help the user keep track of which variables were which. For example, typical input is R37=45.5 to set an angle, whereas THETA=45.5 is really what the user is thinking about.

The Dynamic Parameter Array software described herein attempts to solve the virtual address space problem and simplifies the user interface under RSX-11M and VMS.

IMPLEMENTATION

This section contains:

- A discussion of the relationship between the DPA system and other routines which may share the memory management region.

- A brief description of the implementation method of each piece of the DPA software.

DPA's Environment

The DPA subsystem will be distributed to users as part of a general-purpose data acquisition system used at LAMPF. But DPA can be used outside the data acquisition environment without any changes to DPA at all, for example in Monte Carlo calculations.

The connection between DPA and other supplied routines takes place by means of an RSX-11M memory management region which contains the array of parameters to be shared and may also contain other types of data. The first 32-word block of the region contains an array of offsets into the region (the Header array HDR), which is followed by the storage areas of the region. This allows the region to be divided into 32 areas which may be used for separate purposes, each of arbitrary length, each of which must begin on a 32-word boundary (for ease in mapping).

Details of Implementation

This section contains specifics of the DPA implementation. The points discussed are the structure of the memory management region, creation of and mapping the region, access to the region by the user at the terminal, and the disk files associated with DPA.

Structure of the Region - Access to the data in the region is available to any task which knows the "name" of the region. When the region is created, its name is defined as the default UIC under which the creating task runs with the first octal digit translated to a letter; e.g., if the task which creates the region runs under UIC [025,033], the name of the region is A25033. Thus any task running under the same UIC can access the region. The only information about the structure of the region needed by a secondary task which must access data in the region is that the first 32-word block contains offsets to the storage areas within the region. Therefore the task can map to the first 32-word block, obtain the necessary offset, and then remap to the required information.

Figure 1 is a simple view of the structure of the region. Each word in the Header array (the first 32 words of the region) contains the offset to the associated storage area; e.g., the first word in the Header array points to the beginning of the DPA parameter array which is always in the first storage area. Words 2 and 3 of the Header array point to the beginning addresses of data for other data acquisition routines. Words 4 through 16 are reserved for future expansion. Words 17 through 31 are available for special purposes the user may have. Word 32 of the Header array contains the size of the whole region. If any entry in the Header array contains 0, the associated storage section is not used; e.g., Words 4 through 15 will all contain 0 for the present release of the DPA software.

The advantage of this structure is that it provides

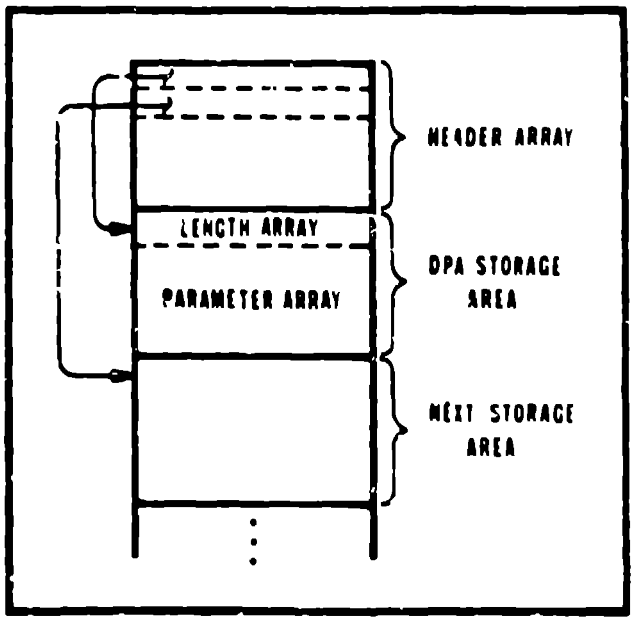


FIGURE 1
Structure of the Memory Management Region

the ability to make full use of the memory management region. Since virtual address space used to the map the region is allocated in 4K word increments, the ability to place data other than the DPA parameter array in the region allows the user to use virtual address space that might otherwise be wasted. For example, if the region were to contain only the parameter array and the parameter array required only 1000 words, approximately 3000 words would be unused of the minimum 4K words allocated to map the region. This structure allows use of the region to be based entirely upon the needs of the user without limiting him.

Creation of the Region - Data in the memory management region is accessed by means of a FORTRAN COMMON block named REGION in the task which creates the region; for example

```
    INTEGER*2 HDR(32), LEN(4)
    COMMON/REGION/HDR,LEN,I,R,II,RR,DLD,REST
```

(The variables in REGION are explained in the discussion of Subroutine PRMCRE.) The parameter array is thought of as a single array made up of four sections or sub-arrays (arrays I, R, II, and RR); first is the INTEGER*2 section, which contains all INTEGER*2 parameters the user wishes to place in the region; next are all the REAL*4 parameters; the third section contains all the INTEGER*4 parameters; and the last section contains the REAL*8 parameters. The lengths of the subsections of the array may change as the task is developed, so four words (an array named LEN) are reserved to hold the number of entries in each section of the array; i.e., LEN(1) contains the number of INTEGER*2 parameters placed in the region; LEN(2) the number of REAL*4 parameters, etc. At taskbuild time COMMON REGION is declared a virtual section with the taskbuilder VSECT option. The starting address of the virtual section is the base address of an APR chosen to allow room for the whole region to be mapped by one window. This eliminates the code as well as the time necessary to perform remapping as different storage areas in the region are referenced.

Subroutine PRMCRE is called by the task which must create the memory management region. It creates and maps the region and places some specific control data in the region; for instance, this routine finds the lengths of the subsections of the parameter array and places these values in array LEN.

Mapping the Region - When the region is created by PRMCRE, the whole region is mapped. This allows the calling task to access the parameter array as well as any other information the user wishes to put in the region.

Subroutine PRMMAP can be called by any other task needing to map the parameter array within the already existing region. In this case only the parameter array is mapped.

The functions performed by PRMMAP are as follows: The routine forms the name of the region to be mapped from the default UIC of the task calling PRMMAP. The routine attaches the region and maps the first two 32-word blocks. HDR(1) gives the starting address of the DPA storage region which, if present, will always start with the second 32-word block of the region. The first four words of the DPA storage region contain LEN, which will provide the size of the DPA storage region. PRMMAP then maps the storage region beginning with LEN(1) and continuing through the parameter array and returns status to the calling task.

Interactive Access to the Region - An interactive task PRM may be run by the user to map the region and provide the user dynamic access to the parameter values. This task provides the user the following capabilities: he may initialize all the parameters in the PRM section of the region from a disk backup file; he may make a copy of the parameter values in a disk backup file: he may define meaningful synonyms to be used in referring to specific parameters (described under Task PRM, item 7); he may assign values to individual parameters; he may display or print out the parameters, their values and their defined synonyms; he may sequentially review parameters at the terminal and selectively change values.

Execution of the commands input by the user is handled in one large loop: The command is received and parsed, the subroutine responsible for the particular command is called, any errors detected are reported at the terminal, and if no errors are found the command is executed. Changes necessary to the backup file or to the parameter array in memory are made synchronously; e.g., if a new version of the backup file is required, a separate task is spawned to create the new version. But execution of PRM waits until completion, because following commands will affect the new version of the backup file.

The Associated Disk Files - There are two disk files necessary to the DPA subsystem. One, PRM.DAT, contains the name of the disk backup file; this allows the user to change the name when appropriate and yet to have the name of the file easily available to the DPA software.

The other disk file associated with DPA is the backup file. It is a direct-access file and consists of two major sections. The first contains the parameters, one per record, consisting of the value and a pointer to the synonym defined for the parameter, if any. The second section is the synonym table; each record contains a defined synonym and a pointer back to the parameter for which it was defined.

The user may at his own option create files containing text describing the parameters in the array. There may be one file for each section of the array. If these files exist, they are used by the commands which display parameter values and synonyms (see the last paragraph).

DETAILS OF FUNCTIONALITY

This section discusses the functionality provided by the three pieces of DPA software from the point of view of the user.

Subroutine PRMCRE

In the user's task, before the call to PRMCRE, the variables in COMMON REGION cannot be assigned values, because they have virtual addresses but no physical addresses until PRMCRE maps the region. An example of the setup of REGION follows:

```
    INTEGER*2 HDR(32),LEN(4),I(65),REST(1000),ER
    INTEGER*4 II(20)
    REAL*4 R(110)
    REAL*8 RR(23)
    LOGICAL*2 FLG, DLD
    BYTE PT(6)
    COMMON/REGION/HDR,LEN,I,R,II,RR,DLD,REST
    CALL PRMCRE(FLG,PT,HDR,LEN,I,R,II,RR,DLD,ER)
```

where:

- FLG is true if the region is to be deleted from memory when the last task attached to it is detached, and false if the region is to be left in memory. The region may then be deleted from memory by use of the PRM Destroy command described later.
- PT is the ASCII representation of the name of the partition in which the region is to be created.
- HDR is the 32-word INTEGER*2 array which contains offsets to the storage sections of the region. It is passed to PRMCRE to obtain the virtual address for the start of the region.
- LEN is the 4-word INTEGER*2 array which will, following the call to PRMCRE, contain the lengths of the 4 sections of the parameter array.
- I, R, II and RR are the arrays which make up the four sections of the parameter array.
- PRMCRE will store in DLD the value of FLG, so it can be accessed and modified by other user tasks as necessary.
- REST is any other information the user wants stored in REGION.
- ER returns status to the calling routine.

Subroutine PRMMAP

PRMMAP exists to allow a secondary task to map to the DPA parameter array in an already existing memory management region without knowing a great many details of how the region may be set up. The call to PRMMAP from the user's task is as follows:

```
    COMMON/REGION/HDR,LEN,ARRAY
    CALL PRMMAP(HDR,ER)
```

where:

- HDR is as above and is passed to obtain the address of the start of the region.
- LEN is as above.
- ARRAY is dimensioned equal to or larger than the size of the whole parameter array as set up by the user's primary task which created the region. In this manner Array can be used to access the parameters without knowledge in the secondary task of the sizes of the subsections of the array.
- ER returns status to the calling routine.

Task PRM

Task PRM is an interactive task which may access the memory management region while the primary task is running. There are several commands which may be given to PRM to provide the user access to the parameters in the region. Following is a list of the functions PRM allows the user to perform.

1. Backup Parameter Values - The user can make a snapshot of the parameter values in a disk backup file.
2. Set Region to be Deleted on Last Detach - One of the parameters of the Detach Region system service allows setting whether the region should be deleted from memory upon last detach. A PRM command can be used to change this setting.
3. Make a New Version of the Backup File - This function allows creation of a new version of the backup disk file when it is necessary to change the size of the parameter array. The user's task can be edited and recompiled with no changes made to the backup file. A command can then be used to create a new version of the backup file, copying from the old backup file the parameter values still in use and the synonyms defined for those parameters, and leaving out parameters not included in the new version of the array or adding parameters with values of 0 if the new version of the array is larger.
4. Initialize Values in the Region - The parameter values in the region can be set from a backup file.
5. Scan Parameters - The user can examine and selectively alter the values of parameters at the terminal. The display at the terminal includes the parameter, its value, and any synonyms defined for it. After a parameter is displayed, the user may type a new value to be assigned to the parameter, a carriage return to display the next parameter, or a Control-Z to stop scanning.
6. Control Verification - The user can control whether confirmation must be given before values in the parameter array and in the backup file can be changed.
7. Define a Synonym for a Parameter - The user may define meaningful synonyms for individual parameters. A synonym may be defined as an array of "n" elements, in which case it is associated with the

parameter and the next n-1 parameters. An example of this is the command line

R5 := ANGLES(5)

(Parentheses are removed from the subscript of a parameter to reduce keystrokes.) Following execution of this command, the synonym for R5 is ANGLES(1); the synonym for R6 is ANGLES(2); for R7 it's ANGLES(3); etc.

The synonym definition is entered in the backup file synonym table along with the length (for a synonym array) and a pointer to the record of the backup file containing the associated parameter. The parameter may thereafter be referred to in other PRM commands by its parameter name (e.g., I6) or by its synonym.

8. Set the Value of a Parameter - The value of the parameter is changed to the new value given (following error checking to be sure the value is appropriate for the parameter type) in both the parameter array in memory and in the backup disk file.

9. Display Parameters - The user can display portions of the parameter array; an optional modifier may be used to specify the output device or file. The information displayed is the parameter name, its associated synonym, and its value. If a whole section of the array is displayed, the descriptions from the title files are also displayed.