Control Techniques for User's Global Resources

Kevin Darling
The Gap, Inc.
Eastern Distribution Center
3434 Mineola Pike
Erlanger, Ky. 41018

INTRODUCTION

In the past, we have designed and implemented systems to do a stream of tasks and only concerned ourselves about resource contention when it was necessary. The implication of this is that most processes were batch and processed serially. To run a job out of sequence or against a program currently running could impact the data and user access.

Well, systems have become more user oriented. The systems are now online and users have direct control of the input and can now see the result of their work online. To support this user interactivity, we have had to respond to the resource and data contention needs of these new applications.

To support the users, we have developed a software and data base solution that provides systems with a means to control the common data and resources. These global utilities or GUTIL routines as we will refer to them, have varying uses. They provide system developers a common set of rules and routines to complete the work and keep code common. These routines have also helped reduce the amount of coding in COBOL programs since the routines are simply called by the program. The catagories of routines available include:

- Global parameter control
- Work-file processing, including build definitions and file equations
- Output file equation definitions
- Error and warning message processing
- Non data base dependent routines

Following will be a description of the routines and data base to support these user controls.

GLOBAL PARAMETERS

Global parameters are data structures to hold fixed data and modifiable data. These parameters are generally accessed by multiple processes and features contained in global paramters allow them to be access globally and

also allow a specific process to lock the parameter exclusively.

To support the global parameter data structure, a master data set is used in a GUTIL data base. The data structure can be represented as follows:

GLOBAL-PARMS (master):

* Parameter name - This is the key to the data set. This field that contains the actual name of the parameter programs will reference.
* Access ID - This field contains the ID for the program that currently is using the data item. This ID number is assigned when the data base is opened.
* Flag - This field is to maintain the current access status of the parameter. The value determines whether there is open access, read access only or locked with no access by any other process.
* Type - This field is to determine the format the data is returned to the program. If the type is 0, it returns a single element integer. If the type is a 1, it returns a character string of up to eight characters. If the type is a 2, it returns a 2 element double integer.
* Length - The definition of the length of the data field to be returned.
* Binary data - This data item is a 2 element data area where the integer data is stored.
* ASCII data - This data item is a defined character string for storage of the character data.
* Binary min and max - These values control the range the parameter falls in. If the value becomes greater than the maximum, the routines automatically wrap the value around to the minimum.

Other data items maintained in the data structure but not critical to the actual data processing include a time stamp for the last modification and the modifier name for the process that last modified the data.

The uses for the global parameter data structure include:

* Nearest printer to an LDEV - This structure allows you to define a terminal parameter that stores the nearest printer to that terminal. Programs could dynamically access this information and route the output being requested by the user to the printer. If a change is necessary while a user is working, like to route to a different printer, the parameter could be modified and the next request for a report would reference the new printer location.

* Informative parameters to control programs - These parameters include concepts for location identification for reports and program control. The parameters could also be used to supply data for controlling programs. This data could be constant or could be very dynamic and changed periodically.
* File building key information - This structure enables you to define parameters used to build files. In our use of this concept, we have defined integer data types that get incremented from some start value through and ending value to create a unique key for data work files. If the number series reaches the maximum number, the routines automatically wrap the value around to the minimum value. Using this and the file building features we will discuss later, we create files for use by processes and maintain the key series separately for each major process.
* Dynamic data parameters - This data structure is used to hold data that could be and usually is modified frequently by many processes. Such uses include data retention values, time stamps, and parameters that provide unique key entries for data bases.

The routines that are in place to support global parameters include:

* GETGPARM - This routine gets the global parameter data and based on the type, returns the data to the calling program.
* SETGPARM - This routine sets the data value based on the type to the value that is passed to the routine.
* INCGPARM - This function increments by one the data value in the parameter. The parameter needs to be a type 0.
* DECGPARM - This function decrements by one the data value in the parameter. The parameter needs to be a type 0.
* GETGPARMSTAT - This routine returns the current status of the global parameter. This status is the flag value kept in the data base for the global parameter.

WORK-FILES

Work-files are the most complex component of the GUTIL routines system. The software supports the automatic creation, maintenance, file equation definition, and purging of these files. This is done in such a manor that the system developer only need define the characteristics and file equation for the file one time and the system is intellegent enough to resolve the processing for the file.

Work-file equations are generally pre-defined file equations that get established by the procedure for which they are established. These file equations allow some use of variable definition that the routines are intelligent enough to resolve.

The work-files themselves are files which when built can be used for any purpose. The files can also be created with user labels. The user label area is especially good for maintaining processing, restart, and data that is common to all data in the file. In the case of the restart information, the programs could use this concept to be able to help control the restart of programs in event of a program failure or graceful termination. The files themselves with the user labels and the data base entries also provides a secure version control on the files.

To support the work-file concept, there are four master data sets and three detail data sets. The data sets and their data include:

CATALOGS (master):
- * Catalog - This is the key to the data set and is a number that represents a grouping of files. These files generally have a root that is used to store and reference the file.
- * File root map - This is the actual map that is used to generate the root for the file. There could be GUTIL variables included in the definition that the serving routines resolve in order to complete the file build.
- * Description - This is simply a definition, for reference, of the defined purpose of the catalog.

FILE-CODES (master):
- * File code - This is the number that is used by the build command's CODE= parameter. It clearly defines the file type for use by other routines but also allows programmers and support personnel to easily determine the use of a file.
- * Description - This is simply a definition, for reference, of the defined purpose of the file code.

FILE-ROOT (automatic master):
- * File root - This is the actual root that gets assigned to files when the file is created. There is one entry per unique root.

PROCEDURE (master):
- * Procedure - This typically is a program or procedure name. It is used as a global reference to data so routines can do things for a specific program or routine.
- * Description - This is simply a definition, for reference, of the program or procedure and basically what it does.

WORK-FILE-BUILD (detail):
- * Catalog - This is used to define the catagory the file was created under. It allows a path into the data to display all files that are referenced by such a grouping. It is a key to the data set.

* File code - This is used to define the specific file code class the file belongs. It typically is a finer resolution as to getting specific files needed. It is a key to the data set.
* File name map - This is the definition of how the name is to be constructed if it contains variables to be resolved.
* Lock word - This is the lock word that is assigned to the file when it is built.
* Group name - This is the group in which the file is to be created.
* Record size - The length of the data record.
* Label length - The number of words that are required for the label information the file may use.
* Various MPE file create information fields including blocking factor, extents, record limit, and KSAM key information if the file is a KSAM file.

WORK-FILE-EQUATIONS (detail):
* Catalog - This is the global catagory for the file equation grouping. It is a key to the data set.
* Procedure - The program or procedure name that is used to index and reference a specific group of file equations. It is a key to the data set.
* File code - This is the code number assigned to a more specific group of files. It is a key to the data set.
* File equation - This is the actual file equation the routines use to be set up for the calling program.

WORK-FILES (detail):
* Catalog - This is the global catagory for the file grouping. It is a key to the data set.
* File code - This is the code number assigned to a more specific group of files. It is a key to the data set.
* File root - This is the actual root or name of the file as it appears on the system. It is a key to the data set.
* Date/Time created - This represents the date and time the file was created. This information is also kept in the label and is used to help maintain version control.
* File name - This is the full name of the file including the group in which the file resides.
* Label length - This represents the length of the label as it is defined.

The use of work-files can be extended from their basic use of storing user data. One extention which uses the concept of user labels includes storing common data used in processing in the label. This could reduce the amount

of data that has to be stored in each record. A second application of work-files is to use the user label for storing check points for processing. The programs would update fields in the label at critical points of the processing. If the program would fail, a simple clean-up of the data back to a check point could be done and then the program would be restarted. Upon restart, the program would first determine if it was already processing the data and where it had left off. Processing would then be set to begin at the point of interruption and the program would continue until normal completion. Use of user labels does require some tools to be written to maintain the data should modification be necessary. Such utilities could and should be user developed to meet the need of the particular application data file. But, tools like DISKED5 can be used to do the simple data editting tasks.

Most of the applications we have developed rely heavily on being able to generate unique files for processing through the day. We also use work-files that are created once and that are used over. Such applications include data base clean-up programs. The file has restart information in the label that initially starts as ready to process. The program uses the file to store the keys found in searching that are candidates for deletion. Upon completing the search the phase is set to deleting and the record number is set to the top of the file in the label. The program begins purging and updates each time the pointer. When completed with that phase, the phase is updated to the next search to be done or set to program completed for the next run. In this example, if the program fails during the search phase, it simply redoes the search when restarted. If the delete function has begun, the program gets the record number it last purged and moves on to the next record and continues processing.

To support the work-files philosophy, several routines are available. They include:

* BUILDWFILE - This routine uses the definitions defined in the work file build data set to actually build the file. If there are variables or parameters that need resolution for completing the actual build, this routine resolves them. Typically, such items that need resolution include getting a global parameter to fill in a particular value.
* DELETEWFILE - This routine deletes the work-file entry from the data base but does not purge the data file. This allows the entries that need to be searched for a program be reduced while still maintaining the data for backup.
* ENTERWFILE - This routine reads the label information of the work-file and then recreates the work-files data set entry. This is required so that the automatic version control built into the system will allow use of the data file.
* EQUATEWFILE - This routine uses the data passed to establish the

defined file equations for the specific file. It uses the definition in the work-file equations data set. And like the build function, will resolve any parameterized variables.

* GETWFILEINFO - This routines retrieves the label information and returns the data to the calling program.
* PURGEWFILE - This routine actually purges the data file and data base entry for the data set.
* SECUREWFILE - This routine logically locks the data file in the work files data set to control the access of the data file.
* SETWFILEINFO - This routine writes the label information back to the data file.
* STATUSWFILE - This routine returns the status of the data file, including the security and logical lock information.

OUTPUT FILE EQUATIONS

Output file equation control is available to control the set-up and issuing of file equations for a program of process. Before a program begins processing, it establishes the necessary file equations for referencing output devices. During the processing if there are changes or new equations that need to be set, the program can reexecute the file equations store in the data base and continue processing.

Data base support for the routine include two master and one detail data sets. One master already discussed for work-files is Procedures. The others used to support output file equations include:

FILE-FORMAL (master):
* File formal - This item is the eight character reference for the file equation which also can be seen in a SHOWOUT FNAME field for the output queue.
* Description - This item describes the file formal for reference and documentation purposes.

PRINT-FILE-EQUATION (detail):
* Procedure - This is a key entry from the master data set. It is used to be able to define all file equations for a specific program or procedure.
* File formal - This is a key entry from the master data set. It is used to be able to define a specific file equation for a program and not the entire set available.
* File equation - This field contains the actual file equation the routines use to set up the equations for the programs.

Output file equation control in a controlled environment has two purposes. The first is upon initial start-up of the program, all necessary file equations are defined for the output needs of the program. Much like work-files, the equations can be parameterized such that some resolution of the

equation may be necessary. A second use that relies even more heavily on the parameterization available is the use of one common file equation with the device ID being undefined. This value could then be resolved by either using a global parmeter that has been updated to reflect the output location or by using the global parameter NEAREST-PDEV-TO-###, where ### is the terminal requesting the output, to define the final destination.

The single routine used to support this feature is:

* EQUATEPFILE - This routine uses the procedure/file-formal to define the file equations requested. It resolves the parameterization in the file equation based on the same rules as work-file equations are defined.

WARNING AND ERROR MESSAGE PROCESSING

Warning and error message processing is available for the special processing of messages back to users, operators, and programmers. This facility enhances the error messages the HP provides to better define errors and display the data to various places in specified formats. These routines use highly parameterized data to resolve the processing of information passed by programs. Warnings are typically used for informative purposes while errors usually will be followed by the program terminating in some error state.

The data base support for message processing includes again the Procedures data set already defined and also the following:

MESSAGES (detail):

* Procedure - This is a key to this data set from the master. It is used to be able to access messages for a specific program and allow for the same error number to be stored for different programs.
* Message - This is a sorted data item which is the actual message number.
* Type - This defines whether the message is a warning or an error and what level of warning or error it is.
* Flags - This item defines information concerning the display and responce needed for a message, if any. Examples of this include definition of the display locations (eg. console, log file, or $STDLIST) and whether a console reply is necessary or not.
* Message parameters - These define up to four data parameters and the format in which to be displayed. These parameters typically contain data that should be displayed for better error resolution.
* Message text - This defines the message text, including any parameters that need to be resolved. Also included are references to the message parameters which are passed by the program.

The key use here is to consolidate the messages for systems into one place for easy maintenance. It also provides programs with functionality that can be used to better debug program errors and aborts. By storing the parameters and messages in a data base, the form of the message can be easily modified even while programs are running. This should be especially handy if a program begins to issue messages to the console which are simply informative and the operations person wishes that the messages would stop displaying. The message could be modified to be rerouted to $STDLIST or modified to better display the error data if it appears difficult to understand.

Routines available for message include:

* DUMBMSG - This routine processes messages based on information in the data base. This message can be sent to more than one location and may request operator reply. It can automatically cause an abort or simply log the message to a file and continue. If the message causes an abort, 104 words of the stack, starting with the address of the first parameter, is displayed.
* LOGABORT - This routine logs an entry to the abort log file and terminates the porgram.
* LOGSTART - This routine creates an entry in the process log file indicating that a program has started.
* LOGSTOP - This routine creates an entry in the process log file that a program has stopped.
* SMARTMSG - This routine is the same as DUMBMSG except that it checks the last message before returning the new message. If the new message is more serious than the old, it returns the message; otherwise, it does not.

SPECIAL ROUTINES FOR GUTIL

Any program that uses the routines described above that need access to the data base need to execute two other routines. The first is INITGUTIL. This routine is designed to open the data base and then initialize the data and common areas associated with using the routines. Second is RELEASEGUTIL. This routine closes the data base and releases any special resources consumed by the GUTIL routines.

NON DATA BASE DEPENDENT ROUTINES

GUTIL also includes routines that are not data base dependent. These routines are typically are for specific tasks or data manipulation. Examples of the routines included are:

* CAPCHECK - This routine uses the Local Attribute available with MPE as well as a security template to determine whether the user has the authority to use a particular function.
* STARTJCW - This routine starts a process that displays a reply message to the console and then processes the reply back for the calling program. The JCW needs to be checked by the program to determine what should be done.
* TIMESTAMP - This routine returns the date and time in a two element double integer. This is used to track modification in the GUTIL routines and it can be used by programs to apply a time stamp to user data to know when a particular update occurred.
* CDIGIT - This routine calculates check digits based on the check digit calculations available in the routine.

CONCLUSION

These routines have been used in our shop for over four years now for a variety of uses. These routines have provided our staff with common routines and data structures for the development of user systems. These routines also provide the control and checks and balances necessary for the successful operation of the systems.

ACKNOWLEDGEMENTS

I would like to acknowledge Mr. Chris Hagood for writing a significant portion of the routines discussed. And also to his documentation of the system which made it easier to use and manage.

I would also like to acknowledge our staff for helping to continue to develop new routines and enhance old ones to make the system effective for getting the job done.