The IAS-SYSTEM Data Base:
A Portable Application of the B*-Tree
(Final Report)

Harald SONNBERGER, Kurt RODLER
Klaus PLASSER and Wilfried PHILIPP

Forschungsbericht/
Research Memorandum No. 179

November 1982

This research was supported by the Austrian Science Foundation (Fonds zur Förderung der wissenschaftlichen Forschung) by grant No. 4012. Project title: Software Development in Econometrics (Software-Entwicklung in der Ökonometrie)
Die in diesem Forschungsbericht getroffenen Aussagen liegen im Verantwortungsbereich der Autoren und sollen daher nicht als Aussagen des Instituts für Höhere Studien wiedergegeben werden.
Contents

Abstract

1. Introduction

2. Requirements for the IAS-SYSTEM Data Base

3. Logical Realization of the IAS-SYSTEM Data Base
   3.1 The Data Base Environment
      3.1.1 The Application Routines
      3.1.2 The Linking Routines
   3.2 The Data Base Organization Module
      3.2.1 Access Operations for Individual Data Base Records
         3.2.1.1 The OS-File Handling Facilities
         3.2.1.2 The IAS-File Handling Facilities
      3.2.2 Access Operations for Individual Data Base Items
         3.2.2.1 Operations on Single, Fully Specified, IAS-Items
         3.2.2.2 Operations on a Set of Partially Specified IAS-Items
   3.3 The Data Base Access Module
      3.3.1 The Concepts of B-Tree and B*-Tree
      3.3.2 The Concept of a Modified B*-Tree
      3.3.2.1 The Usage of System Buffers
      3.3.2.2 Analysis of the Worst Case Behavior for Insertion and Deletion

4. Physical Realization of the IAS-SYSTEM Data Base
   4.1 The File on Unit 11
      4.1.1 The First Record
      4.1.2 The Records 2-21
      4.1.3 The Record 22
      4.1.4 The Records up from 23
4.2 The File on Unit 12

4.3 The File on Unit 13

5. Implementation of the IAS-SYSTEM Data Base
   5.1 Portability Considerations
   5.2 Parameterization Considerations
   5.3 The I/O Buffers
      5.3.1 The Numerical Buffer NUMSTR
      5.3.2 The Character Buffer CSTR
   5.4 The Interfaces to the Data Base Modules
      5.4.1 The DBORG-Interface
      5.4.2 The DBACC-Interface

Conclusions

References

Appendix
   - The Logical Structure of the Retrieval and Printing
     of Time Series including Program Listings
   - The Semiportable Routines within the IAS-SYSTEM
   - The Conversion Routines within the IAS-SYSTEM
Abstract

This paper is the final report of the first project year to the Austrian Science Foundation who has supported this project by grant No. 4012. According to the draft from March 18, 1981 the main topics of this year were the solution of problems in connection with the production of a portable software for econometrics and corporate planning, i.e. the OS-file handling and character set representation, syntax analysis, data management and data handling.

Concerning the portability problems it can be stated that recent efforts resulted in an implementation of the basis system on a Siemens computer at GMD (Gesellschaft für Mathematik und Datenverarbeitung) in Bonn and on a CDC computer at TU (Technical University) in Wien. The modified syntax analysis for the IAS-SYSTEM Level 3 will be presented independently in a forthcoming paper. This final report reviews the data base module as it has been designed and implemented during the last year. Special emphasis has been devoted to the logical and physical realization of the data base together with implementation details. In the appendix the logical structure of the programs concerning the retrieval and printing of time series is given as an example, to document the programming style and programming conventions as described in the interim report. Additionally the semi-portable routines and the conversion routines are listed documenting the efforts and the solutions to various portability problems which resulted in a program package which is portable to 99 %.

As this paper demonstrates the concrete solution to a given problem within the context of software engineering it should be of special interest for programmers and implementers of similar large software systems. Together with the syntax analysis, the utility functions and some twelve commands the currently available software forms the basis of the
IAS-SYSTEM Level 3 which is currently being tested thoroughly by the project team.

The directions of the next year go mainly to the implementation of application routines for estimation, testing, simulation, forecasting and report generation. At the end of this second project year a portable and flexible IAS-SYSTEM which is improved considerably with respect to Level 2 should be at hand.

Wien (Vienna),

November 1982

IAS-SYSTEM Project-Team

Institute for Advanced Studies
1. Introduction

The Institute for Advanced Studies has a long tradition in the development of software. To support econometric modeling and research at the Institute the development of an Interactive Simulation System called IAS-SYSTEM was started in 1974. According to the increasing number of users the IAS-SYSTEM has been improved permanently. For efficiency reasons the first two major releases were programmed in a special FORTRAN dialect. This proved to be a severe restriction, however, as soon as the Institute began to distribute the System on license contracts: it was very laborious to implement the IAS-SYSTEM on computer series other than the one the Institute was equipped with.

Within the IAS-SYSTEM Level 1 the data were stored in a straightforward linear way. This was not really a restriction for the small models that were used at that time but when people started forecasting with large econometric models the connected execution times soon exceeded certain tolerance limits.

With the further development to the IAS-SYSTEM Level 2 one of the main improvements was exactly this access organization. From this time on the retrieval, insertion and deletion of time series, equations or models was organized via a binary tree. For the maximum size of the data base consisting of 10,000 elements one could expect that about 14 accesses to secondary storage had to be performed. Although this was quite a good improvement compared to sequential search it was far from being optimal as no effort was made on balancing the binary tree. Unbalanced trees quite frequently occurred as the names of the items are used as primary key and many economists number their time series, e.g. CP, CP1,... Sometimes this caused the System to perform twenty, thirty or even more access operations which again slowed down the solution process.
One additional drawback of both levels was the user's dependence on the Data Base Administrator in case of errors in the data base. For the binary tree structure there were mainly three reasons for this, namely concurrent write accesses of different users to the same data base area, execution termination by the user during input or delete operations and the occurrence of a system crash. Although the reasons for errors were quite different the resulting errors were very much the same: due to the buffering operations of the executive system usually wrong pointers within the binary tree structure were set. In this situation one solution was to contact the Data Base Administrator to repair the binary tree. Another one was to copy the erroneous IAS-file out from the data base via a sequential search and copy it back again thus producing a new binary tree. For a data base which was nearly full this was not possible. In this case the whole data base had to be reorganized which again could be accomplished only by the Data Base Administrator.

These were the main reasons for changing the data base module within the IAS-SYSTEM. When in 1979 and 1980 the FORTRAN 77 Compiler was announced for most of the main computer series it was decided that this was the best occasion to start the conversion to a portable IAS-SYSTEM including the improvements with respect to the OS-file handling and the index organization. The final implementation of the new IAS-commands and the modified $B^*$-tree are discussed in this paper.
2. Requirements for the IAS-SYSTEM Data Base

Besides the natural requests a user usually associates with the concept of a data base like efficient data storing and comfortable data handling some additional functions exist which should be carried out by the data base management system, e.g. security, integrity, synchronization, crash protection and recovery. The objective of this chapter is not to discuss and repeat these general ideas - for that purpose the reader is referred to the standard data base literature (e.g. Ullman (1980), Date (1976)) - but to outline the special requirements that must be taken into account because of the fact that the IAS-SYSTEM Level 3, a portable program package for econometric research and corporate planning is the main user of the data base.

The additional data base functions mentioned above will be discussed throughout this paper with one exception - the synchronization. Since the whole IAS-SYSTEM and so also the data base module have to be portable, the idea of supporting concurrent accesses for read and/or write operations had to be dropped, as the FORTRAN 77 Compiler does not provide any features that will allow the implementation of that concept (e.g. READ and LOCK, TEST and SET). As a consequence the concept of the IAS-SYSTEM data base itself has changed from Level 2 to Level 3 - it has become a personal and permanent working area of one user, created and maintained by the user himself.

To determine the user's requests to the IAS-SYSTEM data base let us first look at the hierarchical data model representing the conceptual view of the IAS-SYSTEM data base.
The interpretation of Figure 1 is as follows: A data base may contain several IAS-files, one IAS-file may contain different elements, one element may have different versions and one element of a certain file with a certain version may have attached to it a certain type out of the set of \{DATA, EQU, MOD, TEXT\}. The characteristics file, element version and type/subtype together uniquely identify a certain item\(\text{*}\) within the IAS-SYSTEM and are used as primary key within the data base.

Regarding this conceptual view three different levels of data base organization can be distinguished:

a) the handling of the data base on the level of OS-files
b) the handling of IAS-files
c) the handling of IAS-items

\(\text{*}\) In the following the term IAS-item is used whenever special reference to the IAS-SYSTEM should be emphasized, otherwise the more general term data base item is used.
ad a) This new request is a direct consequence of the new data base view as a personal permanent working area of the user: to fortify the personal aspect it is necessary that each user is independent from any person in creating and maintaining his data base. On the other hand the user should not know any details about the concrete implementation of the data base, e.g. how many Operating System files a data base consists of, what size they have and so on. Particularly the possibility of changing the composition and/or the internal file names to improve the efficiency of the index organization or to meet special hardware characteristics should be provided. Therefore it was important to hide the necessary OS-file handling activities from the user. A detailed description of the OS-file facilities is given in chapter 3.2.1.1 where the new DB-command is introduced, too.

ad b) The IAS-file handling is a traditional request that has been taken over from Level 2 to Level 3 of the System because the underlying concept proved to be very useful. It raises data protection within a data base by the optional use of read/write keys and facilitates the work in connection with the default file concept. For more details see chapter 3.2.1.2 where also the new FILE-command comprising the old commands *ASG, *CAT, *FREE and *USE, is discussed.

ad c) This lowest level includes the basic data base operations like insertion, deletion, retrieval, copying and updating (see also chapter 3.2.2). Simple examples for such requests within the IAS-SYSTEM are:

*SER,I  F1.ABC  insert series ABC to file F1
*DEL,E  F2.XYZ  delete equation XYZ in file F2
For the syntax, semantic and description of these commands compare the User Reference Manual: Part One.

Usually the execution of the above commands on a fully specified IAS-item are not time critical, since they are accessing the data base only once or twice. On the other hand there are commands that may imply the access to some hundred IAS-items, e.g.

```
*MOD,S F3.DEF    solve model DEF of file F3
```

For solving a model, the model itself, each equation and each time series have to be read. As models consisting of 200 to 400 equations are quite frequent this would mean that up to '000 read operations would have to be performed only to bring the model into main storage. This example of a time critical situation shows the necessity for an efficient index organization, especially for the rapid access to IAS-item information in the narrow sense (e.g. the values of time series).

Besides operations on fully specified IAS-items the IAS-SYSTEM asks for a group of operations (deleting, copying and printing) that are working on sets of IAS-items, e.g.

```
*DEL,E F1.    delete all equations of file F1
*COPY,D F1.,F2. copy all data items of F1 to F2
```

Obviously the operation on sets is defined by the specification of a partial key. In the first example all equations of the specified file have to be deleted, the specification of an element name and a version is omitted. One solution to this partial key match problem is a search for all matching keys through the whole data base. In the literature this procedure is rejected and the implementation of secondary indices
is proposed - in the above example this would imply that there must be a set of pointers or a list directed link to all elements of a certain type - which improves the retrieval of sets of IAS-items considerably. The disadvantage of this secondary pointer structure that is stored in the according records is that in case of deletion all pointers pointing to the deleted IAS-item have to be reset, so that an advantage in case of retrieval faces a disadvantage in case of deletion.

Chapter 3.2.2.2 deals with the operations on sets of IAS-items in more detail and demonstrates that by a simple rearrangement of the key the implementation of secondary indices can be avoided and for the special IAS-SYSTEM applications an optimal support of operations on sets of IAS-items can be guaranteed.
3. Logical Realization of the IAS-SYSTEM Data Base

During the design of the data base one of the most important objectives was to achieve physical and logical data independence. While physical data independence would allow to change the index organization without affecting the logic of the application programs the logical data independence should provide a concept which is highly flexible with respect to the use by other application programs than the IAS-SYSTEM. To this end it is absolutely necessary to define unique interfaces between different levels of abstraction. Fig. 2 gives a schematic view of the IAS-SYSTEM modules.

Figure 2: The IAS-SYSTEM

![Diagram]

From Fig. 2 it can be seen that the core of this stratified structure is the data base access module called DBACC in the sequel. This module performs the access to one physical record, e.g. in case of initializing a single record by the DB-command, or to one logical record (i.e. an IAS-item), which may consist of a number of physical records. The data base organization module, from now on called DBORG has to organize the efficient,
possibly repeated callings of DBACC depending on the kind of request of the application program. It is the DBORG-module which represents the unique interface between data base environment and the data base itself.

From Figure 2 it can be seen that the application programs can be split into two groups: the first group can have direct access to the DBORG-module, while programs of the second group have to call some linking routines first. This subdivision resulted from the fact that the linking routines are needed to support the transmission of the specific information of an IAS-item to or from the data base (see chapter 3.1.2) what is heavily dependent on the type of the IAS-item - e.g. time series values, equation strings - but what is not necessary for all data base requests. Therefore the idea of a further obligatory and unique interface between application and linking routines has been dropped because of efficiency reasons. Of course it is not possible for any application or linking routine to access DBACC directly; for DBORG is the unique interface between the data base environment and the data base.

3.1 The Data Base Environment

Looking at Fig. 2 it can be seen that the IAS-SYSTEM is subdivided into two parts, the minor one which is called the utility routines module or utility commands module comprising the TIME-command or the EDIT-command etc. which have no access to the data base and the major part which consists of the application programs, the linking routines and the data base module.
3.1.1 The Application Routines

These routines represent the outmost stratum of the data base concept. An application routine is a subroutine and/or a program module requesting an access to any part of the data base, i.e. to an IAS-item, to an IAS-file, to an OS-file as a whole or a single record of it. Most of the IAS-SYSTEM commands represent program modules in that sense, like

*DB,C XY create, catalog and initialize a new data base in terms of OS-files
*FILE,C Z catalog a new IAS-file
*COPY,M Y,Z. copy all models from file Y to file Z

3.1.2 The Linking Routines

These routines are necessary for a special group of data base operations that can be characterized by the fact that it transmits buffer contents to or from the data base. This group comprises the basic data base operations insert, retrieve and update. The following commands represent valid examples.

*EQU,I ABC store equation ABC to the data base
*SER XYZ print values of time series XYZ
*UPD,A XYZ update time series XYZ in absolute differences

The problem to solve is that the information is totally dependent on the type of the IAS-item transmitted, i.e.in case of a time series this would be an ordered set of numerical values, in case of an equation this would be an algebraic expression and the corresponding polish string, possibly with estimated parameters and statistics, in case of a model this would be a string of equation names. Therefore it was necessary to find a universal mode of structuring the information so that it can be processed within the data base module without regard to the type of an actual IAS-item. These considerations resulted
in another level of abstraction, namely two generally defined I/O-buffers which are described in chapter 5.3.

Main task of the linking routines now is the preparation or interpretation of these two I/O-buffers - the NUMSTR-buffer for numerical information and the CSTR-buffer for the character information - depending on the type of the IAS-item before accessing or after having accessed the data base module.

3.2 The Data Base Organization Module

Among the many different features the DBORG-module has to provide there is one which is important for consistency checks between the IAS-SYSTEM and the accessed data base and for a simple security concept. We shall start with this concept first.

In our view a data base is a set of random access OS-files. As the user should be able to create, initialize, assign, increase and change the data base within the IAS-SYSTEM it is obvious from consistency reasons that some specific characteristic of the underlying data base like size, record length, next free record etc. has to be part of the information stored in the data base. This information is gathered in the data base status array (IDBST) - which will be described in detail in the implementation chapter. Two different types of variables of IDBST can be distinguished

- implementation dependent variables, e.g.
  - record length of OS-files
  - number of records per file

and

- the data base maintenance variables, e.g.
  - number of IAS-items stored
  - pointer to next free record
The first record of the index file (the file on unit 11, see chapter 4.1.1) was decided to contain this database specific information maintained in the array IDBST. Whenever a database is assigned this array is copied from mass-storage to main-storage thus allowing the use of different databases with one and the same IAS-SYSTEM.

Additionally the array IDBST can be used to support security concepts. Whenever one of the database maintenance variables has changed, e.g. in case of insertion or deletion, this variable is updated in the first record of the index file ensuring that no errors occur even in case of a system interrupt or a break down of the system. To reduce the resulting overhead the concept of security points has been introduced, e.g. in case of a COPY-operation of one IAS-file to another the first record of the index file is updated only once namely at the end of the whole COPY-operation, before leaving the database module.

3.2.1 Access Operations for Individual Database Records

These requests cover the features for

- the OS-file handling facilities
- the IAS-file handling facilities

3.2.1.1 The OS-File Handling Facilities

In Level 3 of the IAS-SYSTEM the user should be independent as far as possible of any Database Administrator. In the former levels of the System the Data Base Administrator had the following tasks:

- cataloging the required OS-files
- preparing an OS-procedure for the assignment of the OS-files, so that the user need not have detailed knowledge of the OS-files and their units
- initializing the data base within the IAS-SYSTEM
- reorganizing full data bases
- correcting the data base after system crashes by restoring the binary tree.

For all these tasks of the Data Base Administrator except the data base reorganization that is done automatically by the new index organization a new command has been introduced.

```
*DB,option dbname:readkey:writekey, size
```

Options:

- **C** catalog a new data base
- **D** delete the assigned data base
- **F** free the assigned data base
- **R** assign an existing data base read-only
- **S** save a data base (recovery)
- **W** assign an existing data base write-enabled
- **SPACE** like W

The data base name is a user specified name with a maximum of 18 characters, that has to match the syntactical requirements of a file name of the underlying Operating System. Read and write keys are names in the sense of the IAS terminology (see PHILIPP et al. 1982).

By the last parameter of the command the user can specify the size of the data base he intends to catalog. Enlarging an existing data base is also provided for (only in connection with option W or SPACE, when the data base is assigned write-enabled). The size specifier corresponds to the number of IAS-items the user wants to store. The possible range covers the default value 1000 to the maximum value of 10000 IAS-items - if the size specifier is no multiple of 100 it is automatically rounded to the next higher multiple of 100.
Cataloging a new data base implies the following steps

- calculate the implementation dependent variables of the data base status array with respect to the size specification
- initialize the data base maintenance variables
- catalog and assign the OS-files using the implementation dependent variables and connect them to the appropriate units
- initialize the data base by writing the free record indication pointer and the free record concatenation pointer to each record
- store the data base status array and further information like data base name, read/write keys and some statistical information to the OS-files.

The procedures for the assignment of an existing data base is very similar to the one described above. The main difference is that the data base status must be read from the data base files first and the data base specification has to be checked.

Assigning an existing data base implies the following steps

- assign the OS-files and connect them to the appropriate units
- copy the data base status array from the index file to main storage
- update the data base status array in case of data base enlargement by recalculating the implementation dependent variables
- in case of enlargement initialize the new records
- update the statistical data base information
Freeing a data base implies the following steps
- suspend the connection between the OS-files and the units
- close and keep the OS-files
- reset the IAS-file control table and the use control table to its initial stage

Deleting a data base implies the following steps
- suspend the connection between the OS-files and the units
- close and delete the OS-files
- reset the IAS-file control table and the use control table to its initial stage

The data base recovery module allows the user to start a recovery procedure in case of errors in his data base. As soon as inconsistencies are discovered within the data base he is automatically informed by the data base module indicating the necessity of a data base recovery. Here the System creates a new correct data base by trying to read sequentially the records of the erroneous data base and inserting correct IAS-items to the new data base.

Saving a data base implies the following steps
- catalog and assign a new set of OS-files, using the implementation dependent variables of the erroneous data base and connect them to the appropriate units
- initialize the data base by writing the free record indication pointer and the free record concatenation pointer to each record
- read sequentially record by record of the erroneous main files (see chapters 4.2, 4.3) and gather all records that belong to one IAS-item using forward and backward concatenation pointers
- insert correct IAS-items to the new data base rebuilding the $B^*$-tree
- print messages for erroneous IAS-items that they get lost

3.2.1.2 The IAS-File Handling Facilities

The main objectives of the introduction of the IAS-file concept are

- raising data protection
- facilitating work by the use of internal files

The data protection is realized by the possibility to attach read and/or write keys to an IAS-file to prohibit unallowed access to data or models within this file. The internal file concept is realized by the default file feature including the following arrangements (see User Reference Manual: Part One)

- base file
  the connected file is used as base file for comparison in reports, lists and tables

- data file
  all data items not provided with a file-name are taken from this file

- equation file
  as above for equations

- model file
  as above for models

- text file
  as above for text

- solution file
  when solving models the results of the endogenous variables are updated in the solution file.
Within the Level 2 of the IAS-SYSTEM the file handling was scattered over several commands

*CAT for cataloging an IAS-file
*ASG for assigning an IAS-file
*FREE for freeing an IAS-file
*USE for using, i.e. connecting an IAS-file to an internal file

According to the syntax concept of Level 3 of the IAS-SYSTEM all operations for one conceptual element should be handled with one command, e.g. the MOD-command handles the input, output, solving, updating etc. of models. For that reason the FILE-command covers all IAS-file handling operations within IAS-SYSTEM Level 3.

**FILE,option** IAS-filename:readkey:writekey

Options:

C catalog and assign an IAS-file write-enabled
F free an assigned IAS-file
R assign an IAS-file read-only
W assign an IAS-file write-enabled
SPACE like W

In addition to these basic IAS-file handling routines, the following options are available, representing the former USE-command of Level 2.

B use IAS-file as base file
D use IAS-file as data file
E use IAS-file as equation file
M use IAS-file as model file
S use IAS-file as solution file
T use IAS-file as text file
These six options may also be combined in one FILE-command, e.g. *FILE,CDT AB means catalog the IAS-file AB assign it read/write enabled and use it as data and text default file.

Cataloging a new IAS-file implies the following steps

- if not already cataloged then catalog the specified IAS-file by initializing the next free IAS-file info block with
  - reset free block indicator
  - IAS-file name
  - read key
  - write key
  - number of assignments
  - date and time of last assignment

Assigning an existing IAS-file implies the following steps

- check if the IAS-file is not already assigned, if it is already cataloged and if the keys match; if all three conditions are fulfilled then
  - update the statistics in the info block of the IAS-file (number of assignments, date/time of last assignment)
  - add (the encoded representation of) the IAS-file name to the internal file control table and the code of the read/write permission to the internal read/write permission table.

Freeing on IAS-file implies the following steps

- check if the IAS-file is assigned, then
  - delete the IAS-file name from the internal file control table, its read/write permission from the read/write permission table and update the use control table.
Using or connecting an IAS-file to an internal file, implies

- check if the IAS-file is already assigned, if not assign it
- update the use control table

The deletion of an IAS-file is not part of the FILE-command. To increase data base security by avoiding unintentional deletions by wrong option specification, this has to be done by the DEL-command.

As soon as the data base is assigned, the file control table contains at least one IAS-file namely the internal system file $SYS. It is cataloged and assigned together with the data base and contains specific IAS-items that are provided for all users of the data base. These IAS-items are assigned to that special IAS-file because of their type, examples are user-defined functions of the CALC-command.
3.2.2 Access Operations for Individual Data Base Items

The basic data base access operations for individual IAS-items (or groups of IAS-items) are

- finding an IAS-item
- reading an IAS-item
- inserting an IAS-item
- updating an IAS-item
- printing (a table of) an IAS-item (group)
- deleting an IAS-item (group)
- copying an IAS-item (group)

For all these operations, the data base organization stratum has to check the IAS-file status before performing the access. The IAS-file concept should guarantee data protection as well as simplify work with the application program, the latter fact results from the default file concept. The IAS-file check contains the following activities

- insertion of the default IAS-file to the identifier if none is specified; i.e. if an access to a time series is requested and the user has not specified the IAS-file name explicitly then the identifier is accomplished with the name of the default data file
- checking if a user specified IAS-file is assigned at all
- checking if the read/write permission of the IAS-file (specified or default) matches the requirements of the data base operation.
3.2.2.1 Operations on Single, Fully Specified, IAS-Items

The first four basic operations finding, reading, insertion and updating of an IAS-item are evidently included in that group. The other three operations printing, deleting and copying can either be operations on single, fully specified IAS-items (1 IAS-item) or a set of partially specified IAS-items. Examples for the request of a single IAS-item are the following:

*DEL,E XY.E1 delete equation XY.E1
*COPY,D A.X1,B.X1(1) copy data item A.X1 to B.X1(1)

Whenever a command works on a fully specified IAS-item, then there are no further tasks for the data base organization module than to transfer the control to the data base access stratum by calling the interface DBACC (see chapter 5.4).

3.2.2.2 Operations on a Set of Partially Specified IAS-Items

For three operations - printing, deletion, and copying - it is possible to work on sets of IAS-items, e.g.

*PRT,D AB. print information about all data items of IAS-file AB
*DEL,DEMX AB. delete all data items, equations, models and text items of IAS-file AB
*DEL AB. delete all IAS-items plus IAS-file-info block
*COPY,E K1.,K2. copy all equations from IAS-file K1 to IAS-file K2

The problem here was the organization of efficient repeated callings of the DBACC routine. Usually the printing of all time series of an IAS-file would require a search through the whole data base or the implementation of secondary indices to improve the search process. The index organization
based on a $B^k$-tree together with the effective composition of the key in the order filename-type/subtype-elementname-version make an expensive inversion routine for the data base unnecessary.

With these keys and the $B^k$-tree organization all IAS-items are already in the ordered sequence, first ordered by file, then by type/subtype and then by element and version. Therefore a sequential search on the leaves of the $B^k$-tree simulates an inversion routine perfectly. (On $B^k$-trees see chapter 3.3, on the composition of the keys see chapter 5.1).

Now the task of the data base organization stratum is to perform the necessary loops over the specified set of IAS-items and call the data base access module. The following scheme shows the processing sequence:

- check hierarchy/operation code (see chapter 5.4.1)
- check status of IAS-file
- initiate address vector for searching process
- search IAS-item matching the partial key starting at the specified addresses
- perform the data base operation for that IAS-item
- update the address vector and repeat searching.

The stop-condition in this loop is easily explained by a simple example. The command *DEL,D AB. specifies the request for deletion of all data items of IAS-file AB. The System searches for the first key, if any, with IAS-filename AB and type D(ATA). When one is found it proceeds sequentially until the type changes to E(QU) or the IAS-filename changes.
3.3 The Data Base Access Module

The main tasks of the DBACC-module are

- access to one physical data base item
- access to one logical data base item

A physical data base item is a single record of one of the data base OS-files. A logical data base item is a time series, an equation, a model or a text element which may consist of one or more physical items (records) of the main files. While the access to physical data base items is easily accomplished by specifying the appropriate record number of one of the direct access OS-files, access to a logical data base item is more complicated.

Here the DBACC-module a priori has no information which of the records of the OS-files have to be read. The necessity of an efficient index organization is evident. An index is defined to be an ordered set of pairs \((x, \alpha)\) where \(x\) is called a key and \(\alpha\) is some associated information. The key \(x\) identifies a unique element in the index, the associated information is typically a pointer to a record or a collection of records in a random access file containing the information specified by the key.

Generally it must be assumed that the index is so voluminous that only rather small parts of it can be kept in main storage at one time. Thus the index must be kept on mass storage. As the access to secondary storage takes \(10^4\) to \(10^5\) times longer than access to main storage the goal of an efficient index organization is to minimize the number of accesses to the mass storage.

Some of the main objectives for the index organization of the IAS-SYSTEM are repeated below from previous chapters:
- fast access to an IAS-item via the specified key
- self-reorganization to dissolve the dependency from the
  Data Base Administrator
- support of sequential access
- good storage utilization

Within the traditional data base literature (see KNUTH 1973, ULLMAN 1980, WEDEKIND 1974/76) some different index-organizations like heapfile organization, hashed files, indexed files, binary trees, multilevel trees (B-trees) and dense index files (B*-trees) are proposed. For the above requirements the B*-tree seemed to be a good index organization because it behaves very well in view of all requirements. One important advantage of B*-tree organization over hashing files e.g. is that it supports not only random access but also sequential access - in collating sequence by key value.

In the following chapters a brief review and discussion of the concepts of B-tree and B*-tree organization is given.

3.3.1 The Concepts of B-Tree and B*-Tree

The definition of a B-tree as pointed out in BAYER and McCREIGHT (1971) is discussed in this chapter. The following characteristics describe a B-tree.

1) Each path from the root to the leaf has the same length h, also called the height of the B-tree.

2) Each node except the root and the leaves has at least \( K+1 \) sons. The root is a leaf or has at least two sons.

3) Each node has at most \( 2K+1 \) sons.

To explain the process of retrieval, insertion and deletion a simple example of a B-tree is used where \( K=2 \), i.e. each node contains at least 2 and at most 4 keys (the associated information is omitted as this is of no interest for this explanation).
Figure 3:

To retrieve the information associated with key 9 we look at the root: as 9 is less than 12 we follow the pointer to record 2 where we find the key 9 and the corresponding information.

To insert a key we first perform a retrieval to locate the corresponding leaf. If there is place for one additional key then the new key is inserted.

The interesting problem is what happens when the leaf is already full. In our example this will be the case when we try to insert the key 20. Here the leaf has to be split into two leaves. Fig. 4 shows the resulting B-tree

Figure 4:

A similar situation can occur for the root, too. In this situation the root must be split, and a new root has to be created. Obviously this adds another level to the B-tree. Fig. 5 shows the resulting index when the key 78 is inserted in the above example.
The result is again a balanced tree, i.e. every path from the root to a leaf has the same length.

If we wish to delete the information associated with a certain key we use the retrieval procedure to find the path from the root to a node containing this key. If this key is contained in a leaf the key is deleted. If after deletion the leaf still has k or more keys we are done. If the key is contained in a node the process is more complicated. To delete this key on that node without replacing it by another key would mean that the connection to the corresponding subtree is lost, for instance deletion of key 17 in Fig. 5 means that the leaf 6 containing the keys 18 and 20 gets lost. In this situation the key 17 has to be replaced by the smallest key of the leaf 6, namely by the key 18.

In general that means that we have to retrieve the nodes down along the right pointers to the leaf and replace the key by the smallest key of that leaf. If necessary concatenations of two adjacent leaves have to be performed as in the case of deletion of the key 17. The resulting index is shown in Fig. 6.
These examples show that the B-trees grow and contract in only one way, namely nodes split off a brother or two brothers are merged or catenated into a single node. The splitting and catenation processes are initiated at the leaves only and are propagated towards the root. If the root node splits a new root must be introduced and this is the only way the height of the tree can increase.

Up to this point the information associated with a certain key was omitted. It should be apparent that by storing the information together with the keys and the pointers, the height of the B-tree would be greater than necessary. As the length of the information usually is a multiple of the length of a key, (see chapter 5.3 for the structure of NUMSTR and CSTR), a lot of space will be wasted due to the fact that many nodes contain less than 2K keys. These ideas
led to the concept of the $B^*$-tree or a dense index, compare Knuth (1973), Wedekind (1974/76).

In the $B^*$-tree concept a strict separation between the (key, pointer)-pairs and the corresponding information is observed. One file, the index file contains these pairs while the main-file contains the information. The pointers of the leaves are interpreted as pointers to records of the mainfile where the information belonging to a specified key starts. With this scheme pointers have two different meanings depending on their occurrence. The pointers in the root and the nodes which are not leaves together with their keys are only used to direct the search algorithm whereas the pointers in the leaves actually specify record addresses where the corresponding information can be found. Due to this ambiguity the keys have to be repeated in the leaves. The B-tree of Fig. 5 is equivalent to the following $B^*$-tree.

Figure 7:
If the information associated with a key is of variable length and sometimes very long - as this is the case for the IAS-SYSTEM, e.g. for time series, equations or models - the B*-tree is a more efficient index organization than a B-tree. The additional access is highly compensated by the diminished height of the B*-tree.

3.3.2 The Concept of a Modified B*-Tree

During the last years the data base philosophy within the IAS-SYSTEM has changed considerably. At the beginning of Level 2 usually one large data base containing up to 10,000 IAS-items was accessed by one or more users. The resulting problems of concurrent access led to the concept that the data base was considered to be a working and storage area belonging to only one user. For that reason the generation of smaller data bases like MINI (up to 500 IAS-items), MIDI (up to 1500 IAS-items) or MEZZO (up to 3000 IAS-items) for one user was accomplished. To extend that concept within the Level 3 an user himself should be able to create a data base of variable size between 1000 and 10,000 IAS-items. In addition to that he should have the possibility to increase an existing data base up to the maximum of 10,000 IAS-items if necessary without the necessity of rebuilding the whole B*-tree index organization.

Since it is one of the main goals of this project to reduce mass-storage accesses to a minimum to gain a maximum of processing speed it was decided to consider a B*-tree with only two levels, i.e. in the worst case there are only two accesses to the data base to find the pointers to the numerical and character contents of a certain IAS-item. A data base enlargement would then entail an increase in the number of records of the index file as well as a raise of the record length. Because of the waste of mass-storage the solution of assigning the records with a maximum length from the
beginning was discarded. However, records with variable length lead to serious problems, too

- the maintenance algorithms have a higher complexity
- in case of data base enlargement it would be necessary to reorganize the B*-tree organization on the index file

The question that arises is: "Is it possible to modify the B*-tree organization for our application so that it has minimal height of two and is general enough to support the variable data base size described above?"

Our proposal to a solution of this problem is the following. It was decided to use a fixed record length and to fill the leaves of the tree always up to the maximum number of keys and pointers. To maintain size variability only the allowed number of keys for the root is changed and the necessary leaf records on the index file and the necessary numerical and character records on the main files are initialized.

According to sizes of current econometric models 10,000 IAS-items seem to be a reasonable upper limit for the data base size. Fig. 8 outlines the situation of this maximal data base. For the two-level B*-tree this number leads to a maximal number of 100 keys per record. Here a data base is created with a root containing 99 keys directing the search algorithms and 100 leaves containing up to 100 keys.

Figure 8:
If a data base of say 2000 IAS-items is created a root containing up to 19 keys and 20 leaves containing up to 100 keys are needed. Here the record length of the index file does not depend on the size of the data base that is requested as long as the data base size lies between 1000 and 10,000 IAS-items. The only difference is the number of records which have to be initialized. For 10,000 IAS-items 101 records are used (and have to be initialized), for 2,000 IAS-items the number is 21 records, to give two examples.

As soon as the root record is full and one additional leaf has to be split the data base is considered to be full. The limits for the data base sizes are theoretical ones, in practice it will not be possible to store exactly 2000 IAS-items in a data base which is initialized for 2000 IAS-items. Because of the maintenance algorithms some of the key positions will probably remain empty.

With respect to storage utilization it can be guaranteed that the leaves are at least 50% full. To improve this storage utilization a simple overflow and underflow technique is implemented within the data base, additionally.

In case of insertion the standard algorithm would split a leaf when this leaf is full. Now instead of splitting in any case the improved algorithm looks for the possibility to move keys to the left or right neighbours of the full leaf. The leaf is only split if both neighbours are full. If one of the neighbours is not full the keys of the full leaf and the partially empty leaf are equally distributed among these two leaves. This process is called overflow technique.

In case of deletions the standard algorithm would concatenate two leaves when the leaf contains less than k keys after a deletion. Again the improved algorithm investigates the
left and right neighbours whether it is possible to move keys from one of them to the respective leaf. Only when both neighbours contain less than \( K+2 \) keys the underflow is not possible and the concatenation process will be initiated. Otherwise the keys of the two adjacent leaves will be distributed equally among them.

With these two methods, the overflow and underflow technique, a storage utilization of at least 66% can be guaranteed (see KNUTH 1973).

3.3.2.1 The Usage of System Buffers

Within the IAS-SYSTEM a key consists of the filename, element-name, version name and type/subtype. By an internal coding routine (see chapter 5.1) it is possible to store this information into four numeric storage units. The corresponding pointer value and a few words for record specific information - used/free record flag, number of keys per record, free record concatenation pointer etc. - would mean that records with a length of 510 numeric storage units are needed to implement the \( B^* \)-tree with \( K=50 \). This record length should also be a good compromise between the two conflicting objectives to minimize the number of mass storage access operations and to minimize buffer size.

Now if we try to keep the root always in main storage actually only one access is needed to find out that a specified key is or is not in the data base. If the recently used leaf is also kept in main storage many IAS-items (in the best case exactly 100) can be processed without additional access to the index. This covers the situation of sequential processing like a list of all time series of an IAS-file or the like. Consequently with a buffer of about 1k words an optimal support of sequential access is gained.
This gain of processing speed however will be reduced to some extent in case of write operations. Here it is necessary for security reasons to copy each record whether root or leaf from main storage to mass storage whenever it has changed. Otherwise it could happen that in case of a system crash whole leaves get lost, e.g. when the system crash occurs after a preceding splitting or concatenation action. For one very time critical class of write operations, namely the updating of the endogenous variables of models after simulation however, the gain of processing speed again is considerably as this action does not change the $B^*$-tree organization.

3.3.2.2 Analysis of the Worst Case Behavior for Insertion and Deletion

The worst case behavior for insertion is easily outlined by Fig. 9 and Fig. 10

Figure 9:

```
  1
 / \  
2   3
   \ /
    4
   full full full
```

We try to insert a key to record (3). As it is full we try to overflow to record (2) and then to record (4) - as both trials are unsuccessful we split record (3) and get the following $B^*$-tree.
For this worst case of insertion the following number of accesses has to be performed

2 Read-accesses to records (2) and (4)
1 Read-access to record (5)
1 Write-access to record (5)
1 Write-access to record (3)
1 Write-access to record (1) (updating of the root)

Obviously this situation seldom occurs. Only in case that a data base area is almost full - both the left as well as the right neighbour are likely to contain exactly 99 keys.

In case of deletion the worst case analysis gives the same result. Here we try an underflow from the left or right neighbour. If one of them has more than 51 keys the underflow is possible. If both have exactly 50 keys then concatenation of two leaves requesting 6 basic accesses to mass storage is necessary.

The analysis shows that even in the worst case the number of accesses to mass storage are far less than the average number of accesses for the simple binary tree organization of IAS-SYSTEM Level 2. For the simulation of large econometric models this should result in a substantial improvement of the time behavior of the data base maintenance algorithms.
4. Physical Realization of the IAS-SYSTEM Data Base

It was specified in the previous chapters that three different types of contents and information have to be stored, namely

- control information of the data base
- numerical information of data base items
- character information of data base items

It is obvious that a data base may consist of a numerical part, e.g. the single data values of a time series or the estimated parameters of an equation and a character part, e.g. the arithmetic string of the equation including the header or title that can be attached by the user. The decomposition into these two parts is a consequence of the FORTRAN 77 Standard (ANSI 1978, ISO 1980) that does not define the ratio of character storage units and arithmetic storage units. So mixing these two types of data is often forbidden, e.g. in common blocks. The decomposition into these two parts is absolutely necessary to avoid portability problems.

The control information of the data base itself can be divided into three parts

- information concerning the data base and its realization, the array IDBST (data base status array, see 4.1.1) as well as further data base specific information like read and write keys belong to this part
- information concerning the IAS-files, i.e. the IAS-file info blocks (see 4.1.2);
- information about the access path to the data base items, namely the modified B*-tree, containing all keys, the appropriate pointers and further information.
As the control information is the most important part for maintaining the IAS-data base, all three different types are gathered on one OS-file (index file) with the internal unit specifier 11. For the numeric and character information two additional OS-files (main files) with internal units 12 and 13 are initialized. The structure of these three files is discussed within the next pages.

4.1 The File on Unit 11

Fig. 11 gives a general view of the structure of the file.

**Figure 11**

<table>
<thead>
<tr>
<th>Rec #</th>
<th>Data base Control Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>Info 1. IAS-file</td>
</tr>
<tr>
<td></td>
<td>Info 11. IAS-file</td>
</tr>
<tr>
<td>21.</td>
<td>...</td>
</tr>
<tr>
<td>22.</td>
<td>Root of $B^*$-tree</td>
</tr>
<tr>
<td>23.</td>
<td>1. Leaf of $B^*$-tree</td>
</tr>
<tr>
<td></td>
<td>2. Leaf of $B^*$-tree</td>
</tr>
</tbody>
</table>
4.1.1 The First Record

This record contains the database specific control information, i.e. the array IDBST in the first 40 words and another database information block of 40 words starting in word 51. As mentioned in chapter 3.2 the array IDBST is divided into static information (words 1 to 20) which is necessary for the correct initialization of the data base and dynamic information (words 21 to 40) which changes during the insertion/deletion process.

In the following the whole list of these variables is given.

| IDBST(1) | Record length of file 11 |
| IDBST(2) | Number of records of file 11 |
| IDBST(3) | Pointer to root record of $B^k$-tree in file 11 |
| IDBST(4) | Pointer to first leaf in $B^k$-tree in file 11 |
| IDBST(5) | Max. number of keys in root |
| IDBST(6) | Max. number of keys in leaf |
| IDBST(7) | Middle key in leaf |
| IDBST(8) | Position of middle key in leaf |
| IDBST(9) | Factor for computing size of file 12 and 13 |
| IDBST(10) | Record length of file 12 |
| IDBST(11) | Number or records of file 12 |
| IDBST(12) | Record length of file 13 |
| IDBST(13) | Number of records of file 13 |
| IDBST(14) | Length of integer info of one record |
| IDBST(15) | Length of character info of one record |
| IDBST(16) | Pointer to next free record of $B^k$-tree in file 11 |
| IDBST(17) | Number of keys stored in file 11 ($B^k$-tree) |
| IDBST(18) | Number of records used in file 11 for $B^k$-tree |
| IDBST(19) | Number of catalogued IAS-files in file 11 |
| IDBST(20) | Number of elements stored in file 12 |
| IDBST(21) | Number of records used in file 12 |
IDBST(34)........
IDBST(35)........
IDBST(36)........ Pointer to next free record of file 13
IDBST(37)........ Number of elements stored in file 13
IDBST(38)........ Number of records used in file 13
IDBST(39)........
IDBST(40)........

Starting at word 51 of the first record the following information is stored

word(s)
51-55 integer-4-coded data base name, (see chapter 5.1)
56-57 integer-5-coded read key, (see chapter 5.1)
58-59 integer-5-coded write key, (see chapter 5.1)
  60 date/time of cataloging of data base
  61 date/time of current assignment
  62 date/time of last assignment
  63 number of assignments
  64 number of enlargements
  65-510 rest of the record not used

4.1.2 The Records 2-21

Each record in that range contains up to 10 IAS-file information blocks as pointed out by Fig. 11. Since the number of records used for this purpose is limited to 20 the maximum number of IAS-files for a data base is 200. The first 10 words of each IAS-file info block contain the following information:

word(s)
1-2 integer-5-coded IAS-file name
3-4 integer-5-coded read key of IAS-file
5-6 integer-5-coded write key of IAS-file
  7 validity pointer
  8 date/time of cataloging IAS-file
  9 date/time of last assignment of IAS-file
  10 number of assignments
4.1.3 The Record 22

Independent from the size of the underlying data base, this record contains the root of the $B^\ast$-tree. This root is filled with keys to the maximum value as it is specified in IDBST(6). The structure is the following:

IROOT(1) .......... Number of keys in data base root
IROOT(2) .......... Free record concatenation pointer
IROOT(3) .......... Not used
IROOT(4) .......... Not used
IROOT(5) .......... For further statistical use
IROOT(6) .......... 1st pointer (less than 1st key)
IROOT(7) .......... 1st key
IROOT(8) .......... 2nd pointer (greater than 1st key)
IROOT(9) .......... 2nd key
IROOT(10) .......... 3rd pointer (greater than 2nd key)
...

4.1.4 The Records up from 23

Starting at record number 23 each record contains one leaf of the $B^\ast$-tree. The maximum number of leaves is dependent on the size that has been specified when cataloging and initializing the data base. For the smallest possible data base 10 leaves would suffice, for the largest possible data base, 100 leaves are necessary. The following structure is valid for all leaves.
ILEAF(1)...... Number of keys in data base leaf
ILEAF(2)...... Free record concatenation pointer
ILEAF(3)...... Pointer to left neighbour
ILEAF(4)...... Pointer to right neighbour
ILEAF(5)......
ILEAF(6)......
ILEAF(7)...... For further statistical use
ILEAF(8)......
ILEAF(9)......
ILEAF(10)...... 1st pointer (less than 1st key)
ILEAF(11)......
ILEAF(12)......
ILEAF(13)...... 1st key
ILEAF(14)......
ILEAF(15)...... 2nd point (greater than 1st key)
ILEAF(16)......
ILEAF(17)......
ILEAF(18)...... 2nd key
ILEAF(19)......
ILEAF(20)...... 2nd pointer (greater than 2nd key)

There are two important differences to the root-structure. The first is the interpretation of the pointer values. Whereas within the root structure the pointers specify leaf records the meaning of pointers within leaf records is totally different. Here each pointer value is a composition of two addresses, one where the numerical information and one where the character information starts. The second difference are the concatenation pointers to the left and right neighbours of each leaf supporting sequential access as well as overflow and underflow techniques. Whenever two leaves are concatenated or one leaf is split then the corresponding concatenation pointers have to be updated; otherwise important information would be lost.
4.2 The File on Unit 12

Because of portability reasons it is necessary to divide each database item into a numerical and a character part. The file on unit 12 is provided for receiving the numerical contents. This information is highly type dependent; it is obvious that it will look different for a time series or a model. Only the first eight words of each record are kept constant to have the possibilities for some kinds of consistency checks.

numerical storage units

1 Continuation pointer to the next record, if more than one record is needed for storing the database item

2 backward concatenation pointer to ease the database recovery

3 free record concatenation pointer; points to the next unused record - the value is negative if the record is in use

4 not used (for further statistical use)

5-8 key (integer-5-coded identifier)

The rest of the record and possibly some continuation records contain the numerical contents of the database item. The appropriate constant 8 word block is repeated in each continuation record, of course.

It was decided to use a record length of 108 numeric storage units; 8 numeric storage units are needed for the fixed record specific part and 100 numeric storage units are available for the contents of the database item.

The number of records available is depending on the database size; it is the maximum possible number of keys multiplied by a constant factor (in the current implementation
this factor has the value 2) taking long numerical and/or
characterstrings into account. The resultant number of re-
cords is initialized for unit 12 as well as unit 13.

4.3 The File on Unit 13

The records of unit 13 have in principle the same structure
as the records of unit 12. Each of them starts with a fixed
record part, followed by some additional numerical informa-
tion describing the final character contents of the data
base item. It should be remarked that although this unit is
reserved for the character information it is unavoidable to
include numerical values. The first fixed part is identical
to that of unit 12 and allows consistency checks. Addition-
ally a description part for the characterstring is needed,
e.g. in case that the characterstring has to be subdivided
into several blocks - this part is limited to the fixed
length of 12 numeric storage units. So each record totals
up to 20 numeric storage units. The length of the character
information is limited to 120 character storage units per
record; continuation records are possible as for unit 12,
of course.
5. Implementation of the IAS-SYSTEM Data Base

The main objectives from the implementation point of view was to write a portable and flexible - with respect to future changes - program system (see PLASSER et al. 1982). To achieve this goal certain considerations had to be made with respect to portability, parameterization, I/O-buffers and interfaces.

5.1 Portability Considerations

The arguments which led to the selection of FORTRAN 77 as the programming language are summarized within the interim report (see PLASSER et al. 1982). Although the FORTRAN 77 supports portability to a high degree there still remain two problem areas which have to be solved, namely

- the OS-file handling, and
- the character set representation

On account of the new DB-command programs had to be written for opening and closing OS-files. Both of them have to call semiportable routines

- YOPEN for cataloging and opening
- YCLOSE for closing and deletion

of the respective OS-files; for the naming conventions of subroutines see again PLASSER et al. (1982). These nonportable "standard modules" which are used in other parts of the IAS-SYSTEM as well, e.g. for the input logfile, output logfile and message file were introduced because the OS-file
handling features within the FORTRAN 77 Standard (ANSI 1978, ISO 1980) are not as powerful as necessary. Problems arise from the fact that there is no parameter provided for the specification of the desired maximum number of records for a direct access file in the OPEN statement - on UNIVAC for example the files would always be cataloged with standard size that is too small for maintaining the data base. Another weakness of FORTRAN 77 is the INQUIRE statement which again is not as powerful as it should be for our application. To avoid problems of concurrent access it would be very useful to know if another program uses a file or uses a file exclusively, if it employs the file only for READ-operations or if it also WRITES onto the file etc..

One minor problem is the data base name. Since the IAS-SYSTEM should offer as much comfort as possible it is not desirable to restrict the user more than the Operating System when choosing a data base name. Checking a data base name for its conformity with the syntax of the Operating System is a task of the respective application program. High flexibility is achieved by concentrating this and similar checks to special semiportable routines which can be changed easily.

With respect to the character set and its representation it is well known that it is machine dependent although an ANSI/ISO standardized character set exists (ANSI 1977). For that reason all special characters (e.g. separators) are stored in a special common block and can be changed if a character is not available on a certain machine.

To avoid mixing of numeric and character variables it was decided that only numeric variables should be processed within the IAS-SYSTEM and the character representation should be used for communication purposes between the user and the System only. The consequence of this concept is that nearly each character specification entered by the user must be
transformed into a numeric, mostly integer value. Depending on the specific semantics of the input or output there are a few conversion routines for encoding and decoding character strings, e.g. encoding/decoding an identifier into/from its integer representation or encoding/decoding a time definition into/from periodicity, start period and end period. The encoding is done at interpretation time of a specific subfield or parameter by the syntax analyzer so that within the subroutines of the IAS-SYSTEM the usage of numeric representation is forced and guaranteed. The decoding is also done by a standard routine whenever necessary, in case of a communication between the System and the user.

In connection with the data base the conversion of identifiers is of main interest.
Since the user should not be too restricted in choosing his/her names for identifiers the character set contains

" coffe","A","",...,"Z","0",...,"9","€","",","".

With 39 of these 40 characters - the blank must be omitted, it must not be part of an identifier or name - the user may assemble a valid identifier, e.g. an IAS-item identifier (see PHILIPP et al. 1982)

filename up to 8 characters

type 1 character

subtype 1 character

element name up to 8 characters

version name up to 2 characters
For the representation of any of these 40 characters at least 6 bits are necessary. Assuming that the IAS-SYSTEM will run on computers with a minimum word length of 32 bits, 5 characters can be converted into a single computer word.

The standard conversion routines for encoding (character to integer) and decoding (integer to character) strings of arbitrary length according to the following collating sequence, are the subroutines ST5ENC and ST5DEC (see appendix).

" " → Ø
"A" → 1
... 
"Z" → 26
"Ø" → 27
... 
"9" → 36
"9" → 37
"&" → 38
"®" → 39

As an example the two representations of a time series of an IAS-file ECON with the element name A1 and a blank version are explained.

Character representation:

<table>
<thead>
<tr>
<th>1.word</th>
<th>2.word</th>
<th>3.word</th>
<th>4.word</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>C</td>
<td>O</td>
<td>N</td>
</tr>
</tbody>
</table>

filename | element name | version

Type "DATA"

Subtype "SER"
Numeric representation

1. word \[ 84734848 = 5 \cdot 64^4 + 3 \cdot 64^3 + 15 \cdot 64^2 + 14 \cdot 64 \]
2. word \[ 275 = 4 \cdot 64 + 19 \cdot 64^0 \]
3. word \[ 24117248 = 1 \cdot 64^4 + 28 \cdot 64^3 \]
4. word \[ \emptyset \]

Similar conversion problems arise for the handling of specific character strings like data base names, where the user is not restricted to this basic character set. Here additional symbols may appear like ",", ",", ... etc. Ideally a full ASCII-character set (ANSI 1977) should be available.

To convert these character strings the routines ST4ENC and ST4DEC have been written (see appendix). These two routines encode/decode four characters into/from a computer word.

Both of them are using the standard intrinsic functions ICHAR (for encoding) and CHAR (for decoding) of the FORTRAN processor. It should be noted that the result of calling the ICHAR or CHAR function is processor dependent.
5.2 Parameterization Considerations

With respect to the second goal - the flexibility and adaptability of the program system with respect to future changes - parameterization seems to be a powerful solution. The programming language FORTRAN 77 offers a PARAMETER-statement which is heavily used not only within the data base complex but whenever it is possible and necessary throughout the whole IAS-SYSTEM.

"The PARAMETER statement allows constants to be referenced by symbolic names. This facilitates the updating of programs in which the only changes between compilations are in the values of certain constants especially array dimension declarators. The PARAMETER statement can be revised instead of changing the constants throughout the program." (ASCII 1982)

From the static part of the array IDBST (see Chapter 4.1.1) it is obvious that the following constants should be parameterized.

<table>
<thead>
<tr>
<th>IDBST(1)</th>
<th>record length of file 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDBST(3)</td>
<td>pointer to root record</td>
</tr>
<tr>
<td>IDBST(4)</td>
<td>pointer to first leaf</td>
</tr>
<tr>
<td>IDBST(6)</td>
<td>maximal number of keys in root</td>
</tr>
<tr>
<td>IDBST(7)</td>
<td>maximal number of keys in leaf</td>
</tr>
<tr>
<td>IDBST(11)</td>
<td>factor for computing size of file 12 and 13</td>
</tr>
<tr>
<td>IDBST(12)</td>
<td>record length of file 12</td>
</tr>
<tr>
<td>IDBST(16)</td>
<td>record length of file 13</td>
</tr>
<tr>
<td>IDBST(18)</td>
<td>length of integer information of one record</td>
</tr>
<tr>
<td>IDBST(19)</td>
<td>length of character information of one record</td>
</tr>
</tbody>
</table>

All other words of the array IDBST are either variables, which are data base maintenance variables, e.g. the number of records of file 11 or the number of cataloged IAS-files - or they are not yet used.
By employing an appropriate PARAMETER-statement within the BLOCK DATA program where the parameter values are assigned to the different numeric storage units of the array IDBST it is possible to change the physical realization of the database easily by only changing this PARAMETER statement accordingly, e.g. for very large data bases with up to 20,000 IAS-items it is only necessary to change the value of IDBST(1), IDBST(5) and IDBST(7) by changing the PARAMETER statement and recompiling the whole System.

5.3 The I/O Buffers

As mentioned above the information transmitted to or from the data base can be divided into numerical values and characters. Because the FORTRAN 77 standard does explicitely not specify the relation between numeric and character storage units, it is necessary to separate these two kinds of information. The strict separation into a numerical string and a characterstring avoids portability problems.

From an efficiency point of view it was necessary to distinguish two types of character strings further. The first restricted type are the possible combinations of filename, element name, version name and type/subtype where only 40 signs are valid, namely A,...,Z,Ø,...,9 and the special signs $, &, % and blank. For these character strings IAS-SYSTEM conversion routines described in the portability chapter are used to convert them character by character to an integer value. The second full type are possible OS-file names, executable text elements like macros and general headers, using a processor dependent character set. It was decided to store this kind of information on a special character file (unit 13).
5.3.1 The Numerical Buffer NUMSTR

The structure of the I/O-buffer NUMSTR is explained by Fig. 12

Figure 12:

- LNSTR
- LISTR
- LDINFO
- NDIABLK
- LDIBK1
- DINF
- ISTR
- LDSTR
- NDDBLK
- LDSBK1
- DSTR
- LDSBK2
- 2.block
- LCINFO
- NCDQBK
- LCQBK1
- 1.block
- LCSTR
- ... 

Length of numerical string
Length of integer string
Length of data info
Number of data info blocks
Length of 1.data info block
Length of data string
Number of data string blocks
Length of 1.data string block
Length of 2.data string block
Length of character info
Number of character info blocks
Length of 1.character info block
Length of character string

The following rules are obligatory for the NUMSTR-buffer:

- the numerical string consists of the integer string ISTR and the character string information CINF
- the integer string ISTR consists of the data information DINF and data string DSTR
- each string or substring starts with its own length
- each of the three substrings - data information, data string and character information - may consist of several blocks
- each block starts with its own length and the number of its subblocks
- the length of the character information is restricted to 12 words

With these rules an existing but empty data base item, that means data information, data string as well as character information containing no information would lead to a twelve word minimal string

```
12  LNSTR  }
  7   LISTR  }
  3   LDINF  }
    1   NDIBLK  }
    1   LDIBK1  }
  3   LDSTR  }
    1   NDSBLK  }
    1   LDSBK1  }
    4   LCINFO  }
    1   NCIBLK  }
    2   LCIBK1  }
    \  LCSTRO  }
```

The numerical string for a time series containing three values for the time period 1975 to 1977 may have the following form.
32  LNST
26  LISTR
19  LDINF
 2  NDIBLK
16  LDIBK1
 1  Validity pointer
160363764  Date/Time of first input (coded)
160363764  Date/Time of last update (coded)
 0  not used
 0  Input from terminal
 1  Number of values per year
1975  Start time
1977  End time
 0  Start time { of forecasted values
 0  End time
 0  not used
 0  Aggregation mode
 1  LDIBK2
 6  LDSTR
 1  NDSBLK
 4  LDSBK1
17381195776  } Real values of time series printed as integers
17515413504
17548967936
 5  LCINF
 1  NCIBLK
 3  LCIBK1
 76  LCSTR
160363764  Date/Time of last header update (coded)

For all other IAS-items like equations, models and text similar modes for structuring the NUMSTR-buffer exist. This general procedure has proved to be very flexible for all applications.
5.3.2 The Character Buffer CSTR

In most cases the CSTR-buffer will contain the header of the respective IAS-item, where the full processor dependent character set is allowed. It can happen that the character string has to be subdivided into blocks, e.g. for executable text elements like macros. For these situations a similar structure will be used for constructing the CSTR where the necessary information is stored in the CINF of the NUMSTR-buffer.
5.4 The Interfaces to the Data Base Modules

As pointed out in chapter 3 there are two data base modules namely the data base organization module DBORG and the data base access module DBACC, both having unique interfaces. While the DBORG-module can - at least theoretically - be called by any application program, the DBACC-module must only be called by the DBORG-module.

5.4.1 The DBORG-Interface

The data base organization module communicates with the routines of the data base environment and the application programs by its interface routine DBORG with the following calling sequence.

* ... Error return
IERRCD ... Error code designating the occurred error condition
IHIRC0D ... Hierarchy code
IOPTCD ... Operation code
ICUR ... Currency indicator
IADR ... Address vector for data base access
KIDNTF ... Integer coded identifier
NUMSTR ... I/O-Buffer for numerical contents of an IAS-item
CSTR ... I/O-Buffer for character contents of an IAS-item

Within the next pages the possible combinations of IHIRC0D and IOPTCD and all other parameters are described in detail. For the I/O-buffers refer to chapter 5.3.
The Error Code

The following error codes can occur when leaving DBORG:

<table>
<thead>
<tr>
<th>Code</th>
<th>Error condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>5020</td>
<td>illegal hierarchy and/or operation code</td>
</tr>
<tr>
<td>5051</td>
<td>no IAS-file assigned at all</td>
</tr>
<tr>
<td>5160</td>
<td>specified IAS-file not assigned</td>
</tr>
<tr>
<td>5280</td>
<td>read/write permission of IAS-file does not match</td>
</tr>
</tbody>
</table>

The Hierarchy Code

This parameter is a bit combination indicating which parts of the identifier (key) have been specified by the user:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Corresponding identifier part</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ø</td>
<td>version</td>
</tr>
<tr>
<td>1</td>
<td>element</td>
</tr>
<tr>
<td>2</td>
<td>subtype</td>
</tr>
<tr>
<td>3</td>
<td>type</td>
</tr>
<tr>
<td>4</td>
<td>file</td>
</tr>
</tbody>
</table>

If the user specified the filename only, e.g. in case of *PRT,T F. then the 4th bit is set and IHIRCD has the value 2**4, which is 16. If the user specified filename and type, e.g. in case of *COPY,D F1.,F2. then the 3rd bit and 4th bit are set and the value of IHIRCD is 2**3+2**4, which is 24.

Notice that within the IAS-SYSTEM no commands are possible which specify for example only the 1st bit and the 3rd bit. If one bit is set then all other bits greater than that must be set, too. It is not possible to print all elements of a certain name over all cataloged IAS-files - this would violate the security requirements and it is not possible to get all types of a certain element name either.
The values of this bit combination defines the level (hierarchy) of the data base request

<table>
<thead>
<tr>
<th>Value</th>
<th>Affected data base level</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>OS-file</td>
</tr>
<tr>
<td>16</td>
<td>IAS-file</td>
</tr>
<tr>
<td>24</td>
<td>type</td>
</tr>
<tr>
<td>28</td>
<td>subtype</td>
</tr>
<tr>
<td>30</td>
<td>element</td>
</tr>
<tr>
<td>31</td>
<td>version</td>
</tr>
</tbody>
</table>

These hierarchy codes can be combined with certain operation codes to uniquely identify the requested operation.

The Operation Code

This parameter does make sense only in connection with the hierarchy code. The most important combinations are listed below.
<table>
<thead>
<tr>
<th>IOPTCD</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>*DB,D</td>
</tr>
<tr>
<td>3</td>
<td>*DB,F</td>
</tr>
<tr>
<td>4</td>
<td>*DB,C</td>
</tr>
<tr>
<td>5</td>
<td>*DB,R</td>
</tr>
<tr>
<td>6</td>
<td>*DB,W</td>
</tr>
<tr>
<td>11</td>
<td>*DUMP 11</td>
</tr>
<tr>
<td>12</td>
<td>*DUMP 12</td>
</tr>
<tr>
<td>13</td>
<td>*DUMP 13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IOPTCD</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>*F,B</td>
</tr>
<tr>
<td>-2</td>
<td>*F,D</td>
</tr>
<tr>
<td>-4</td>
<td>*F,E</td>
</tr>
<tr>
<td>-8</td>
<td>*F,M</td>
</tr>
<tr>
<td>-16</td>
<td>*F,S</td>
</tr>
<tr>
<td>-1024</td>
<td>*F,W</td>
</tr>
<tr>
<td>-2048</td>
<td>*F,R</td>
</tr>
<tr>
<td>-4096</td>
<td>*F,F</td>
</tr>
<tr>
<td>-8192</td>
<td>*F,C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IOPTCD</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ø</td>
<td>*PRT,C</td>
</tr>
<tr>
<td>6</td>
<td>*PRT</td>
</tr>
<tr>
<td>7</td>
<td>*DEL</td>
</tr>
<tr>
<td>8</td>
<td>*COPY</td>
</tr>
<tr>
<td>9</td>
<td>*COPY,O</td>
</tr>
<tr>
<td>10</td>
<td>*COPY,I</td>
</tr>
</tbody>
</table>
IHIRCD=31  All operations are requested for a single item

<table>
<thead>
<tr>
<th>IOPTCD</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ø</td>
<td>Find an IAS-item</td>
</tr>
<tr>
<td>1</td>
<td>Read both NUMSTR and CSTR</td>
</tr>
<tr>
<td>2</td>
<td>Read only NUMSTR(\star)</td>
</tr>
<tr>
<td>3</td>
<td>Insert an IAS-item</td>
</tr>
<tr>
<td>4</td>
<td>Update both NUMSTR and CSTR</td>
</tr>
<tr>
<td>5</td>
<td>Update only NUMSTR(\star)</td>
</tr>
</tbody>
</table>

IHIRCD≥16

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Print a table</td>
</tr>
<tr>
<td>7</td>
<td>Delete a (group of) IAS-item(s)</td>
</tr>
<tr>
<td>8</td>
<td>Copy a (group of) IAS-item(s)</td>
</tr>
</tbody>
</table>

\(\star\) This is an efficient way to accelerate the data base performance in case of model solving because all important information is stored and handled in a numerical representation within the IAS-SYSTEM. No access to the character file is necessary!

The Currency Indicator

This parameter specifies the access mode by defining the state of the address vector IADR.

<table>
<thead>
<tr>
<th>Value</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ø</td>
<td>The access is done by searching the (B^{\star})-tree</td>
</tr>
<tr>
<td>1</td>
<td>The access is done directly via the address specified in the array IADR. This possibility of access by address improves the performance considerably in case that a single command accesses an IAS-item more than once, e.g. when updating many time series during successive model solutions</td>
</tr>
</tbody>
</table>
The Address Vector

This parameter vector may contain the address of a data base item for direct data base access without searching the index.

The Integer-coded Identifier

Depending on IHIRCD/IOPTCPD the identifier might be

- a data base identifier
- an IAS-file identifier
- an IAS-item identifier

5.4.2 The DBACC-Interface

The data base access module performs the real access to a logical or a physical data base item and can be called only from the data base organization module. The calling sequence for the unique interface routine DBACC is the same as for DBORG,

* ... Error return
IERRCD ... Error code designating the occurred error condition
IHIRCD ... Hierarchy Code
IOPTCPD ... Operation code
ICUR ... Currency indicator
IADR ... Address vector for data base access
KIDNTF ... Integer coded identifier
NUMSTR ... I/O-buffer for numerical contents
CSTR ... I/O-buffer for character contents

The only difference to the DBORG-interface is that the validity ranges for a few parameters are changed.
The parameter IHIRCD is restricted to the values 16-31. Only for the case of access to a single record the value 1024 is introduced. Here the record number is transmitted by IADR.

The parameter IOPTCD may have the values for the basic database operations, i.e. 0-8 for IHIRCD=16-31 and the following six additional codes in case of record access.

11 ... read a record of file on unit 11
12 ... read a record of file on unit 12
13 ... read a record of file on unit 13
21 ... write a record to file on unit 11
22 ... write a record to file on unit 12
23 ... write a record to file on unit 13

The parameter ICUR may also have the value 2. This indicates that the address array IADR does not contain the address of the database item to be handled but a starting address for the search for the next database item matching the specified (partial) key.

The parameter KIDNTF must not contain the name of an IAS-file or a database identifier. Only database item identifiers are allowed.
Conclusions

This paper discusses the new IAS-SYSTEM data base module. Starting with various requests of the IAS-SYSTEM a detailed description of the logical and physical realization of the data base module together with implementation details is given. It should be the basis of discussion for programmers and implementers of other institutions working in similar areas as well as for the internal and external users of the IAS-SYSTEM. This paper should give the users of the IAS-SYSTEM a better feeling what really happens if they employ the IAS-SYSTEM for their data handling, estimation simulations, report generation, etc.

During the last year twelve commands have been implemented among them the basic data base operations like *DB, *FILE, *DUMP, *SER, *COPY, *DEL, *UPD and a considerably improved calculation processor, *CALC. The directions of the next year go mainly to the implementation of application routines for estimation, testing, simulation and report generation. At the end of this second project year a portable and flexible IAS-SYSTEM which is improved considerably with respect to Level 2 should be at hand.
References


PHILIPP, W., K. PLASSER, K. RODLER and H. SONNBERGER: The Syntax Analysis of the IAS-SYSTEM. Institutsarbeit, Institute for Advanced Studies, Wien, forthcoming


SPERRY UNIVAC Series 1100: FORTRAN (ASCII) Level 10R1, Programmer Reference, 1982


APPENDIX
THE LOGICAL STRUCTURE OF THE RETRIEVAL AND PRINTING OF A
TIME SERIES, INCLUDING SELECTED PROGRAM LISTINGS

SER
  SERCC
  SERBTI -
  SERSTO -
  SERPRT
  SERRD
    RDEXWS -
    SEREX
    DBORG
      DBOPEN -
      DBCLOS -
      DUMPDB -
      PRTCAT -
      FCTS
      FBRNCH -
      DBACC
        DBFIND
          KEYCHK
          KEYCOM
        DBREAD
          RDNSTR
            READ12
          RDCSTR
            READ13
          DBINP -
          DBUPD -
          DBLOCK -
          DBDEL -
          DBCOPY -
          PRTDBI -
          DELDBI -
          COPDBI -
        AGGREG
          COMTM
          AGGSAS
          AGGDEF
        SP076
        SPDATA
        SP128 -

Underlined programs are listed on the following pages;
'-' after a program name indicates a subtree, not entered
for this problem;
the remaining routines are executed, but are of minor impor-
tance.

Common utility routines (e.g. VECCOP for copying a vector of
integers) are neither mentioned nor listed.
... *** Program for controlling the processing of *SER-COMMAND ***
... *** the universal command for the handling of time series ***
... ***

... COMMENT

The checking of the command string is done in subroutine SERCC
(see there also for the allowed options and fields/subfields
and their meaning).

Depending on the specified option for printing or storing
different time definitions are taken (see comment of SERSTO).

As a consequence of the further command requirements, this
subroutine can be divided into three parts:
- insertion of time series(a) via batch-mode
  (IOPTCD=0, INPOPT=1, subroutine SERBTI)
- interactive insertion of a single time series
  (IOPTCD=0, INDPRT>1, subroutine SERSTO)
- printing of a time series
  (IOPTCD<0, subroutine SERPRT)

... OUTPUT BY PARAMETERLIST

IERRCD    I       Error code

... RETURN   Normal exit
... RETURN1  Error exit

SUBROUTINE ser(*,ierrocd)

CHARACTER*80 calstr
DIMENSION kidntf(4),kdflsr(8),iadrf(5)

ierrocd = 0

... check command string

calstr = ''
CALL vecnl(kdflsr,1,8)
CALL seroc(*980,ierrocd,ioptcd,indprtn,inpopt,kidntf,iadrf,kdflsr
   * ,iaggm,calstr)

... execute

IF(ioptcd.eq.0) THEN
C ... store option has been specified
C IF(inpopt.eq.1) THEN
C ... input from data deck in batch mode
C CALL serbti(#980,ier0cd)
ELSE
    CALL sersto(#980,ier0cd,inpopt,kidntf,iadr,kdf1er,iaaggm
*     ,calstr)
    IF(ier0cd.eq.-2) GOTO 970
ENDIF
ELSE
C ... print specified time series
C CALL serprt(#980,ier0cd,iopdet,indprt,kidntf)
ENDIF
C RETURN
C ... error section
C 970 CALL msgprt(#990,1501)
980 ierroc = ier0cd
990 RETURN1
END
*** Subroutine for reading a time series from mass storage ***

The time series is either read from the database or from the working-scratch file. The purpose of reading may either be the destination of the time range the series is defined, or the evaluation (aggregation) of the data.

**INPUT BY PARAMETERLIST**

- **KIDNTF I(4)**: Integer-5-coded time series identifier, identifier for intrinsic variable (1), (2) ... )
- **INPOPC I**: Operation code
  1 ... numeric part and character part is read
  2 ... only numeric part is read
- **INPCUR I**: Address currency indicator
  0 ... access to be done via KIDNTF
  1 ... access to be done via IADR

**INPUT BY PARAMETER STATEMENT**

- **LNSTR I**: Dimension of NUMSTR
- **LCSTR I**: Number of character storage units of CSTR

**INPUT BY COMMON/calcol/**

- **INTTBL I(NUMINT)**: Table of intrinsic variables

**INPUT BY COMMON/status/**

- **ISWITCH I(20)**: Debug switch

**OUTPUT BY PARAMETERLIST**

- **DATVEC R(*)**: Data vector containing the prepared time series
  ATTENTION: DATVEC must not be equal to NUMSTR in COMMON/DBNINF/
- **HEADER C(*)**: Header description of time series
- **MODAGG I**: Aggregation mode
  <0 ... undefined
  0 ... sum
1 ... average
2 ... stock
3 ... deflator
IERRCD I Error code

... OUTPUT BY COMMON/dbcinf/
CSTR C*LCSTR Item header

... OUTPUT BY COMMON/dbninf/
NUMSTR I(LNUMSTR) Numerical string containing the full item information

... TRANSPUT BY PARAMETERLIST
LDATVC I I: >0 ... maximum length the prepared (aggregated) time series may have
=0 ... no data requested, only the common time range
0: number of values in DATVEC
ITMDEF I(3) Time definition
I: time range desired
0: common time range of desired range and range the time series is defined for
IADR I(5) Data base or external working storage access addresses
I: in case of INPCUR=1
0: in case of INPCUR=0

... RETURN Normal exit
... RETURN1 Error exit

SUBROUTINE serrd(*,ierred,kidntf,inpoc,inpcour,datvec,header
,modagg,ldatvc,itmdef,iadr)

CHARACTER*(*) header
CHARACTER*21 qitedt
LOGICAL inddb
DIMENSION kidntf(4),datvec(*),itmdef(j),itmscr(j),iadr(5)
*kaggd(8)

INCLUDE ias-util.parcalo,LIST
INCLUDE ias-util.pdata,LIST
INCLUDE ias-util.comdbninf,LIST
INCLUDE ias-util.dcidbcinf,LIST
INCLUDE ias-util.comdbcinf,LIST
INCLUDE ias-util.comstatus,LIST
INCLUDE ias-util.dolcommsg,LIST
INCLiDE ias-util.comcommsg,LIST
INCLiDE ias-util.dclcsign,LIST
INCLiDE ias-util.comcsign,LIST
INCLiDE ias-util.dclcalc1,LIST
INCLiDE ias-util.comcalc1,LIST
ierrod = 0
iour = inpour
ioprtc = inpopo

C ... fetch time series from mass storage file (intrinsic variables)
C ... or from IAS-data-base
C
IF(kidntf(3).LT.0.OR.(iadr(1).LT.0.AND.inpcur.eq.1)) THEN
  IF(iadr(1).GE.0) THEN
    iadr(1) = -kidntf(3)
    iadr(2) = 1
    iadr(3) = lnmstr
  ELSE
    kidntf(3) = iadr(1)
    iadr(1) = -iadr(1)
  ENDIF
C
CALL rdexws(*980,ierv0cd,iadr(1),iadr(2),numstr)
CALL vecoop(numstr,4,6,itmser,1)
modagg = numstr(7)
idatab = numstr(3)+4
CALL vecnul(kaggid,1,8)
iadr(1) = -iadr(1)
inddb = .FALSE.
ELSE
  CALL serex(*980,ierv0cd,kidntf,ioprtc,iour,numstr,cstr,idatab
    ,itmser,modagg,kaggid,iadr)
C ... set error code for dummy series
C
  IF(numstr(6).EQ.0) ierrod = -1
  inddb = .TRUE.
ENDIF
C ... DEBUG of NUMSTR and CSTR
C
IF(iswtoch(4).NE.0) THEN
  msgnum = 7920
  CALL nodedb(*990,numstr,cstr,msgnum)
ENDIF
C
IF(ierrod.EQ.0) THEN
C ... no time range stored in the fetched time series
C
  IF(itmser(1).LE.0) THEN
    ierrod = 7291
    GOTO 970
  ENDIF
... data exists in time series (no dummy series)

IF(ldatvec.gt.0) THEN

... transfer (with aggregation) data to output vector DATVEC

CALL aggreg(#980,ier0cd,inidb,kidntf,itmser,modagg,kaggid
          ,rstr(idatab),datvec,ldatvec,itmdef)

ELSE

... only time range of time series is requested

IF(itmser(1).lt.itmdef(1)) THEN
  ierrod = 7311
  GOTO 970
ENDIF

CALL comtm(itmser,itmdef)
ENDIF
ENDIF

... transfer header

IF(inddb.and.inpopc.eq.1) THEN
  inhelp = LEN(header)
  IF(inhelp.gt.76) inhelp = 76
  header(1:inhelp) = ostr(1:inhelp)
ENDIF

RETURN

... error section

970 IF(inddb) THEN
  msgin = qidedt(kidntf,4)
ELSE
  msgin = intvrb(ABS(kidntf(3)))
ENDIF
msgin(22:22) = msgstp
CALL msgprt(#990,ierrod)

980 ierrod = ier0cd

990 RETURN1
The database organization module performs:

- handling of database as a whole (OS-file handling)
  . catalog
  . assign
  . free
  . delete
  . enlarge
  . recover
  . single record dump
- handling of IAS-files
  . catalog
  . assign
  . free
  . use default file
  . access permission check
- handling of IAS-items
  . set access (partial key)
    .. delete
    .. print
    .. copy
  . single item access (full key)
    .. find
    .. retrieve
    .. insert
    .. update

INPUT BY PARAMETERLIST

**IHIRCD I** Hierarchy code (bit combination)

bit
set
1 ........ version
2 ........ element
3 ........ subtype
4 ........ type
5 ........ file
do ........ data base

**IOPRTC I** Operation code, must be seen in connection with hierarchy code

IHIRCD = 0
1 ........ data base recovery (save)
2,3 ....... data base close
4-7 ....... data base open (catalog)
11-13 .... dump facility
IHIRC = 16 and IOPRTC = 0
   print list of cataloged files
IHIRC = 16 and IOPRTC < 0
   bit combination indicating the IAS-file handling operation (see subroutine FBRANCH)
   16 <= IHIRC <= 31 (operation on item set)
     6 .... print item list
     7 .... delete
     8 .... internal copy
     9 .... copy out
    10 .... copy in
IHIRC = 31 (operation on single items)
     0 .... find
     1 .... retrieve whole item
     2 .... retrieve numerical information only
     3 .... insert item
     4 .... update whole item
     5 .... update numerical information only
     7 .... delete
     8 .... internal copy
     9 .... copy out
    10 .... copy in

... INPUT BY COMMON /status/

   IDBST  I(40)   Data base status array
   (3) pointer to root record

... INPUT BY COMMON /intctt/

   IRWPRM  I(0:10)  Read/write permission of assigned IAS-files

... INPUT BY PARAMETERCOMPOUND /parrec/

   LREC14  I   Record length of file 14

... TRANSPUT BY PARAMETERLIST

   ICUR  I   Currency indicator
   if set, access is done via addresses instead of item identifier
   IADR  I(*)   Address vector for direct access without search
   KIDNTF I(*)   Integer-coded identifier,
   Integer-4-coded data base name
   Integer-5-coded IAS-file or partial/full specified item identifier
   NUMSTR I(*)   Numerical data base I/O-buffer
   in case of print-operation the coded print-size option is transmitted in (1)
   CSTR  C(*)   Character data base I/O-buffer
SUBROUTINE dborg(*, ierrod, hirod, ioprtc, icur, iadr, kidntf
   * , numstr, cstr)
C
   CHARACTER*(*) ostr
   CHARACTER*2! qldedt
   DIMENSION invalid(0:10), iadr(*), iadr1(1), kidntf(*), numstr(*)
   * , indfct(2)
   LOGICAL back11
   INCLUDE ias-util.dolcommsg, LIST
   INCLUDE ias-util.comcommsg, LIST
   INCLUDE ias-util.comcsign, LIST
   INCLUDE ias-util.comrc, LIST
   INCLUDE ias-util.comstatus, LIST
   INCLUDE ias-util.comintott, LIST
   INCLUDE ias-util.parrecl, LIST
   COMMON /idummy/ ibuff(lrec14)
   DATA invalid /j*1,2,2*0,1,2,1,2,1/
C
   ierrod = 0
C
   IF(hirod.eq.0)THEN
C   ... hierarchy code 0: operation affects whole data base
   IF(ioprtc.ge.4.and.ioprtc.lt.7) THEN
C   ... open data base

   CALL dbopen(*9900, ier0cd, ioprtc, kidntf, iadr)
   ierrod = ier0cd
   ELSE IF(ioprtc.ge.2.and.ioprtc.lt.4) THEN
C   ... close data base

   CALL dbclo(*9900, ier0cd, ioprtc, kidntf)
   ELSE IF(ioprtc.ge.11.and.ioprtc.lt.14) THEN
C   ... dump (parts) of the data base files

   CALL dumpdb(*9900, ier0cd, ioprtc, iadr)
   ELSE
C   ... error

   ierrod = 5020
   ENDIF
C
   ELSE IF(hirod.eq.16.and.ioprtc.eq.0) THEN
C   ... print table of cataloged IAS-SYSTEM-files

   CALL prtcnt(*9900, ier0cd, numstr(1))
   ELSE
C   ... search file-control-table for specified file
... get default file if none specified

IF(ioprtc.ne.9) THEN
  ind = 1
ELSE
  ind = 2
ENDIF
CALL fots(*9900,ier0cd,kidntf((ind-1)*8+1),indfct(ind))
IF(ioprtc.eq.8) THEN
  CALL fots(*9900,ier0cd,kidntf(9),indfct(2))
ELSE
  ind = MOD(ind,2)+1
  indfct(ind) = 0
ENDIF

back11 = .FALSE.
IF(inirod.eq.16.and.ioprtc.lt.0) THEN

... IAS-SYSTEM-file handling is requested

IF(ioprtc.le.-2**(13)) back11 = .TRUE.
CALL fbrncb(*9900,ier0cd,kidntf,indfct(1),ioprtc)

ELSE
  IF(indfct(1).ge.0.and.indfct(2).ge.0) THEN

  ... for each data base operation remaining the specified

  ... IAS-SYSTEM-file must be assigned and the appropriate read/write

  ... permission must satisfy the requirements of the operation

  IF(irwprm(indfct(1)).ne.3.and.
    irwprm(indfct(1)).ne.ivalid(ioprtc)) indfct(1) = -1
  IF(irwprm(indfct(2)).ne.3.and.
    irwprm(indfct(2)).ne.ivalid(ioprtc+1)) indfct(2) = -1

  IF(indfct(1).ge.0.and.indfct(2).ge.0) THEN

  ... checks are ok

  * IF(ioprtc.ge.3.and.ioprtc.le.9.and.ioprtc.ne.6)
    back11 = .TRUE.

  IF(ioprtc.lt.6.and.inirod.eq.31) THEN

  ... access (read, write, update) to a single item

  CALL dbacc(*9900,ier0cd,inirod,ioprtc,iour,iadr
    ,kidntf,numstr,ostr)

  ELSE

  ... set start values for partial key search

  iour = 2
  iadr(1) = idbst(3)
iadrl(2) = 0
iadrl(3) = 0
iadrl(4) = 10
iadrl(5) = 10

C

*C
IF(ioprtc.eq.6.and.(ihirod.lt.31.and.
ihirod.ge.16)) THEN

C

... print table of data base items

indprt = numstr(1)
CALL prtdbi(*9900,ier0cd,ihirod,ioprtc,icur
 ,iadr,kidntf,indprt,numstr,ostr)

C

ELSE IF(ioprtc.eq.7.and.(ihirod.lt.28.and.
ihirod.ge.16.or.ihirod.eq.31)) THEN

C

... delete data base item(s)

CALL deldbi(*9900,ier0cd,ihirod,ioprtc
 ,indfct(1),icur,iadr,kidntf)

C

ELSE IF((ioprtc.ge.8.and.ioprtc.1e.10).and.
(ihirod.ge.16.and.ihirod.lt.28.or.
 ihirod.eq.31)) THEN

C

... copy data base item(s) to data base item(s) either
C
... or to/from an OS-file

CALL copdbi(*9900,ier0cd,ihirod,ioprtc,icur
 ,iadr,kidntf,numstr,ostr)

C

ELSE

C

... access to data base has been tried with an illegal combination
C
... of hierarchy code and operation code

ierrocd = 5020
ENDIF
ENDIF

C

ELSE

C

... read/write permission of IAS-SYSTEM file does not match

ierrocd = 5280
ENDIF

C

ELSE IF(ier0cd.eq.5051) THEN

C

... no IAS-SYSTEM-file at all is assigned

ierrocd = ier0cd

ELSE
... the specified IAS-SYSTEM-file is not assigned
    ierrcd = 5160
    ENDF
    ENDF

... rewriting of the first record of file 11 is necessary
... to keep consistency of IDBST and the data base
    IF(back11) THEN
      iadr1(1) = 1
      CALL dbacc(*9900,ier0cd,1024,11,1,iadr1,kidntf,ibuff,cstr)
      CALL veocpo(idbat,1,40,ibuff,1)
      CALL dbacc(*9900,ier0cd,1024,21,1,iadr1,kidntf,ibuff,cstr)
    ENDF
    ENDF

... an error has occurred
    IF(ierrcd.ne.0) GOTO 9800
    RETURN

... error section

9800  IF(ierrcd.eq.5160.or(ierrcd.eq.5280) THEN

... file not assigned or read/write permission wrong
    DO 9810 i0=1,2
       IF(indfct(i0).lt.0) THEN
          magin = qidget(kidntf((i0-1)*8+1),2)//msgstp
          CALL msgprt(*9990,ierrcd)
          ierrcd = ierrcd+1
       ENDF
    9810 CONTINUE
    ELSE

... 5020 ... illegal hierarchy and/or operation code
    IF(ierrcd.eq.5020)
       WRITE(magin(1:9),FMT='(2I6,A1)') ioprtc,ihircd,msgstp
       CALL msgprt(*9990,ierrcd)
       CALL syserr(*9990,'DBORG')
    ENDF

9900  ierrcd = ier0cd
9990  RETURN
END
... *** Program to search file-control-table KFCT for a specific ***
... *** IAS-system-file ***
...

... COMMENT

If a file name is specified the routine determines it's
position in the file-control-table KFCT.
If none is specified the program determines the default file
(the type requested must be specified in the 2nd word of the
file identifier according to the common definition of a full
identifier) and the index of this file in KFCT.

... INPUT BY COMMON/status/

ISWITCH I(20) (13) Data base switch
    0 ... no data base assigned
    >0 ... data base assigned
ISTAT I(20) (11) Number of files in file-control-table

... INPUT BY COMMON/intcett/

KFCT I(2,11) File-control-table,
    contains the integer-5-coded names of all
    assigned IAS-system-files
INDUCT I(0:10) Use-control-table,
    each array element contains the index of
    the appropriate use (or default) file in the
    file-control-table KFCT
(0) .... for the IAS-system-system-file $SYS
(1) .... for the BASE-file
(2) .... for the DATA-file
(3) .... for the EQUATION-file
(4) .... for the MODEL-file
(5) .... for the SOLUTION-file
(6) .... for the TEXT-file
(7)-(10) not yet used

... OUTPUT BY PARAMETERLIST

INDFCT I Index of specified (or default) file in KFCT
IERRCD I Error code

... TRANSPUT BY PARAMETERLIST

KFILE I(2) Integer-5-coded file name
- if a file is specified on input, KFILE is a
pure input parameter
- if none is specified, the default file is
returned, for this purpose it is necessary
that the desired type is stored at the
end of the second word (see common definition
of a full identifier)

.... RETURN  Normal exit
.... RETURN1 Error exit

SUBROUTINE fots(*,ierrod,kfile,indfct)
CHARACTER*5 otype
CHARACTER*1 intfil(1:6)
DIMENSION kfile(2)

INCLUDE ias-util.comstatus,LIST
INCLUDE ias-util.comintott,LIST
DATA intfil /'B','D','E','M','S','X'/

ierrod = 0
indfct = -1

... is a data base assigned?
... if a data base is assigned, is a file assigned?

IF(iswtoch(13).eq.0) THEN
  ierrod = 5041
  CALL msgprt(*990,ierrod)
ELSE IF(istat(11).eq.0) THEN
  ierrod = 5051
  GOTO 900
ENDIF

kfil2h = MOD(kfile(2),64**2)
IF(kfile(1).eq.0) THEN
  ... get default file if no file is specified
  ktype = kfil2h/64
  kstype = kfil2h-ktype*64
  CALL st5dec(*980,ier0cd,ktype,1,otype)
  ... check internal file types
  i0 = 0
  IF(otype(5:5).eq.' ') THEN
    indfct = 1
  ELSE IF(otype(5:5).eq.'F') THEN
    indfct = 0
  ELSE
    DO 200 i0=1,6
IF(intfil(i0).eq.type(5:5)) THEN
    indfct = induct(i0)
    GOTO 300
ENDIF

200      CONTINUE
C
C ... specified file type cannot be found
C
ierroc = 5060
CALL msgprt(*990,ierrcd)
CALL syserr(*990, 'FCTS')
ENDIF

C ... determine word 1 and 2 (file,type) of full identifier
C
300       kfile(1) = kfct(1,indfct)
         kfile(2) = kfct(2,indfct)
         IF(i0.eq.2.or.i0.eq.4) THEN
C
C ... evaluate type for immediate database access
C
         CALL addts(*980,ierrcd,'D ',kfile)
         ELSE
         kfile(2) = kfile(2)+ktype*64
         ENDIF
         kfile(2) = kfile(2)+ktype

C
         ierrcd = -1
         ELSE
C
C ... search file-control-table (KFCT) for specified file
C
         kfil2h = kfile(2)-kfil2h
         DO 100 i0=0,istat(11)
             IF(kfile(1).eq.kfct(1,i0).and.kfil2h.eq.kfct(2,i0)) THEN
C
C ... found
C
             indfct = i0
             GOTO 900
         ENDIF
100      CONTINUE
ENDIF

C
900      RETURN
C
980      ierrcd = ierrcd
990      RETURN1
END
*** Subroutine for access to IAS-SYSTEM DATA BASE

... COMMENT

This is the unique interface to the data base files

... INPUT BY PARAMETERLIST

IHIRC I  data base access hierarchy code
IOPTCD I  data base access operation code

The following combinations of IHIRC and IOPTCD are possible

IHIRC=16-31  Access to elements specified by KEY
IOPTCD= 0  Find element
IOPTCD= 1  Read element
IOPTCD= 3  Insert element
IOPTCD= 4  Update element
IOPTCD= 6  Print element
IOPTCD= 7  Delete element
IOPTCD= 8  Copy element
IOPTCD= 9  Copin element
IOPTCD=10  Copout element

IHIRC=1024  Access to records specified by IADR
IOPTCD=11  Read record from system-file 11
IOPTCD=12  Read record from system-file 12
IOPTCD=13  Read record from system-file 13
IOPTCD=21  Write record to system-file 11
IOPTCD=22  Write record to system-file 12
IOPTCD=23  Write record to system-file 13

ICUR I  Currency indicator
0  No record address specified in array iadr(*)
1  Record address specified in array iadr(*)

IADR I(*)  Address array
KEY I(*)  Key of the element

... OUTPUT BY PARAMETERLIST

ICUR I  Currency indicator
0  Element not found
1  Element found (record address specified)

NSTR I(*)  String of the numerical information

CSTR C(*)  String of character information

... RETURN  normal exit
... RETURN 1  error exit
SUBROUTINE dbacc(*,ierroc,hiroc,loptod,iour,iadr,key,nstr,cstr)

CHARACTER*21 qldedt
CHARACTER(*) ostr
LOGICAL rchng,rfound,lfound
DIMENSION key(*),iadr(*),nstr(*)

INCLUDE ias-util.comstatus,list
INCLUDE ias-util.comdbrecs,list
INCLUDE ias-util.dolcommsg,LIST
INCLUDE ias-util.comcommsg,LIST
INCLUDE ias-util.dolcomsign,LIST
INCLUDE ias-util.comcomsign,LIST

ierroc=0

indhic=hiroc
indopt=loptod

IF((indhic.GE.16.AND.indhic.LE.31).AND.indopt.LE.5)THEN
  rchng=.false.

... Access to elements
...

IF(iour.EQ.0)THEN
  ... No address is specified in array iadr(*)

  CALL dbfind(*990,ierroc,indhic,indopt,key,irkpos,ilkpos,
    ipleaf,ipnstr,ipcostr,rfound,lfound)

  iadr(1) = ipleaf
  iadr(2) = ipnstr
  iadr(3) = ipcostr
  iadr(4) = irkpos
  iadr(5) = ilkpos

  IF(lfound)THEN
    iour=1
  END IF

  ... Only find specified

  IF(indopt.EQ.0)THEN
    GOTO 900
  ELSE IF(indopt.EQ.3)THEN
    ierroc=4951
    CALL msgprt(*990,ierroc)
  END IF
ELSE

  IF(indopt.EQ.0)THEN
    GOTO 900
  ELSE

  END IF
ELSE IF (indopt.NE.3) THEN
  ierrod=4961
  msgin=qidedt(key,4)//msgstp
  CALL msgprt(*990,ierrod)
END IF
END IF
ELSE
  ipleaf = iadr(1)
  ipnstr = iadr(2)
  ipcstr = iadr(3)
  irkpos = iadr(4)
  ilkpos = iadr(5)
END IF

C ... Read element from data base

IF (indopt.EQ.1.OR.indopt.EQ.2) THEN
  CALL dbread(*990,ier0od,indopt,key,ipnstr,ipcstr,nstr,cstr)
C ... Insert element to data base

ELSE IF (indopt.EQ.3) THEN
  CALL dbinp(*990,ier0od,indopt,key,irkpos,ilkpos,ipleaf,
             ipnstr,ipcstr,nstr,cstr,rchang)
C ... Update element in the data base

ELSE IF (indopt.EQ.4.OR.indopt.EQ.5) THEN
  CALL dbupd(*990,ier0od,indopt,key,ipnstr,ipcstr,nstr,cstr)
C END IF

ELSE IF ((indhir.GE.16.AND.indhir.LE.31).AND.indopt.GT.5) THEN
  C ... Partial key operation for *PRT, *DEL and *COPY

  ipleaf=iadr(1)
  irkpos=iadr(4)
  ilkpos=iadr(5)

  IF (indopt.LT.8) THEN
    CALL dblook(*990,ier0od,indhir,indopt,key,irkpos,ilkpos,ibleaf,ipnstr,ipcstr,rfound,lfound)
    IF (.NOT.lfound) THEN
      ierrod=-1
      GOTO 900
    END IF
  END IF
ENDIF
C ... Print element(s) of a certain kind
IF(indopt.EQ.6)THEN
  CALL dbread(*990,ier0cd,indopt,key(5),ipnstr,iposstr,
             nstr,ostr)
  
C
  irkpos=irkpos+5
  ilkpos=ilkpos+5
C
C ... Delete element(s) of a certain kind
C
  ELSE IF(indopt.EQ.7)THEN
    CALL dbdel(*990,ier0cd,indcpt,key(5),irkpos,ilkpos,iplleaf,
               ipnstr,ipostr,rfound,rchang)
  
C
C ... Copy element(s) of a certain kind
C
  ELSE IF(indopt.GE.8.OR.indopt.LE.10)THEN

    CALL dcopy(*999,ierrocd,inchr,indopt,key,irkpos,ilkpos,
                 iplleaf,ipnstr,ipostr,rfound,lfound,rchang)

C
C ... *COPY,I and IAS-SOF encountered ?
C
  IF(indopt.EQ.9.AND.ierrocd.EQ.-1)GOTO 900
C
  irkpos=irkpos+5
  ilkpos=ilkpos+5
C
  ELSE
    ier0cd=4520
    CALL msgprt(*990,ier0cd)
    CALL syserr(*990,'DBACC')
  END IF
C
  iadr(1)=iplleaf
  iadr(4)=irkpos
  iadr(5)=ilkpos
C
  ELSE IF(indcrt.EQ.1024)THEN
C
C ... Access to address-specified records
C
C
  IF(indopt.EQ.11)THEN
    CALL read11(*990,ier0cd,inopt,iadr(1),nstr)
  ELSE IF(indopt.EQ.12)THEN
    CALL read12(*990,ier0cd,iadr(1),nstr)
  ELSE IF(indopt.EQ.13)THEN
    CALL read13(*990,ier0cd,iadr(1),nstr,ostr)
  ELSE IF(indopt.EQ.21)THEN
    CALL writ11(*990,ier0cd,iadr(1),nstr)
  ELSE IF(indopt.EQ.22)THEN
    CALL writ12(*990,ier0cd,iadr(1),nstr)
  ELSE IF(indopt.EQ.23)THEN
  END IF
CALL writ13(*990,ier0cd,iadr(1),nstr,ostr)
ELSE
   ier0cd=4520
   CALL msgprt(*990,ier0cd)
   CALL syserr(*990,'DBACC')
END IF

ELSE
   ier0cd=4510
   CALL msgprt(*990,ier0cd)
   CALL syserr(*990,'DBACC')
END IF

IF(rochang)THEN

C ... Backwriting of the root

CALL writ11(*990,ier0cd,ibst(3),iroot)
END IF(iswitch(4).GT.0) CALL msgprt(*990,4890)

900 RETURN
C 990 ierroc=ier0cd
999 RETURN 1
C
END
**Subroutine for finding a specified key in data base**

**INPUT BY PARAMETERLIST**

**KEY** I(*) Specified key

**INPUT BY COMMON/STATUS/**

- **IDBST** I(40)  IDBST(3) Pointer to root of B*-tree
- **IDBST** I(40)  IDBST(4) Pointer to first leaf in B*-tree
- **IDBST** I(40)  IDBST(6) Maximal number of keys in root
- **IDBST** I(40)  IDBST(7) Maximal number of keys in leaf
- **IDBST** I(40)  IDBST(21) Pointer to next free record of file 11

**OUTPUT BY PARAMETERLIST**

- **IRKPOS** I Position of the according pointer in root
- **ILKPOS** I Position of the according pointer in leaf
- **IPEAF** I Pointer to leaf
- **IPNSTR** I Pointer to numerical string
- **IPCSTR** I Pointer to character string
- **RFOUND** L true if key is stored in root
- **FOUND** L true if key is found in data base

**RETURN** normal exit

**RETURN 1** error exit

**SUBROUTINE** dbfind(*,ierrocd,indhir,indopt,key,irkpos,ilkpos,*  ipleaf,ipnstr,ipcstr,rfound,found)

**LOGICAL** found,rfound,root

**DIMENSION** key(*)

**INCLUDE** ias-util.comstatus,LIST

**INCLUDE** ias-util.comdbrecs,LIST

ierrocd=0
IF(indhir.NE.31)THEN  
ierrocd=4510
CALL msgprt(*990,ierrocd)
CALL syserr(*990,′DBFIND′)
END IF

**Initialization of leaf-pointer**

ipleaf=0
IF((idbst(21),EQ.idbst(4).AND.iroot(1).LE.idbst(7)).OR.  
   (iroot(1).LE.idbst(6)))THEN
  
C ... Search for key in root

C  
root=.true.
   CALL keychk(*999,iroot,root,key,indhir,irkpos,ipoint,rfound)

C IF(idbst(21).EQ.idbst(4))THEN

C ... Only the root-record contains pointers

C IF(rfound)THEN
   found=.true.
   CALL ptrdeo(*990,ipoint,ipnstr,ipcstr)
ELSE
   found=.false.
   ipleaf=idbst(3)
   ipnstr=idbst(31)
   ipcstr=idbst(36)
END IF
GOTO 900
ELSE
   ipleaf=ipoint
END IF
ELSE
   ierOcd=4530
   CALL msgprt(*990,ierOcd)
   CALL syserr(*990,‘DBFIND’)
END IF

C ... Read the specified leaf

C CALL read11(*990,ierOcd,indopt,ipleaf,ileaf)

C IF(ileaf(1).LE.idbst(7))THEN

C ... Search for key in leaf

C root=.false.
   CALL keychk(*990,iroot,root,key,indhir,ilkpos,ipoint,found)

C IF(found)THEN
   CALL ptrdeo(*990,ipoint,ipnstr,ipcstr)
ELSE
   ipnstr=idbst(31)
   ipcstr=idbst(36)
END IF
ELSE
   ierOcd=4540
   CALL msgprt(*990,ierOcd)
   CALL syserr(*990,‘DBFIND’)
END IF

900 RETURN
IERROD=IER0AD
RETURN 1
END
*** Subroutine for checking whether a specified key ***
*** is in a specified record of file 11 ***

INPUT BY PARAMETERLIST

ISTRNG I Record-string to be searched
ROOT L is true if key is searched in root
KEY I Key of the element
NRKEY I Maximal number of keys in ISTRNG

INPUT BY COMMON/STATUS/

IDBST I(40) IDBST(4) Pointer to first leaf in B*-tree
IDBST(21) Pointer to next free record of file 11

OUTPUT BY PARAMETERLIST

KPOS I Start position of key or right key
PNTR I Pointer to the next record in B*-tree
FOUND L is true if the specified key is in the record

RETURN Normal exit
RETURN 1 Error exit

SUBROUTINE keychk(*,istrng,root,key,indhir,kpos,ipntr,found)

LOGICAL found,root
DIMENSION key(4),istrng(*)
INCLUDE ias-util.comstatus,LIST

found=.false.
ianf=10
iend=5*istrng(1)+6

DO 160 i=ianf,iend,5

CALL keycom(key,indhir,istrng,i,kswtch)
IF(kswtch.GT.0)THEN
   GOTO 160
ELSE
   kpos=i
   ipntr=istrng(kpos)
   IF(kswtch.EQ.0)THEN
      found=.true.,
      CALL vecoop(istrng,i+2,i+2,key,2)
   END IF
   GOTO 900
END IF
160 CONTINUE
C ... The searched key is the largest in the specified record
C
    kpos=end+4
    IF(root)THEN
        IF(idbst(21).EQ.idbst(4))THEN
            ipntr=0
        ELSE
            ipntr=ist rng(kpos)
        END IF
    ELSE
        ipntr=0
    END IF
    found=.false.
900 RETURN
C
990 RETURN 1
C
END
... *** Subroutine for comparing partial keys in data base *** ...

... COMMENT ...

... The bits of indhir set reflect the following comparisons ...

... Bit 5: Filename
... Bit 4: Type
... Bit 3: Subtype
... Bit 2: Element
... Bit 1: Version

... INPUT BY PARAMETERLIST ...

KEY I(*) Specified key
INDHIR I Data base access hierarchy code
ISTRNG I(*) ROOT or LEAF
ISTART Start position for Compare-Operation

... OUTPUT BY PARAMETERLIST ...

KSWITCH I Switch indicating the relation between partial key and key of istrng

... RETURN normal exit
... RETURN 1 error exit

SUBROUTINE keycom(key,indhir,istrng,istart,kswtch)
DIMENSION key(*),ISTRNG(*)
indacc=MOD(indhir,16)

... For *COPY,I file name may not be specified, e.g. *COPY,I ,X.
IF(key(1).EQ.0)GOTO 100
... Test for Filename
IF(key(1).LT.ISTRNG(ISTART+1))GOTO 200
IF(key(1).GT.ISTRNG(ISTART+1))GOTO 400
kohg=(key(2)/64**2)*64**2
ichg=(ISTRNG(ISTART+2)/64**2)*64**2
IF(kohg.LT.ICHG)GOTO 200
IF(kohg.GT.ICHG)GOTO 400
C ... Test for Type
C
100 IF((indacc/8).EQ.1) THEN
    kchg=((key(2)-kcon)/64)*64
    ichg=((istrng(istart+2)-ichg)/64)*64
    IF(kcon.LT.ichg)GOTO 200
    IF(kchg.GT.ichg)GOTO 400
    indacc=MOD(indacc,8)
END IF
C
C ... no test for subtype!
C
    indacc = MOD(indacc,4)
C
C ... Test for Element
C
    IF((indacc/2).EQ.1) THEN
        IF(key(3).LT.istrng(istart+3))GOTO 200
        IF(key(3).GT.istrng(istart+3))GOTO 400
    END IF
C
C ... Test for Version
C
    IF(indacc.EQ.1) THEN
        kchg=(key(4)/64**2)*64**2
        ichg=(istrng(istart+4)/64**2)*64**2
        IF(kchg.LT.ichg)GOTO 200
        IF(kchg.GT.ichg)GOTO 400
        indacc=MOD(indacc,2)
    END IF
C
C ... Test for Version
C
    IF(indacc.EQ.1) THEN
        kchg=key(4)-(key(4)/64**2)*64**2
        ichg=istrng(istart+4)-(istrng(istart+4)/64**2)*64**2
        IF(kchg.LT.ichg)GOTO 200
        IF(kchg.GT.ichg)GOTO 400
    END IF
K
kswtch=0
GOTO 900
200 kswtch=-1
GOTO 900
400 kswtch=+1
900 RETURN
END
**Subroutine for reading an element specified by key**

**INPUT BY PARAMETERLIST**

- INDOPT I Data base access operation code
- KEY I(4) Specified key
- IPNSTR I Pointer to numerical string
- IPCSTR I Pointer to character string

**OUTPUT BY PARAMETERLIST**

- NUMSTR I(*) Numerical string to be read
- CSTR C(*) Character string to be read

**RETURN**

- normal exit
- error exit

**SUBROUTINE** dbread(*,ier0cd,indopt,key,ipnstr,ipostr,numstr,cstr)

**CHARACTER(*) cstr**

**DIMENSION** key(*),numstr(*)

**INCLUDE** ias-util.comstatus,LIST

ier0cd=0

**Initialize length of numerical string and character string**

- numstr(1)=2
- numstr(2)=1

IF(indopt.LE.2.OR.indopt.GE.6)THEN

**Read numerical-string**

CALL rdnstr(*990,ier0cd,key,ipnstr,numstr)
IF(iswtoch(4).GT.0) CALL msgprt(*990,4710)

IF(indopt.ne.2)THEN

**Read character-string**

CALL rdcstr(*990,ier0cd,key,ipostr,numstr,cstr)
IF(iswtoch(4).GT.0) CALL msgprt(*990,4700)
END IF

ELSE

ier0cd=4520
CALL msgprt(*990,ier0cd)
CALL syserr(*990,'DBREAD')
END IF
RETURN
990  ierred=ier0cd
RETURN 1
C
END
... *** Subroutine for reading the numerical string of file 12 ***
...

... INPUT BY PARAMETERLIST

KEY   I(4)    Key of the element
IPNSTR I       Pointer to the next free record in file 12

... OUTPUT TO PARAMETERLIST

NUMSTR I(*)   Numerical string

... INPUT BY COMMON/STATUS/

IDBST   I(40) IDBST(12) Record length of file 12

... RETURN normal exit
... RETURN 1 error exit

SUBROUTINE rdnstr(*,ierroc,key,ipnstr,numstr)

LOGICAL jkequ
DIMENSION key(*),numstr(*)
INCLUDE ias-util.comstatus,LIST
COMMON/idummy/istack(510),ibuff(108),infohr(20),infdum(20)

ierroc=0
icon=0
ibeg=2

200 CALL veconul(ibuff,1,idbst(12))
     CALL read12(*990,ierocc,idpstr,ibuff)

... Testing for right key

IF(jkequ(key,ibuff,4))THEN

... Create initial information from the first record

IF(icon.EQ.0)THEN
    listr=ibuff(9)
    ires =listr
    ircl =idbst(12)-8
END IF

ilen=MIN(ircl,ires)
iend=ilen+8
CALL veccop(ibuff,9,iend,numstr,ibeg)
C ... Continuation cards ?
C
ires=ires-ilen
IF(ires.GT.0) THEN
   icon=icon+1
   ibeg=ibeg+ilen
   ipnstr=ibuff(1)
   GOTO 200
ELSE
   numstr(1)=listr+1
   RETURN
END IF
ELSE
   ierrod=4570
   CALL msgprt(*990,ierrod)
   CALL syserr(*990,"RDNSTR")
END IF
C
990  ierrod=ier0cd
RETURN 1
C
END
... *** Program for reading one record of file 12 ***

... INPUT BY PARAMETERLIST

IREC I Record number to be read

... INPUT BY COMMON/STATUS/

IDBST I(40) IDBST(12) Record length of file 12
IDBST(13) Number of records of file 12

... OUTPUT BY PARAMETERLIST

IERRECD I Error code
ISTR I Contents of the specified record

... RETURN Normal exit
... RETURN 1 Error exit

... SUBROUTINE read12(*,ierrocd,irec,istr)

DIMENSION istr(*)
INCLUDE ias-util.comstatus,LIST

ierrocd=0
lstrng=idbst(12)

IF(irec.GT.0.AND.irec.LE.idbst(13))THEN
  READ(12,REC=irec,IOSTAT=ier0cd,ERR=900)(istr(i),i=1,lstrng).
ELSE
  CALL syserr(*990,ˊREAD12ˊ)
END IF
RETURN

900 ierrocd=ier0cd
990 RETURN 1
END
... *** Subroutine for aggregation of time series ***

... COMMENT

This routine aggregates time series, if necessary. Four different aggregation procedures are available:
Code <0 ... aggregation mode is not defined
Code 0 ... aggregate by summing up
Code 1 ... aggregate by averaging
Code 2 ... take last value of time range as aggregate (for stocks)
Code 3 ... deflator aggregation (e.g. deflator defined as (NOMINAL SERIES)/(REAL SERIES)*100, aggregation is done by aggregating both series according to their aggregation mode - accept for another aggregation mode 3 because of recursion - and repeating the calculation mentioned above.

This aggregation is only possible for time series read from the data base. Evidently, also the nominal and the real series must be items of the data base and the needed IAS-files must be assigned. For this reason deflator aggregation is not possible for intrinsic variables series.

Disaggregation of time series is not possible and therefore inhibited.

... INPUT BY PARAMETERLIST

INDDB L Indicator for time series origin
.T. ... series is stored in data base
.F. ... series is intrinsic variables

KIDNTF I(4) INDDB = .T. ... integer-5-coded item identifier
INDDB = .F. ... (1)-(2) ... 0
(3) ........ index of intrinsic variable in intvtb
(4) ........ 0

ITMSER I(3) Time range for which the time series is defined

MODAGG I Aggregation mode (see above in COMMENT)

KAGGID I Contains the identifier of the nominal and the real time series for deflator aggregation
(1)-(4) ... integer-5-coded identifier of the nominal time series
(5)-(8) ... integer-5-coded identifier of the real time series

RINPVC I(*) Data input vector

... INPUT BY COMMON/calc1/

INTVTB I(NUMINT) Table of intrinsic variables
INPUT BY COMMON/comstatus/

ISWITCH I(20)   (4) ... Debug switch
(9) ... Batch mode switch

OUTPUT BY PARAMETERLIST

AGGVEC I(*)   Aggregated time series
LAGGVC I      Length of aggregated time series
IERRCD I      Error code

OUTPUT BY COMMON/miss/

MISSB I       Number of missing values at the begin
               of the aggregated time series
MISSE I       Number of missing values at the end
               of the aggregated time series

TRANSPUT BY PARAMETERLIST

ITMDEF I      Time definition
I: time range requested
0: time range possible, when combining  
   requested time range and time range  
   of the time series

RETURN        Normal exit
RETURN1       Error exit

SUBROUTINE aggreg(*,ierrod,inddb,kidntf,itmser,modagg,kaggid  
   ,rinpvc,aggvec,laggvc,itmdef)

CHARACTER*21 qidedt
LOGICAL inddb
DIMENSION kidntf(4),itmser(3),itmdef(3),itmcom(3),kaggid(8)  
   ,rinpvc(*),aggvec(*)

INCLUDE ias-util.parcalc,LIST
INCLUDE ias-util.comstatus,LIST
INCLUDE ias-util.dolcommsig,LIST
INCLUDE ias-util.comcommsg,LIST
INCLUDE ias-util.dolstatus,LIST
INCLUDE ias-util.comcsign,LIST
INCLUDE ias-util.commiss,LIST
INCLUDE ias-util.dolcalc1,LIST
INCLUDE ias-util.comcalc1,LIST
ierrod = 0
logadr = ABS(kidntf(3))

C ... disaggregation is not possible
C
IF(itmser(1).lt.itmdef(1)) THEN
  ierrcd = 7311
  IF(inddb) THEN
    msgin = qidedt(kidntf,4)
  ELSE
    msgin = intvtb(logadr)
  ENDIF
  msgin(22:22) = msgstp
  CALL msgprt(*990,ierrcd)
ENDIF

C ... evaluate common time range
C
CALL vecoop(itmdef,1,3,itmcom,1)
CALL contmn(itmser,itmcom)

C ... DEBUG of time ranges
C
IF(iswtch(4).ne.0) THEN
  CALL msgmul(*990,7410,7430)
  WRITE(msgin,FMT='(I2,2(I2,I10),2(2X,I2,2(I1X,I10)),A1)')
  *    itmdef,itmser,itmcom,msgstp
  CALL msgprt(*990,0)
ENDIF

C nditem = itmcom(3)-itmcom(2)+1
IF(nditem.gt.laggvc) THEN

C ... output vector AGGVEC can't hold all values
C
  ierrcd = 6601
  CALL msgprt(*990,ierrcd)
ENDIF

C ... evaluate aggregation ratio, missing start and end values,
C ... begin and end of desired data string in input vector RINPVVC
C
  laggvc = nditem
  nquot = itmser(1)/itmcom(1)

C IF(itmdef(2).eq.0) THEN
  missb = 0
  ELSE
    missb = itmcom(2)-itmdef(2)
  ENDIF
C
C IF(itmdef(3).eq.0) THEN
  misse = 0
  ELSE
    misse = itmdef(3)-itmcom(3)
  ENDIF
C
  idatab = itmcom(2)*nquot-itmser(2)+1
nditem = nditem*nquot
idatae = idatab+nditem-1

CALL vecop(itmcom,1,3,itmdef,1)

IF(nquot.eq.1) THEN

... no aggregation necessary, time series is copied

CALL vecpr(rinpvc,idatab,idatae,aggvec,1)
ELSE

IF(modagg.ge.0.and.modagg.lt.3) THEN

... aggregation mode is sum, average or stock

CALL aggsum(modagg,idatab,idatae,nquot,rinpvc,aggvec)
ELSE IF(modagg.eq.3) THEN

... aggregation mode is deflator

IF(.not.inddb) THEN

msgin = intvrb(logadr)//msgstp
ierrocd = 7321
CALL msgsprt(*990,ierrocd)
ELSE IF(kaggid(3).eq.0.or.kaggid(7).eq.0) THEN

ierrocd = 7340
CALL msgsprt(*990,ierrocd)
ELSE

CALL aggedf(ierrocd,kaggid,aggvec,itmcom)
ENDIF

IF(ierrocd.gt.0) THEN

msgin = qidedt(kidntf,4)//msgstp
CALL msgsprt(*990,7351)
ENDIF
ELSE

... aggregation mode is not defined or not valid

IF(modagg.lt.0) THEN

ierrocd = 7391
ipos = 1
ELSE

ierrocd = 7030
WRITE(msgin,FMT=’(I1)’) modagg
ipos = 2
ENDIF

IF(inddb) THEN

msgin(ipos:ipos+20) = qidedt(kidntf,4)
ELSE

msgin(ipos:ipos+20) = intvrb(logadr)
ENDIF

ipos = ipos+21
msgin(ipos:ipos) = msgstp
CALL msgsprt(*990,ierrocd)
IF(modagg.ge.0) CALL syserr(*990,'AGGREG')
ENDIF

C ...
C DEBUG of original and aggregated time series
C
IF(iswtdch(4).ne.0) THEN
  CALL msgmul(*990,7430,7450)
  ncomma = 3
  IF(iswtdch(9).eq.0) THEN
    nvalrw = 5
  ELSE
    nvalrw = 10
  ENDIF
  CALL rvnp(*990,rinpv,ifdta,ndtem,nvalw,ncomma)
  CALL msgpr(*990,10)
  CALL rvnp(*990,aggvec,1,laggvc,nvalw,ncomma)
  CALL msgpr(*990,10)
ENDIF

C
C RETURN

C
990 RETURN1
END
*** Subroutine for printing a time series on standard output ***
*** unit with 76 print positions per line ***

... INPUT BY PARAMETERLIST

ILOPTCD I Option code
-1 ..... only header output requested
ELSE ... also data is to be printed
INDPRT I Indicator for print-out size
<1 ..... no header output
>4 ..... print full item information
DUMMY L Indicates if item is a dummy
ITMDEF I(3) Time definition for which the series
will be printed
KIDNP T I(4) Integer-5-coded item identifier
DATSTR R(*) Vector containing the desired (aggregated)
data values to be printed
HEADER C*76 Header description

... INPUT BY PARAMETER STATEMENT COMPOUND/pardata/

LNSTR I Dimension of NUMSTR

... INPUT BY PARAMETER STATEMENT COMPOUND/parxdb/

LXXDB I Length of an integer-4-coded item identifier
of an external data base

... INPUT BY COMMON/miss/

MISSB I Number of missing values at the beginning
MISSE I Number of missing values at the end of the series

... INPUT BY COMMON/dbninf/

NUMSTR I(LNSTR) Numerical string containing the full
information and the unchanged data of an item
(for further documentation see subroutine SERWRT)

... OUTPUT BY PARAMETERLIST

IERRCD I Error code

... RETURN Normal exit
... RETURN1 Error exit
SUBROUTINE sp076(*,ierrod,iopcoi,indprt,dummy,itmdef,kidntf,
   ,datstr,header)

CHARACTER*76 header
CHARACTER*21 qidgedt
CHARACTER*14 qdtdec
CHARACTER*10 tttx,tttxt
CHARACTER*7 qtmdec
CHARACTER*5 agggmod(0:3)
LOGICAL dummy
DIMENSION itmdef(3),datstr(*)

INCLUDE ias-util.dolcommsg,LIST
INCLUDE ias-util.comcommsg,LIST
INCLUDE ias-util.dolcsign,LIST
INCLUDE ias-util.comcsign,LIST
INCLUDE ias-util.pardata,LIST
INCLUDE ias-util.parxdb,LIST
INCLUDE ias-util.comdbninf,LIST
INCLUDE ias-util.commiss,LIST
INCLUDE ias-util.dolorigin,LIST
INCLUDE ias-util.comorig,LIST

DATA agggmod,’sum’,’aver.’,’stock’,’defl.’/

ierrod = 0

IF(indprt.gt.0) THEN

   CALL tstedt(*980,ier0cd,’DS’,tttx,tttxt,indst)
   msgin = tttx//qidgedt(kidntf,’4’)//qdtdec(numstr(8))
   //msgstp
   CALL yupon(*980,ier0cd,msgin(1:1))
   CALL msgprt(*990,7510)
   msgin = header//msgstp
   CALL msgprt(*990,0)

ENDIF

IF(indprt.gt.4) THEN

   CALL msgprt(*990,10)

C ... L-option is specified

   CALL msgprt(*990,10)

C ... print time range of values in series and date/time last
C ... header update

   IF(dummy) THEN
      msgin = ’dummy item’
   ELSE
      msgin = qtmdec(numstr(12),numstr(13))//qtmdec(numstr(12))
* ENDIF
  msgin(15:30) = qdtdec(numstr(numstr(2)+6))/msgstp
  CALL msgprtr(*990,7550)
C C ... print time range of prognosted values in series and date/time
C C ... of storage
C
  IF(numstr(15).eq.0.and.numstr(16).eq.0) THEN
    msgin(1:14) = ' '
  ELSE
    msgin = qtmdec(numstr(12),numstr(15))/qtmdec(numstr(12)
    ,numstr(16))
  ENDIF
  msgin(15:29) = qdtdec(numstr(7))/msgstp
  CALL msgprtr(*990,7540)
C C ... print aggregation mode and generation type
C
  IF(numstr(11).ge.0.and.numstr(11).lt.15) THEN
    ind1 = numstr(11)
  ELSE IF(numstr(11).ge.90.and.numstr(11).lt.100) THEN
    ieven = numstr(11)/2
    IF(ieven*2.eq.numstr(11)) THEN
      ind1 = 16
    ELSE
      ind1 = 15
    ENDIF
  ENDIF
  ELSE
    CALL syserr(*990,'SP076')
  ENDIF
  WRITE(msgin,FMT='(I1,A5,A21,A1)')
  numstr(20),aggmod(numstr(numstr(20))),origin(ind1),msgstp
  CALL msgprtr(*990,7550)
C C ... print nominal and real time series if aggregation mode is deflator
C
  IF(numstr(20).eq.3) THEN
    msgin = qidedt(numstr(21),4)/qidedt(numstr(25),4)/msgstp
    ind0 = 29
    CALL msgprtr(*990,7560)
  ELSE
    ind0 = 21
  ENDIF
C C ... print in case of generation type 'interface' the name of the
C C ... data item in the external data base
C
  IF(numstr(11).ge.90) THEN
    ind1 = lxxdb#4
    msgin = ' '
    CALL st4dec(*980,ier0cd,numstr(ind0),lxxdb,msgin(1:ind1))
    msgin(65:65) = msgstp
    CALL msgprtr(*990,7570)
    ind0 = ind0+lxxdb
ENDIF
C ... print calc-string
C
IF(numstr(ind0).gt.1) THEN
  msgin = '  
  msgin(65:65) = msgstp
CALL msgprt(*990,7590)
CALL msgprt(*990,1000)
ENDIF
ENDIF

IF(ioptcd.ne.-1) THEN
  CALL msgprt(*990,10)
  IF(dummy) THEN
    msgin = qidedt(kidntf,4)//msgstp
    CALL msgprt(*990,7650)
  ELSE
C ... print of data
C
  msgnum = 7610
  CALL spdata(*980,ier0cd,msgnum,itmdef,datstr)
ENDIF
ENDIF
C
RETURN
C ... error section
C
980  ierccd = ier0cd
990  RETURN
END
THE SEMIPORTABLE ROUTINES WITHIN THE IAS-SYSTEM

YBLKDT
YCLOSE
YDATIM
YFILE
YINQU
YIOERR
YOPEN
YQMACD
YSET
ZCSFPT
BLOCK DATA yblkdt

INCLUDE ias-util.docosign,LIST
CHARACTER*2 iaslev
CHARACTER*1 star,blank,comma,apost,sfsep,mast, pound,lbrak
     ,rbrak,msgvar,msgstp,msgcon,lsg1,lsg2,lsg3,slash,pointc
     ,sysign,msschr

INCLUDE ias-util.comcsign,LIST

... COMMON CSIGN contains special characters

STAR...............sign indicating an IAS-command
BLANK..............separator between option(command) and data string
COMMA..............separator between command and option and
                  between two fields in input string
APOST..............sign to be used for marking a comment
SFSEP...............separator between two subfields
MAST...............sign starting an system command
POUND...............sign starting a data input
LBRAK...............left pointed braket
RBRAK...............right pointed braket
MSGVAR...............sign to be replaced by message
MSGSTP...............sign indicating the end of a message
MSGCON...............sign indicating continuation of a message
LSG1...LSG3........signs additionally allowed to be used in a name
SLASH...............sign used as division symbol
POINTC...............sign to mark a special position in the line above
SYSSGN...............sign starting a SYSTEM-command
IASLEV...............release level of IAS-SYSTEM (C*2)
CHRL...............latest character read in calcstring
MSSCHR...............character to indicate a missing value

COMMON /csign/ star,blank,comma,apost,sfsep,mast,pound,lbrak
     ,rbrak,msgvar,msgstp,msgcon,lsg1,lsg2,lsg3,slash,pointc
     ,sysign,msschr
DATA pound,lbrak,rbrak /'#','(',')'/
DATA msschr,iaslev
DATA lsg1,lsg2,lsg3,slash,pointc,sysign/'$','#','&','/','\','^','$'</DATA iaslev,chrl,msschr/'!',' ','.'/'

INCLUDE ias-util.comiordc,LIST

... COMMON IORDC containing computer depending values
IAC.............ordinal number of upper code A
IIC.............ordinal number of upper code I
IJC.............ordinal number of upper code J
IRC.............ordinal number of upper code R
ISC.............ordinal number of upper code S
IZC.............ordinal number of upper code Z
I9C.............integer number representing character 9
I0C.............integer number representing character 0
IDIFC...........difference of lower code a to upper code A
IBLC............ordinal number of character blank

COMMON /iordc/ iac,iic,ijc,irc,isc,izc,i9c,i0c,idifc,iblc
DATA iac/65/,iic/73/,ijc/74/,irc/82/,isc/83/,izc/90/
DATA i0c/48/,i9c/57/,idifc/-32/,iblc/32/

INCLUDE ias-util.dolocommand,LIST
CHARACTER*80 striin,spcstr
CHARACTER*6 option
CHARACTER*1 comand

INCLUDE ias-util.comcommand,LIST

... COMMON COMAND includes the whole input string as well as
seperated command/option and the packed specification string

STRIIN...........whole input string
COMAND...........non-encoded command
OPTION...........non-encoded option
SPCSTR...........packed specification string

COMMON /comand/ striin,comand,option,spcstr

INCLUDE ias-util.dolcmdtab,LIST
CHARACTER*8 cmdtbl

INCLUDE ias-util.comcmdtab,LIST

... COMMON CMDTAB includes all available IAS-commands

CMDTBL.............table of commands - length in parameter ICTL

PARAMETER (ictl=100)
COMMON /cmdtab/ cmdtbl(ictl)
DATA cmdtbl/' ', ' ', ' ', 'S', 'SER', 'U', 'UPD', 'E', 'EQU', 'RPT', 'RPT',
* 'TAB', 'TAB', 'C', 'CALC', 'M', 'MOD', 'T', 'TIME', 'F', 'FILE',
* 'P', 'PAT', 'W', 'WAIT', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
* ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
* ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
* 'OLS', 'O', 'IV', 'TSL', 'KLS', 'LIML', '2SLS', '3SLS', 'LIVE', 'FIVE',
* 'FIML', 'TEST', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
* 'DEL', 'EDIT', 'COPY', 'INFO', 'LINK', 'PLT', 'PCH', 'TAPE', 'MAT', 'LP',
* 'DB', 'EXIT', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
*
INCLUDE ias-util.comstrcd, LIST

... COMMON STRCD includes encoded command/option and
... vectors of pointers to fields/subfields in string SPCSTR

ICMDNR............encoded command
IOPTNR(2).........encoded option containing character and
... numerical options
IFLDVC(10)........vector pointing to the last character in a field
ISFDVC(40)........vector pointing to the last character in a subfield

COMMON /strcd/ icmdnr,ioptrn(2),ifldvc(10),isfdvc(40)

INCLUDE ias-util.comstatus, LIST

ISTAT(1)..........read status
ISTAT(2)..........iruntp
ISTAT(3)..........numcmd
ISTAT(4)..........numinp
ISTAT(5)..........isups
ISTAT(6)..........numio
ISTAT(10).........last input line not yet checked if .gt.0
ISTAT(11).........number of files in file control table FCT
ISTAT(14).........continuation record was read at last READ from
... MSG-file if ISTAT(15) > 0
ISTAT(17).........continuation of algebraic string encountered
ISTAT(18).........record number of EXWS-File record in buffer
ISTAT(19).........number of changes occurred to the record of
... EXWS-File actually in buffer

ISWTCH(1).........input log on (=1) or off (=0)
ISWTCH(2).........echo mode on (=1) or off (=0)
ISWTCH(3).........walk in case of system errors. ON/OFF (see above)
ISWTCH(4).........debug mode ON/OFF
ISWTCH(5).........output also to alternate print file. 0,1,2
... 0 = no output log
... 1 = all output to output log
... 2 = as above, additionally all input (like echo
... mode) to output log
ISWTCH(6).........print error message if output is truncated.ON/OFF
ISWTCH(7).........expert mode. ON/OFF
ISWTCH(8).........quantity of print output. 0,1,2,3,4,5,6
ISWTCH(9).........batch mode. ON/OFF
ISWTCH(10)........assign total set of files if ISWTCH(10).EQ.1
... assign only minimal number of files if
... ISWTCH(10).EQ.0
ISWTCH(11)........switch for writing results to the mass storage
file EXWS (extended working storage) for re-use.
if 'ON' all records are written to file EXWS,
if 'OFF' records <= ISIZE(5) are not written.

ISWITCH(13)......DB-assign. 0,1,2,3
  0 = No data base assigned
  1 = Data base assigned read-only
  2 = Data base assigned write-only
  3 = Data base assigned read/write enabled

ISWITCH(14)......EXWS assigned. ON/OFF
  0 = File EXWS not assigned
  1 = File EXWS assigned

ISWITCH(15)......internal OS-file assigned. ON/OFF
ISWITCH(16)......message file assigned. ON/OFF
ISWITCH(17)......input-log assigned. ON/OFF
ISWITCH(18)......output-log assigned. ON/OFF

ISIZE(4)........record length of file EXWS (extended working storage on mass storage)
ISIZE(5)........number of records in file EXWS
ISIZE(6)........number of records in file EXWS which are sensitive to ISWITCH(11)
ISIZE(11)......pointer indicating type of computer
ISIZE(12)......number of records in MSG-file
ISIZE(13)......maximum number of bits which can be used for a positive integer, i.e. 2**ISIZE(13)-1 is the largest integer for this implementation
ISIZE(14)......number of lines per page
ISIZE(15)......number of characters per print line for interactive mode, not counting the vertical spacing character
ISIZE(16)......number of characters per print line for batch mode, not counting the vertical spacing character
ISIZE(17)......number of bits per computer word (see also ISIZE(13))
ISIZE(18)......character length in bits
ISIZE(19)......number of significant bits per character
ISIZE(20)......number of characters per word

IDBST(1).......record length of file 11
IDBST(2).......number of records of file 11
IDBST(3).......pointer to root record of tree in file 11
IDBST(4).......pointer to first leaf in B*-tree
IDBST(6).......max. number of keys in root
IDBST(7).......max. number of keys in leaf
IDBST(8).......middle key in leaf
IDBST(9).......position of middle key in leaf
IDBST(11)......factor for computing size of file 12 and 13
IDBST(12)......record length of file 12
IDBST(13)......number of records of file 12
IDBST(16)......record length of file 13
IDBST(17)......number record in file 13
IDBST(18)......length of integer info. of one record
IDBST(19)........ length of character info. of one record
    (in character storage units)
IDBST(21)........ pointer to next free record of tree in file 11
IDBST(22)........ number of keys stored in file 11 (B*-tree)
IDBST(23)........ number of records used in file 11 for B*-tree
IDBST(26)........ number of catalogued IAS-files in file 11
IDBST(31)........ pointer to next free record of file 12
IDBST(32)........ number of elements stored in file 12
IDBST(33)........ number of records used in file 12
IDBST(36)........ pointer to next free record of file 13
IDBST(37)........ number of elements stored in file 13
IDBST(38)........ number of records used in file 13

COMMON /status/ istat(20),iswtch(20),isize(20),idbst(40)
    ,input,iout,kexws,kosint,msgni,loginp,logout

INCLUDE ias-util.parrelc,LIST
PARAMETER (lrec11=510,lrec12=108,lrec13=20,lrec13=120,lrec14=500)

INCLUDE ias-util.comdbrecs,LIST

... contains data base records of file 12 and file 13

IROOT(1)........ number of keys in data base root
    (2)........ free record concatenation pointer
    (3)........ not used
    (4)........ not used
    (5)......... /
    (6)......... /
    (7)......... / for further statistical use
    (8)......... /
    (9)......... /
    (10)........ 1st pointer (less than 1st key)
    (11)......... /
    (12)......... /
    (13)......... / 1st key
    (14)......... /
    (15)........ 2nd pointer (greater than 1st key)
    (16)......... /
    (17)......... /
    (18)......... / 2nd key
    (19)......... /
    (20)........ 3rd pointer (greater than 2nd key)
    ...
    ...

ILEAF(1)........ number of keys in data base leaf
    (2)........ free record concatenation pointer
    (3)........ pointer to left leaf
    (4)........ pointer to right leaf
    (5)......... /
    (6)......... /
(7)....../ for further statistical use
(8)....../
(9)....../
(10)......1st pointer (less than 1st key)
(11)....../
(12)....../
(13)....../1st key
(14)....../
(15)......2nd pointer (greater than 1st key)
(16)....../
(17)....../
(18)....../2nd key
(19)....../
(20)......3rd pointer (greater than 2nd key)

COMMON /d brecs/ iroot(lrec11),ilsaf(lrec11)

INCLUDE ias-util.comintott,LIST

INDUCT......use control table, contains pointer to file in FCT
(1)......base file
(2)......data file
(3)......equation file
(4)......model file
(5)......solution file
(6)......text file

KFCT.........keyed names of assigned IAS-files

IRWPRM.......read/write permission indicator, corresponding
to files in KFCT
  =-1...array element not in use
  = 0...file assigned read enabled, write protected
  = 1...file assigned write enabled, read protected
  = 2...file assigned read/write enabled

COMMON /intott/ kfct(2,0:10),irwprm(0:10),induct(10),itot(3,40)
DATA induct,kfct,irwprm /10*1,22*0,11*-1/

INCLUDE ias-util.comsys,LIST

... COMMON SYS contains tolerance values for calculation

tol(1) ... EXP(tol(1)) is greatest possible value for single
  precision
tol(2) ... allowed declination for a value to be treated as
  integer
tol(3) ... greatest possible value for single precision
COMMON /sys/ tol(10)
DATA tol /80.0, .1E-20, 1.0E32, 1.0E5, 6*0.0/

INCLUDE ias-util.parcalc,LIST
PARAMETER (maxcal=12, ninvar=7, ninfntn=8, maxsc1=400, ipoll=800,
* maxlbl=800, maxfl=10, maxrft=30, maxval=2000, maxscv=2400,
* maxpar=20, maxkey=4, lloc=500)

INCLUDE ias-util.dolcalc1,LIST
CHARACTER*8 intvtb, infnct
CHARACTER*5 keytbl
CHARACTER*(lloc) clochr
CHARACTER*1 chr

INCLUDE ias-util.comcalc1,LIST
COMMON /calc1/ intvtb(ninvar), infnct(ninfntn), keytbl(maxkey),
* clochr, chr
DATA intvtb/'CALC', 'LC', 'C1', 'C2', 'C3', 'C4', 'C5'/
DATA keytbl/'IF', 'THEN', 'ELSE', 'END'/
DATA infnct/'SIN', 'COS', 'LOG', 'LN', 'LD', 'EXP', 'MIN', 'MAX'/

INCLUDE ias-util.comfnpar,LIST

... COMMON FNPAR contains the number of parameters for
intrinsic functions

COMMON /fnpar/ ninpar(ninfntn)
DATA ninpar/1,1,1,1,1,1,1,2,2/

INCLUDE ias-util.doltpstp,LIST
PARAMETER (ntyp=5, nsytyp=1, nstyp=20)
CHARACTER*1 tochr(ntyp), stchr(nstyp)
CHARACTER*10 tttxt(ntyp), sttxt(nstyp)

INCLUDE ias-util.comtpstp,LIST

... COMMON TPSTP allows the conversion of the type-subtype-code
... to text

TCHR ..... type-characters (4 type-characters +
1 system type-character)
STCHR ..... subtype-characters
TTXT ..... text for the single types (e.g. 'DATA' for 'D')
STTXT ..... text for the single subtypes (e.g. 'SERIES' for 'S')
IPTTST ..... pointer to the first subtype-character
of a specific type (TCHR) within STCHR
COMMON /ipstsp/ ipttst(ntyp)
COMMON /ctptsp/ tchr,ttxt,stkhr,stkxt
DATA ipttst /1,4,8,15,19/
DATA tchr /'X', 'M', 'D', 'F'/
DATA stkhr /34, 7*!, V, S, M, /
DATA ttxt /'text', 'model', 'equation', 'data', $sys'/
DATA stkxt /34, 7*!, /*
* 'value', 'series', 'matrix', 'function', *

INCLUDE ias-util.dcolorin,LIST
CHARACTER*21 origin(0:16)

INCLUDE ias-util.comorigin,LIST

... COMMON ORIGIN contains the text for the different origin types

COMMON /origin/ origin
DATA origin /'terminal input', 'formatted data deck'
* 'calculated by *SER', 'calculated by *CALC', 'copy'
* 'seasonal adj. series', 'seasonal fact. series'
* 'param.variable vector', 'estim.endogenous var.'
* 'estimated residuals', 'interface to ext. db', 'interf.ext.db/transf.'/

INCLUDE ias-util.comexwsbf,LIST

... Table of segments in EXWS

COMMON /exwsbf/ ibuff(lrec14),iexwst(100)
DATA iexwst /1,2,3,4,5,6,7,8,9,10,40*0,31,52,53,15*0,69,70,
* 75,29*0/

INCLUDE ias-util.dolorhsys,LIST
CHARACTER*20 osflnm(13:18)

INCLUDE ias-util.comchrsys,LIST

... COMMON CHRSYS containing IAS-SYSTEM character information

OSFLNM..... Table of OS file names
(13) ... data base name
(14) ... external working storage file
(15) ... internal OS file for (user) communication
(16) ... message file
(17) ... input log
(18) ... output log

COMMON /chrsys/ osflnm
DATA osflnm /"EXWSFL",'INTCOM','IAS-30*IAS-MSG'
*,'INPLOG','OUTLOG'/

END
C ... *** Program for closing and de-assigning operating system ***
C ... *** files ***
C ...
C ... COMMENT
C
This is an unportable routine of the first kind because of the configuration dependent handling of the facility return codes.

The reader should refer to ANSI X3.9-1978 FORTRAN and to UP-8244.1 SPERRY UNIVAC series 1100 FORTRAN(ASCII) or to appropriate documentation of other vendors if this program is to be implemented on other machine ranges.

All character variables must be uppercase on input!!!

C ... INPUT BY PARAMETERLIST
C
IUNIT I unit number to be closed
FILE C*20 < file name to be closed (may be space)
FILSTA C*7 < close-status: 'KEEP' or 'DELETE'
IFREE I free (deassign) file if IFREE .le. 0,
but keep cataloged if STATUS = 'KEEP'
INDPRT I print error messages if INDPRT .ge. 1
print warnings if INDPRT .ge. 2

C ... OUTPUT BY PARAMETERLIST
C
IERRCD I error code

C ... RETURN normal exit
C ... RETURN 1 error exit
C ...

SUBROUTINE yclose(*,ierrcd,iunit,file,filsta,ifree,indprt)
C
CHARACTER*(*) file,filsta
CHARACTER*20 lofile
CHARACTER*20 yqmaed,ierrcd
CHARACTER*7 lostat
CHARACTER*6 rdkey,wrkey
INCLUDE ias-util.dolcommsg,LIST
INCLUDE ias-util.comcommsg,LIST
INCLUDE ias-util.dolcsign,LIST
INCLUDE ias-util.comcsign,LIST
C
ierrcd = 0
lofile = * file
lostat = filsta
C ... compute numerical index for STATUS-specifier
C
IF (filsta.EQ. 'KEEP') THEN
  ifilst = -1
ELSE IF (filsta.EQ. 'DELETE') THEN
  ifilst = -2
ELSE
  GOTO 920
ENDIF
C
C ... FORTRAN-close of specified unit
C
CLOSE(UNIT=iunit, IOSTAT=ier0cd, STATUS=lostat)
C
IF(ier0cd.ne.0) THEN
  SEM
C
C ... closing error
C
  ierrod = -4220
  SEM
  IF(indprt.ge.2) THEN
    cerrod = yqmacd(ier0cd)
    msgin(1:20) = cerrod
    WRITE(msgin(21:22),FMT='(i12)') iunit
    msgin(23:23) = msgstp
    CALL msgprt(*990,4220)
  END IF
  END IF
C
C ... prepare parameters for de-assignment of file
C
IF(lofile.eq. ' ') WRITE(lofile(1:2),FMT='(i12)') iunit
  rdkey = ' '
  wrkey = ' '
  itrks = 0
  indrwr = 0
  indexc = 0
  iwait = 0
C
C ... call YFILE for de-assignment
C
CALL yfile(*990,ier0cd,lofile,rdkey,wrkey,ifilst,itrks,indexc
  '*   ,iwait,ifree,indprt)
C
RETURN
C
C ... error messages
C
920  WRITE(*,*), 'Invalid STATUS specifier', filsta
  ier0cd=101
  CALL syserr(*990, 'YCLOSE')
C
990  RETURN1
END
Program for providing date and time

(1) in CHARACTER FORMAT YYYYMMDDHHMMSS (see below)

(2) in BIT-FORMAT YYYYMMMDDDDDHHHHMMMMSSSSSS (see below)

This is an unportable routine of first order

No input from parameterlist

DATTIM C*14 > date and time in CHARACTER FORMAT YYYYMMDDHHMMSS

IDATIM I date and time in BIT FORMAT

32 26 22 17 12 7 1
YYYYYMMMDDDDDHHHHMMMMSSSSSS

IDATIM = (Y-1980) * 2**26
+ M * 2**22
+ D * 2**17
+ H * 2**12
+ M * 2**6
+ S

unique exit
no error exit

SUBROUTINE ydatim (dattim,idatim)

CHARACTER*14 dattim
CHARACTER*8 date,time

CALL zzdate(date,time)

Calculate INTEGER-value of date & time

UNIVAC 1100 presents date in format MMDDYY

READ(date,2)m2,m1,m3

FORMAT(3I2)
m3=m3-80
isum=m3*2**26 + m2*2**22 + m1*2**17

C ... UNIVAC 1100 presents time in format HHMMSS

READ(time,2)m3,m2,m1
isum=m3*2**12 + m2*2**6 + m1 + isum
idatim=isum

C ... Calculate CHARACTER-variable with date & time

datim='19'//date(5:6)//date(1:4)//time(1:6)

C
RETURN

C
END
*** Program for cataloging, assigning, freeing and deleting ***

*** of operating system files ***

COMMENT

That is an unportable routine of first kind

The reader should refer to ANSI X3.9-1978 FORTRAN and to
UP-8244.1 SPERRY UNIVAC Series 1100 FORTRAN (ASCII) or the
appropriate documentation of other vendors if this program
is to be implemented on other machine ranges.

All character variables must (currently) be uppercase on input !!

This routine contains explicit WRITE-statements rather then
calling the messages from the message file. This cannot be
avoided, however, as the message file might not be available
before having been assigned by this program.

INPUT BY PARAMETERLIST

FILE C*20 operating system file name
RDKEY C*6 read key for specified file
WRKEY C*6 write key for specified file
IFILST I status for OPEN- or CLOSE-statement
  IFILST = 4  <=>  STATUS = 'NEW'  (OPEN)
  IFILST = 3  <=>  STATUS = 'EXTEND'  (OPEN)
  IFILST = 2  <=>  STATUS = 'OLD'  (OPEN)
  IFILST = 1  <=>  STATUS = 'UNKNOWN'  (OPEN)
  IFILST =-1  <=>  STATUS = 'KEEP'  (CLOSE)
  IFILST =-2  <=>  STATUS = 'DELETE'  (CLOSE)
ITRKS I number of tracks to be requested
INDEXC I indicator if file is to be assigned exclusively
  (INDEXC=1) or not (INDEXC=0)
IWAIT I indicator if program should be kept in a wait
  position if the file is temporarily not
  available (wait if IWAIT=1, don't wait if
  IWAIT=0)
IFREE I Free file before assigning it
INDPRT I Print error messages if INDPRT .ge. 1
  Print warnings if INDPRT .ge. 2
  Print program trace if INDPRT .ge. 3

OUTPUT BY PARAMETERLIST

IERRCD I error code (facility key code)

RETURN normal exit
RETURN 1 error exit
SUBROUTINE yfile (*,ierroc,file,rdkey,wrkey,ifilst,itrks,indexe
   * ,iwait,ifree,indpnt)

CHARACTER*52 asgstr
CHARACTER*30 frestr
CHARACTER*20 file
CHARACTER*6 key(2),rdkey,wrkey

INCLUDE ias-util.comstatus,LIST

ierroc = 0
asgstr = ' ' 
frestr = ' '

... Execute only if UNIVAC

IF(isize(11).NE.1)GOTO 900

key(1) = rdkey 
key(2) = wrkey

... evaluate exact file name (size)

IPOS   I    last character of PROJ-ID*FILE[..] in FILE
IPOS2  I    position of last character in ASGSTR

ipos = jlastc(file)
IF (ipos.le.0) GOTO 950

... generate general part of asgstr

IF (ifilst.GE.2) THEN
   asgstr(1:5) = '@ASG,' 
   ipos2 = 10+ipos 
   asgstr(11:ipos2) = file(1:ipos)
END IF

... transfer of keys

DO 400 i0 = 1,2
   ipos2 = ipos2+i0 
   asgstr(ipos2:ipos2) = '/' 
   iposc = jlastc(key(i0))

... non-blank key ?

IF (iposc.gt.0) THEN
   ipos1 = ipos2+i0 
   ipos2 = ipos2+iposc 
   asgstr(ipos1:ipos2) = key(i0)(1:iposc)
END IF

400 CONTINUE
C ... number of tracks specified?

C IF (itrks.gt.0) THEN
  ipos1 = ipos2+1
  ipos2 = ipos2+8
  asgstr(ipos1:ipos2) = ',//
  WRITE(asgstr(ipos1+4:ipos2),'(4)i') itrks
ENDIF

  ipos2 = ipos2+3
  asgstr(ipos2-2:ipos2) = '.

C ... file to be catalogued?

C IF (ifilst.EQ.4) THEN
  asgstr(6:10) = 'UP'
  CALL zosfpt(#990,ier0cd,asgstr,indprt)
ENDIF

ENDIF

C ... free file if IFREE.gt.0

C IF (ifree.GT.0.OR.ifilst.EQ.4) THEN
  frestr = '@FREE ',//file(1:ipos)//'.
  IF (ifilst.EQ.-2) frestr(6:7) = ',D'
  CALL zosfpt(#990,ier0cd,frestr,indprt)
END IF

C ... assignment of file, if status = 'NEW' or 'OLD'

C IF (ifilst.GE.2) THEN
  asgstr(6:10) = 'A'
  IF (indexc.eq.1) THEN
    asgstr(7:7) = 'X'
    ipose = 8
  ELSE
    ipose = 7
  ENDIF
  IF (iwait.ne.1) asgstr(ipose:ipose) = 'Z'
  CALL zosfpt(#990,ier0cd,asgstr,indprt)
ENDIF

C 900 ierre0d = ier0cd
RETURN

C ... Error messages

C 950 WRITE(*,*),'File specification missing'
  ier0cd=1
990 ierre0d=ier0cd
RETURN 1

C END
*** Program for inquiring about files (calling FORTRAN INQUIRE).

** COMMENT **

In order to separate all file handling into special modules with a tendency to non-portability the INQUIRE-statement is located only in this program unit and in no other program unit throughout the whole program.

Compare ANSI X3.9-1978, chapter 12.2.1, for the definition of file existence. File existence does not necessarily imply that the file is contained in some file directory. On the other hand a file may be known to the processor but does not exist for the program.

** INPUT BY PARAMETERLIST **

FILE C(*#) name of the file being inquired about

** OUTPUT BY PARAMETERLIST **

IERRCD I error code (currently not used)
IEXIST I existence indicator
<0 : file does not exist
>0 : file exists

IDUMMY I currently not used

** RETURN **
unique exit

SUBROUTINE yinqu (ierroc, file, iexist, idummy)

CHARACTER*(#) file
LOGICAL exist

INQUIRE(IOSTAT=ierroc, FILE=file, EXIST=iexist)

IF (exist) THEN
iexist=1
ELSE
iexist=-1
END IF

idummy=0

END
... *** Program for editing and printing I/O-errors
... ***

... COMMENT

This is an unportable routine of first order

... INPUT BY PARAMETERLIST

IREC  I  Record number (-99999 if not applicable)
INDPRT I  Print indicator.
Print an error message if the machine
independent error code is .LE. INDPRT*10.
Nothing is printed if INDPRT.LE.0,
everything is printed if INDPRT.GE.9

... OUTPUT BY PARAMETERLIST

No output by parameterlist

... TRANSPUT BY PARAMETERLIST

IOERCD I  I/O error code
on input: machine dependent
on output: machine independent
1: other error
21: attempted to read from unassigned mass
storage area and the record read was not
the first ons. This means mostly that the
file had no ENDFILE record.
31: attempted to read from unassigned mass
storage area and the record read was
the first ons. This means mostly that a
READ was performed on an empty file.
41: attempted to read an empty record from
a direct access file

... RETURN  unique exit

SUBROUTINE yioerr (ioerod,irec,indprt)

CHARACTER*8 areo

INCLUDE ias-util.comstatus,LIST

IF (ioerod.eq.0) RETURN
IF (iswteh(4).GT.0) THEN
  locprte=9
ELSE
    locprt=indprt
END IF

C
ios=ioercd/2**27
irest=ioercd-ios*2**27
iou=irest/2**18
ios=irest-IOUS*2**18

SEM

C
IF (ioc.EQ.1.AND.ios.EQ.5) THEN
  IF (irec.EQ.1) THEN
    ioercd=31
  ELSE
    ioercd=21
  END IF

SEM

C
ELSE IF (ioc.EQ.1053) THEN
  ioercd=41
ELSE
  ioercd=1
END IF

C
IF (ioercd.LE.10*locprt) THEN
  IF (irec.eq.-99999) THEN
    arec='--------'
  ELSE
    WRITE(arec,'(I8)')irec
  END IF

C
WRITE (*,4) iou,arec,ioc,ios
4 FORMAT(‘input/output error in unit’,i4,8x,’Record # ’,A8/
  ’ Status (decimal)’,i8,6x,’Substatus (decimal)’,i8)

C
END IF

C
RETURN
END
Program for assigning and opening files and i/o-units.
This Program calls file-handling procedure IFFILE for assignment of an operating system file.

... COMMENT

This is an unportable routine of the first kind because of the configuration dependent computation of the number of needed tracks on mass-storage and the handling of the facility return codes.

The reader should refer to ANSI X3.9-1978 FORTRAN and to UP-8244.1 SPERRY UNIVAC series 1100 FORTRAN(ASCII) or to appropriate documentation of other vendors if this program is to be implemented on other machine ranges.

All character variables must be uppercase on input!!!

This routine contains explicit WRITE-statements rather than calling the messages from the message file. This cannot be avoided, however, as the message file is not available before being opened by this program.

... INPUT BY PARAMETERLIST

IUNIT I file reference number
FILE C*20 < file name to be associated with the spec. unit
RDKEY C*6 < read key for specified file
WRKEY C*6 < write key for specified file
FILSTA C*7 < status for OPEN-statement.
Status may be 'OLD', 'NEW' or 'UNKNOWN'. 'SCRATCH' is not implemented.
In accordance with the proposals for FORTRAN 8x status 'EXTEND' has been added.

For a D/A file NRCD5 may be increased with status 'EXTEND'. A sequential file will be positioned after the current last record

ACCESS C*10 < (before the ENDFILE-record, if one exists)
FORM C*11 < access method: 'SEQUENTIAL' or 'DIRECT'
IRECL I record length for DIRECT ACCESS FILES
NRCD5 I maximum number of records in a D/A file
(RCD5 = nrods is used in the OPEN-statement but this clause is not standard conforming)

INDWRT I indicator if writing is prohibited (INDWRT=0) or allowed (INDWRT=1)

INDEXC I indicator if file is to be assigned exclusively (INDEXC=1) or not (INDEXC=0)

IWAIT I indicator if program should be kept in a wait position if the file is temporarily not
available (wait if IWAIT=1, don't wait if IWAIT=0)

Free file before assigning it if IFREE .gt. 0.
This parameter is not used if filsta = `NEW'.
A file with status `NEW' is always freed
--- after --- having been cataloged and then
considered to have status `OLD'

Indicator if (processor dependent) mass storage
size is to be computed (LMSIZE.gt.0) or not

Print error messages if INDPRT .ge. 1
Print warnings if INDPRT .ge. 2
Print program trace if INDPRT .ge. 3
or if ISWITCH(4) .GT. 0

... OUTPUT BY PARAMETERLIST

IERRCD I error code
   attention to
   -1 ... read key missing
   -2 ... write key missing

... RETURN normal exit
... RETURN 1 error exit
...

SUBROUTINE yopen(*,ierrcd,iunit,file,rdkey,wrkey,filsta,access
*   ,form,ierecl,nrods,indwrt,indexc,iwait,ifree,lmsize,indprt)

CHARACTER*(*) file,rdkey,wrkey,filsta,access,form
CHARACTER*7 lostat
CHARACTER*6 key(2)

INCLUDE ias-util.comstatus,LIST

... transferring dummy variables to local variables

loocrt=MAX(indprt,3*iswton(4))
ierrcd = 0
loostat = filsta
key(1) = rdkey
IF(indwrt.eq.0) THEN
   key(2) = '
ELSE
   key(2) = wrkey
ENDIF

... compute numerical index for STATUS-specifier

IF (loostat.EQ. `NEW') THEN
   ifilst = 4
ELSE IF (loostat.EQ. `EXTEND') THEN
   ifilst = 3
ELSE IF (loostat.EQ. `OLD') THEN
ifilst = 2
ELSE IF (lostat.EQ. 'UNKNOWN') THEN
   ifilst = 1
ELSE
   GOTO 920
ENDIF

/* compute machine dependent size (number of tracks) to be requested */

IF (lmsize.gt.0) THEN
   IF (indwrt.gt.0) THEN
      IF (form.eq. 'FORMATTED') THEN
         iwrddv = isize(20)
      ELSE IF (form.eq. 'UNFORMATTED') THEN
         iwrddv = 1
      ELSE
         WRITE(*,*) 'Error: Invalid FORM specifier ', form
         CALL syserr(*990, 'YOPEN')
      ENDIF
      iwords = nrcds*(irecl/iwrddv+4)+256
      itrks = iwords/1792+5
   ENDIF
   ELSE
      WRITE(*,*) 'Warning: File not write-enabled, mass ',
      ' storage size not calculated'
      itrks = 0
   END IF
ELSE
   itrks = 0
ENDIF

/* generate specification string for assignment and assign */
/* operating system file */
CALL yfile(*990, ier0cd, file, key(1), key(2), ifilst, itrks
     , indexo, iwwait, ifree, loooprt)

IF (isize(11).EQ.1 .AND. ifilst.EQ.4) lostat='OLD'

/* FORTRAN-opening for assigned file */
/* N.B.: Status = 'EXTEND' is allowed but not standard conforming */
IF (access.eq. 'SEQUENTIAL') THEN
   OPEN(UNIT=1unit, IOSTAT=ier0cd, FILE=file, STATUS=lostat
     , ACCESS=access, FORM=form)
ELSE IF (access.eq. 'DIRECT') THEN
   OPEN(UNIT=1unit, IOSTAT=ier0cd, FILE=file, STATUS=lostat
     , ACCESS=access, FORM=form, RECL=irecl, RCDS=nrcds)
ELSE
   WRITE(*,*) 'Invalid ACCESS specifier ', access
   CALL syserr(*990, 'YOPEN')
ENDIF

IF (ier0cd.eq.0) THEN
ierrocd=0

ELSE

... opening error

WRITE(*,*)'Error when opening unit ',iunit
CALL yioerr (ier0cd,-99999,9)
GOTO 990

ENDIF

RETURN

... other error messages

WRITE(*,*)'Invalid STATUS specifier ',lostat
ier0cd=101
CALL syserr(*990,'YOPEN')

ierrocd=ier0cd
RETURN1
END
C ... *** FUNCTION subprogram for translating an integer variable to a character variable (machine representation of the binary code) with length of 20 char's

C ... *** COMMENT

This is a FUNCTION subprogram.

This is an unportable routine of first kind.

However, if octal code is used to represent machine code, it is very likely that this program runs on the corresponding machine. On byte machines it has to be changed to hexadecimal code.

C ... INPUT BY PARAMETERLIST

INTEG I input variable

C ... OUTPUT BY FUNCTION VALUE

YQOCT C*20 > octal code output variable

C ... RETURN normal and error exit

C ...

CHARACTER*20 FUNCTION yqmacd (integ)

WRITE (yqmacd,FMT='(020)',IOSTAT=ierrod) integ SYN

IF (ierrod.ne.0) yqmacd='777777777777'

RETURN
END
C ... *** Subroutine to set maximal number of errors and other exceptions (see below) ***
C ... ***
C ...
C
C ... COMMENT
C
C This is a non-portable routine of first order. It sets the maximum number of various errors and exceptions which may occur during program execution. If this program is called repeatedly, the error counts are set to zero any time this program is called.
C
This program can only be called in connection with routines YADR, F2CON and it calls routine ZEXCEP which are all located in the same program file as routine YSET. It also calls routines UNDSET, OVFSET, DIVSET and CMLSET from the system library.
C
Therefore a MAP-element should contain the following lines:
C
IN main program
IN IHS*FTN-UTIL.YSET,YADR,F2CON,ZEXCEP
IN SYS*RLIB*.F2MATHFAULTS,F2MATHERR
EQU ZUNDST/UNDSET
EQU ZOVFST/OVFSET
EQU ZDIVST/DIVSET
EQU ZCMLST/CMLSET
C
C ... INPUT BY PARAMETERLIST
C
IUNDFL I max number of floating point underflows captured by the exception handling program.
If IUNDFL=0 or if more than IUNDFL underflows have occurred, underflows are no more captured.
Standard action in this case is that the variable with underflow is set to zero and no error message is printed.
C
IOVFL I max number of floating point overflows captured by the exception handling program.
If IOVFL=0, overflows are not captured, i.e. the program continues normally.
If IOVFL<0 or if more than IOVFL overflows have occurred, the program will terminate.
C
IDIV I max number of divide faults captured by the exception handling program.
If IDIV=0, divide faults are not captured, i.e. the program continues normally.
If IDIV<0 or if more than IDIV divide faults have occurred, the program will terminate.
C
IMATH I max number of mathematical faults captured by the exception handling program.
If IMATH<0 or if more than IMATH math-faults have occurred, the program will terminate.

IEXCEP  I
max number of erroneous exceptions captured by the exception handling program.
If IEXCEP<0 or if more than IEXCEP erroneous exceptions have occurred, the program will terminate.

... OUTPUT BY PARAMETERLIST
No output by parameterlist

... RETURN unique exit

SUBROUTINE yset (iundfl,iovfl,idi,v,imath,iexcep)
CALL zundst(iundfl) SEM
CALL zovfst(iovfl) SEM
CALL zdivst(idi) SEM
CALL zomlst(imath) SEM
CALL zexcep(iexcep) SEM

RETURN
END
Program to prepare the calling of routine ZCSF in order to perform an ACSF$ executive request (e.g. assign, catalog and free operating system files)

COMMENT

This is an unportable routine of second order

INPUT BY PARAMETERLIST

IMAGE C*80 < image of executive control statement.

This image must be terminated by the character sequence : blank period blank

EXAMPLE:

'@ASG,AXZ MIPROG*OLDFILE .'

INDPRT I

Print error messages if INDPRT .ge. 1
Print warnings if INDPRT .ge. 2
Print program trace if INDPRT .ge. 3

OUTPUT BY PARAMETERLIST

IERRCD I

Error code. The following special codes apply:

-1 : write key missing
-2 : read key missing
423 : facility error, including read and write key (both) missing
-424 : facility warning other than read or write key missing

RETURN normal exit
RETURN 1 error exit

SUBROUTINE zcsfpt(*,ierrcd,csfstr,indprt)

CHARACTER(*) csfstr
CHARACTER*20 errcd,yqmacd
CHARACTER*1 rdwrky

INCLUDE ias=util.comstatus,LIST
ipos=len(csfstr)

Call actual executive request ER ACSF$

CALL zzcsf (ier0cd,csfstr(1:ipos))

IF (ier0cd.lt.0) THEN
ierroc=423
ELSE IF (ier0cd.GT.0) THEN
  ierroc=424
ELSE
  ierroc=0
END IF

IF (indprt.lt.3.and.ier0cd.eq.0) RETURN

... Print error messages

cerrcd=yqmacd(ier0cd)

... Read/write keys missing?

crwrky=cerrcd(12:12)
IF (crwrky.eq.'3') THEN
cerrcd(9:9)=‘4’
  ierroc=423
else IF (crwrky.eq.'1') THEN
  ierroc=-1
else IF (crwrky.eq.'2') THEN
  ierroc=-2
END IF

... Printout of CSF-string?

IF (indprt.ge.3.or.
  * (indprt.ge.2.and.ierroc.lt.0)).or.
  * (indprt.ge.1.and.ierroc.gt.0)) THEN
  ipos1 = INDEX(csfstr(1:ipos),'/')
  IF (ipos1.GT.0) ipos=ipos1
  WRITE (iout,92) csfstr(1:ipos)
  FORMAT(1X,A)
END IF

IF (ierroc.gt.0.and.indprt.ge.1) WRITE (iout,94) cerrcd
  FORMAT ('Facility error ',A)
IF (ierroc.lt.0.and.indprt.ge.2) WRITE (iout,96) cerrcd
  FORMAT ('Facility warning ',A)

IF (ierroc.gt.0) RETURN
RETURN
END
THE CONVERSION ROUTINES WITHIN THE IAS-SYSTEM

ST5ENC
ST5DEC
ST4ENC
ST4DEC
*** Subprogram for converting a character string to a vector of numeric keys ***

... INPUT BY PARAMETERLIST

STRING C Character string
KEYDIM I Dimension of KSTR

... OUTPUT BY PARAMETERLIST

KSTR I(*) Numeric keystring

... Special Case

STRING C*20 Contains the file name, type, element name and version name starting at positions 1,9,11 and 19
KSTR Contains the numeric keys calculated from STRING

... RETURN Normal exit
... RETURN 1 Error exit

SUBROUTINE st5enc(*,ierrod,string,keydim,kstr)

CHARACTER*(*) string
CHARACTER*(1) chr
DIMENSION kstr(keydim)

INCLUDE ias-\util.dolcommsg,LIST
INCLUDE ias-\util.comcommsg,LIST
INCLUDE ias-\util.dolcomsign,LIST
INCLUDE ias-\util.comcsign,LIST

ier0cd=0
length=len(string)

dim=((length-1)/5)+1
IF(dim.LE.keydim)THEN
  mpos=0
  DO 220 i=1,dim
  ikey=0
  mpos=mpos+1
  IF(mpos.LE.length)THEN
    chr=string(mpos:mpos)
    ikey=ikey+jchr(chr)*(64**(5*i-mpos))
    IF(mod(mpos,5).GT.0)THEN
      GOTO 200
    ELSE
kstr(i)=ikey
END IF
ELSE
  kstr(i)=ikey
RETURN
END IF
CONTINUE
 ELSE
  ier0od=2011
  CALL msgprt(*990,ier0od)
  CALL syserr(*990,'ST5ENC')
END IF
RETURN
990 ierrod=ier0od
RETURN 1
END
... *** Subprogram for converting a vector of numeric keys
... *** to a character string
... ***

... INPUT BY PARAMETERLIST

KSTR    I        Numeric keystore
KEYDIM  I        Dimension of KSTR

... OUTPUT BY PARAMETERLIST

STRING C(*)    Character string

... Special Case

KSTR    I        Is a vector of four keys
KSTR(1) I        Is calculated from the first 5 characters of file
KSTR(2) I        Is calculated from the last 3 characters of file
and the type of the element
KSTR(3) I        Is calculated from the first 3 characters of element
KSTR(4) I        Is calculated from the last 3 characters of element
and the 2 characters of version

... RETURN    normal exit
... RETURN 1   error exit

SUBROUTINE st5dec(*,ierrod,kstr,keydim,string)

CHARACTER*(*) string
CHARACTER*(1) qohr
DIMENSION kstr(keydim)

INCLUDE ias-util.dclcommsg,LIST
INCLUDE ias-util.comcommsg,LIST
INCLUDE ias-util.dclcsign,LIST
INCLUDE ias-util.comcsign,LIST

ierOod=0
length=len(string)

... Conversion of kstr to string

IF(length.GE.((keydim-1)*5+1))THEN
  string = ' ';
  DO 320 i=1,keydim
       ikey=kstr(i)
    DO 300 m=1,5
         mpos=5*(i-1)+m
         string(5*mpos) = ikey(m)
      320 CONTINUE
    300 CONTINUE
  320 CONTINUE
  RETURN
ENDIF
IF(mpos.LE.length)THEN
  intden=64**(5*i-mpos)
  num=ikey/intden
  string(mpos:mpos)=qchr(num)
  ikey=ikey-num*intden
  IF(ikey.EQ.0.AND.1.E1.keydim)THEN
    GOTO 340
  END IF
ELSE
  CALL syserr(*990,'STRCON')
END IF

300    CONTINUE
320    CONTINUE
ELSE
  ier0cd=2021
  CALL msgprt(*990,ier0cd)
  CALL syserr(*990,'ST5DEC')
END IF

340    CONTINUE
RETURN
990    iercd=ier0cd
RETURN 1
END
... *** SUBROUTINE to encode an arbitrary string in processor ***
... *** dependent code into an IAS-SYSTEM code by use of ***
... *** FORTRAN 77 function ICHAR ***
...

... COMMENT

Four characters are encoded in every word, using eight bits per character. The sign bit has to be treated in a special way for the sake of words with only 32 bits, one of which is the sign bit.

... INPUT BY PARAMETERLIST

STRING C(*) String to be converted
IDIM I Dimension of vector ISTRCD

... OUTPUT BY PARAMETERLIST

ISTRCD I(IDIM) Converted string

... RETURN Normal exit
... RETURN 1 Error exit
...

... SUBROUTINE st4enc (*,ierroc,string,idim,istroc)

CHARACTER*(*) string
LOGICAL minus
DIMENSION istroc(idim)

INCLUDE ias-util.comstatus,LIST

ierroc=0
iword=1
ibyte=0
CALL vecnul(istroc,1,idim)

length=LEN(string)
umwrd=(length+3)/4

IF (idim.NE.numwrd) THEN
  CALL syserr('990,'ST4ENC')
END IF

... Loop over number of characters in input string

DO 600 i0=1,length
  i0char=ICHAR(string(i0:i0))

600
C ... Character out of range?
C
  IF (i0char.GT.254) THEN
    i0char=0
    ierrod=1
    CALL syserr(*990, 'ST4ENC')
  END IF
C
  ibyte=ibyte+1
C
C ... New word?
C
  IF (ibyte.EQ.5) THEN
    ibyte=1
    iword=iword+1
  END IF
C
C ... New word, including first word?
C
  IF (ibyte.EQ.1) THEN
    ivalue=0
    IF (i0char.GT.127 .AND. isize(13).LT.32) THEN
      minus=.TRUE.
      i0char=254-i0char
    ELSE
      minus=.FALSE.
    END IF
  END IF
C
C ... Add value for that character in correct byte and with correct
C ... sign: negative sign if the first character of a word results
C ... in a value of the ICHAR function that is greater than 127
C
  iadd=i0char*2**(8*(4-ibyte))
  IF (minus) THEN
    ivalue=ivalue-iadd
  ELSE
    ivalue=ivalue+iadd
  END IF
  istrod(iword)=ivalue

600  CONTINUE
C
C RRETURN
C
990  RRETURN 1
END
SUBROUTINE to convert the code generated by subroutine ST4ENC back to a character string in processor dependent code, using FORTRAN 77 CHAR-function

COMMENT

INPUT BY PARAMETERLIST
ISTRCD I(IDIM) Code generated by ST4ENC
IDIM I Number of words to be converted

OUTPUT BY PARAMETERLIST
STRING C(*) Output string

RETURN Normal exit
RETURN 1 Error exit

SUBROUTINE st4dec(*,ierroc,ISTRCD,IDIM,string)

CHARACTER(*) string
LOGICAL minus
DIMENSION ISTRCD(IDIM)

ierrcd=0
iword=1
ibyte=0
string=' '

length=LEN(string)
umwrd=(length+3)/4
IF (IDIM .NE. NUMWRD) THEN
   CALL syserr('*990,'ST4DEC')
END IF

DO 600 i0=1,length
   ibyte=ibyte+1
C ... New word (but not first word !) ?
   IF (ibyte.EQ.5) THEN
      iword=iword+1
      iword=word+1
   END IF
C ... New word, including first word ?

C
IF (i(byte).EQ.1) THEN
  ivalue=istod(i(word))
  IF (i(word).LT.0) THEN
    minus=.TRUE.
    ivalue=-ivalue
  ELSE
    minus=.FALSE.
  END IF
END IF

i(div)=2**((8*(4-i(byte)))
i0char=ivalue/i(div)
ivalue=ivalue-i0char*i(div)
IF (i(byte).EQ.1.AND.minus) THEN
  i0char=254-i0char
END IF

IF (i0char.GT.0) THEN
  string(i0:10)=CHAR(i0char)
ELSE
  string(i0:10)=''
END IF

600 CONTINUE
C
C
RETURN
C
990 RETURN 1
END