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)

Contents

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 semiportable 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.

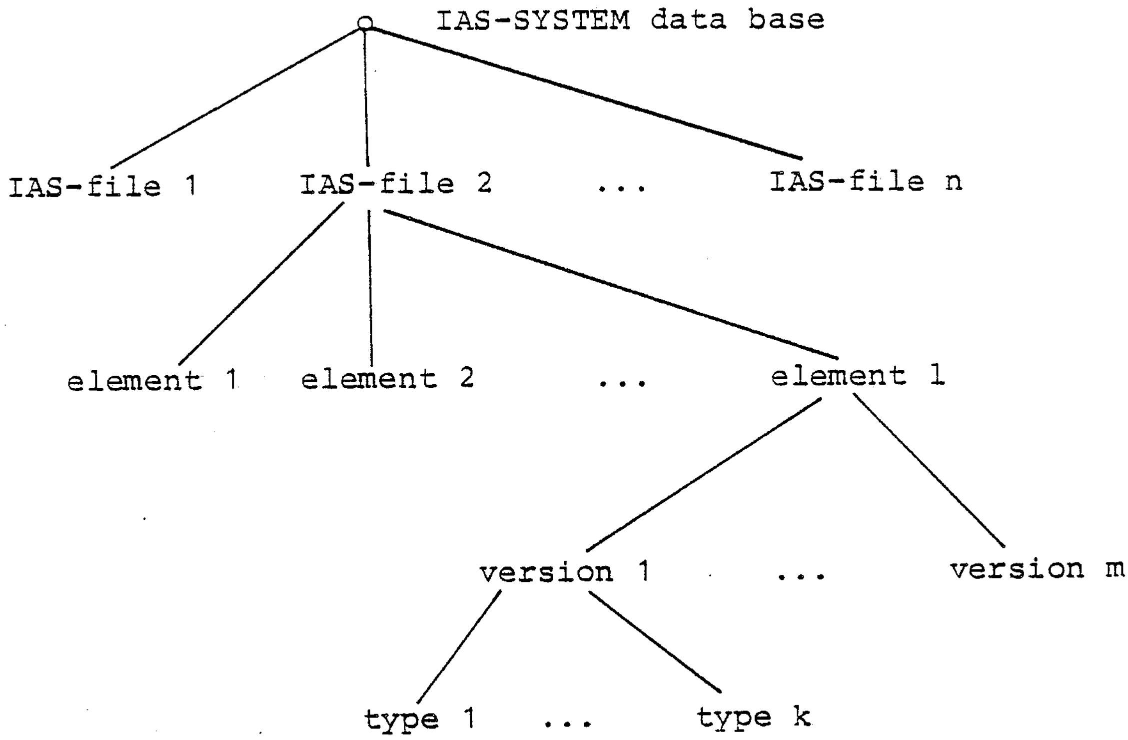
2. Requirements for the IAS-SYSTEM Data Base

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.

Figure 1:



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*) 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

---

*) 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 personnal permanent working area of the user: to fortify the personnal 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 detailled 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 1000 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

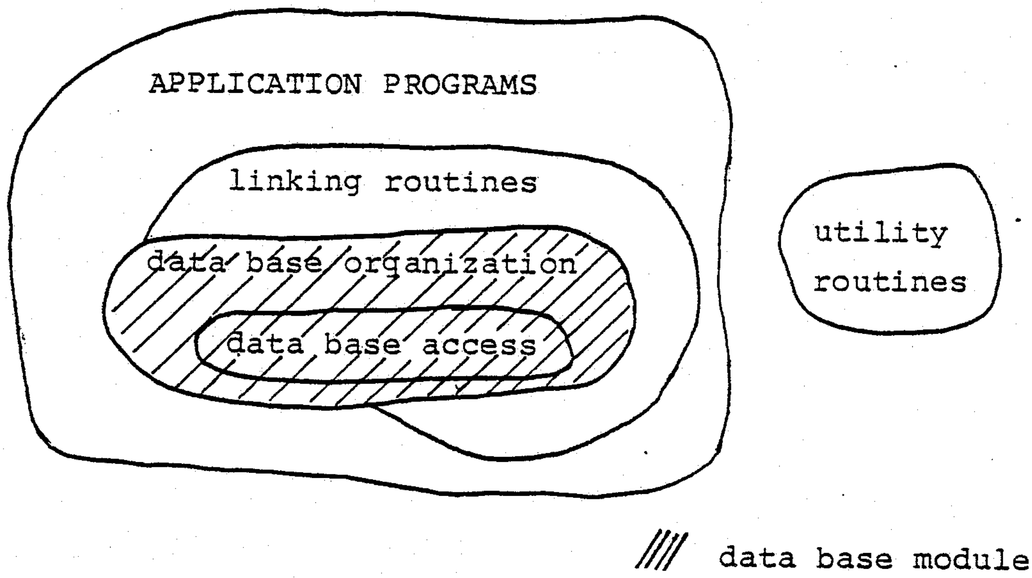
Obviously the operation on sets is defined by the specification of a partial key. In the first example all equations of the specified file have to be deleted, the specification of an element name and a version is omitted. One solution to this partial key match problem is a search for all matching keys 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 quaranteed.

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



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 data base specific information maintained in the array IDBST. Whenever a data base is assigned this array is copied from mass-storage to main-storage thus allowing the use of different data bases with one and the same IAS-SYSTEM.

Additionallv the array IDBST can be used to support security concepts. Whenever one of the data base 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 data base module.

3.2.1 Access Operations for Individual Data Base 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 Data Base 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^*-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^*-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^*-tree simulates an inversion routine perfectly. (On B^*-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,α) where x is called a key and α 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.

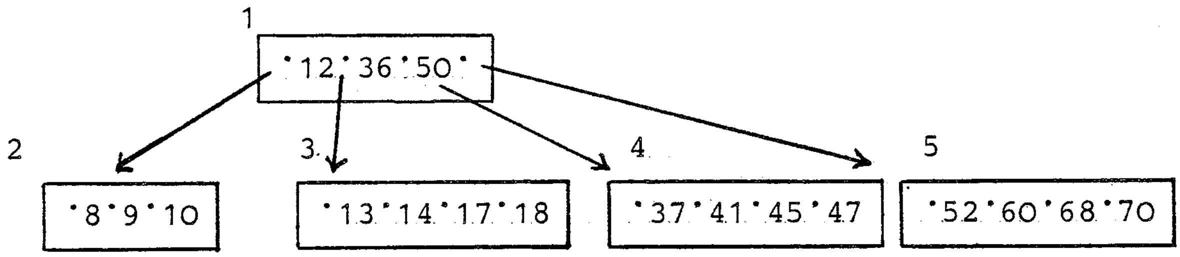
1) Each path from the root to the leaf has the same length h, also called the height of the B-tree.

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:

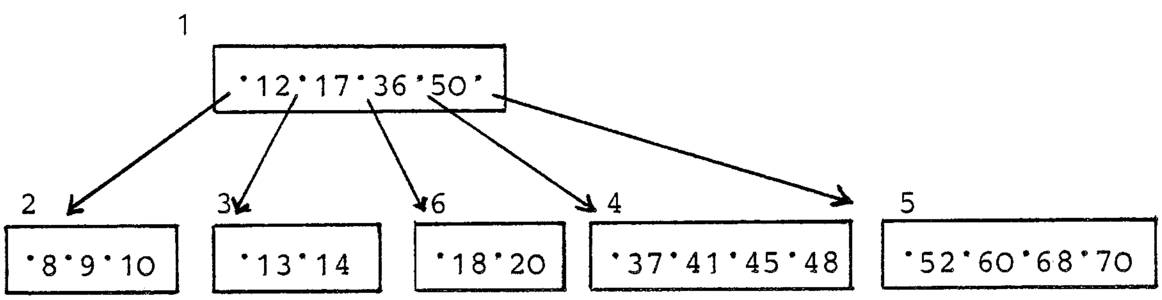


To retrieve the information associated with key 9 we look at the root: as 9 is less than 12 we follow the pointer to record 2 where we find the key 9 and the corresponding information.

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.

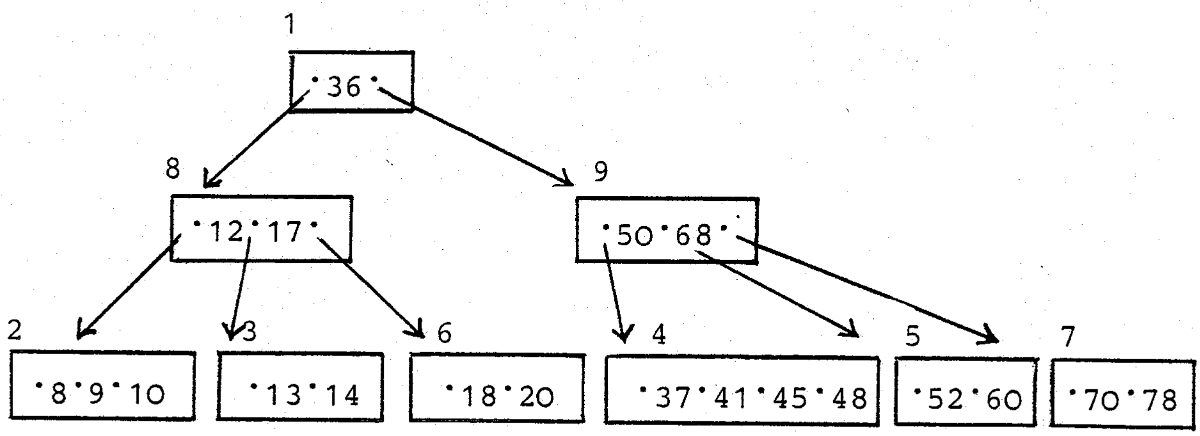
The interesting problem is what happens when the leaf is already full. In our example this will be the case when we try to insert the key 20. Here the leaf has to be split into two leaves. Fig. 4 shows the resulting B-tree

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.

Figure 5:

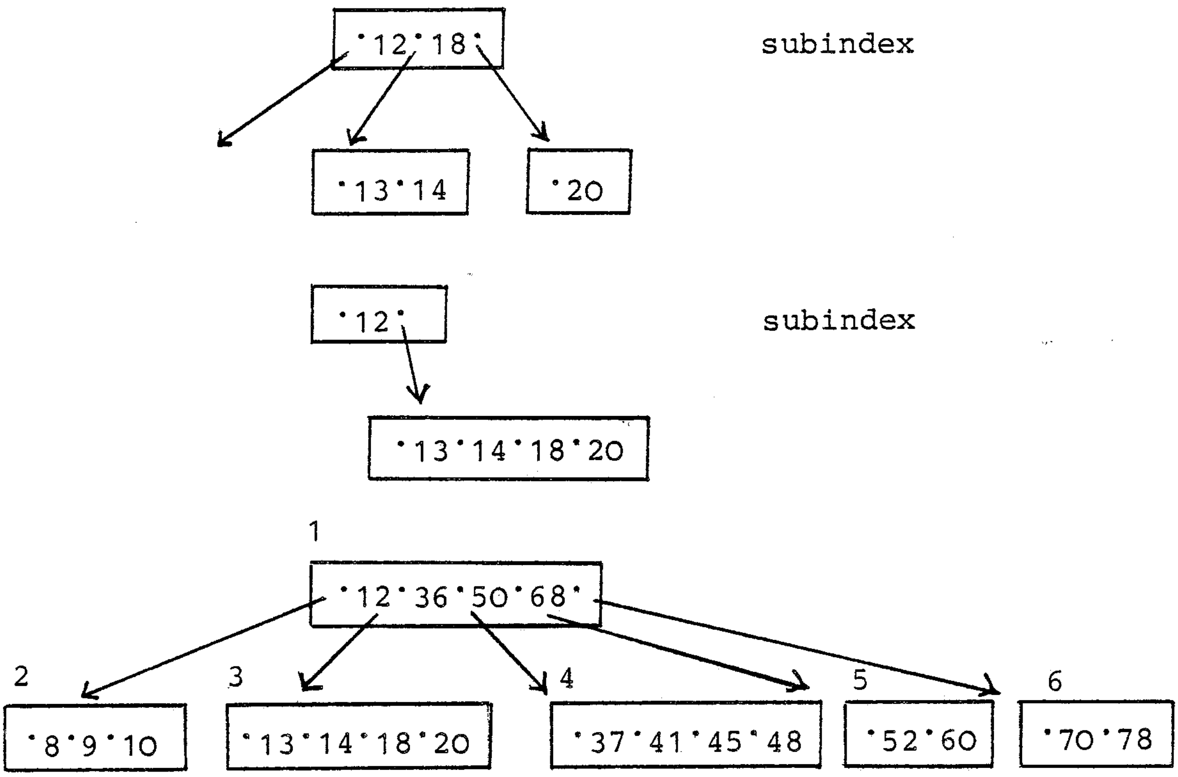


The result is again a balanced tree, i.e. every path from the root to a leaf has the same length.

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.

Figure 6:

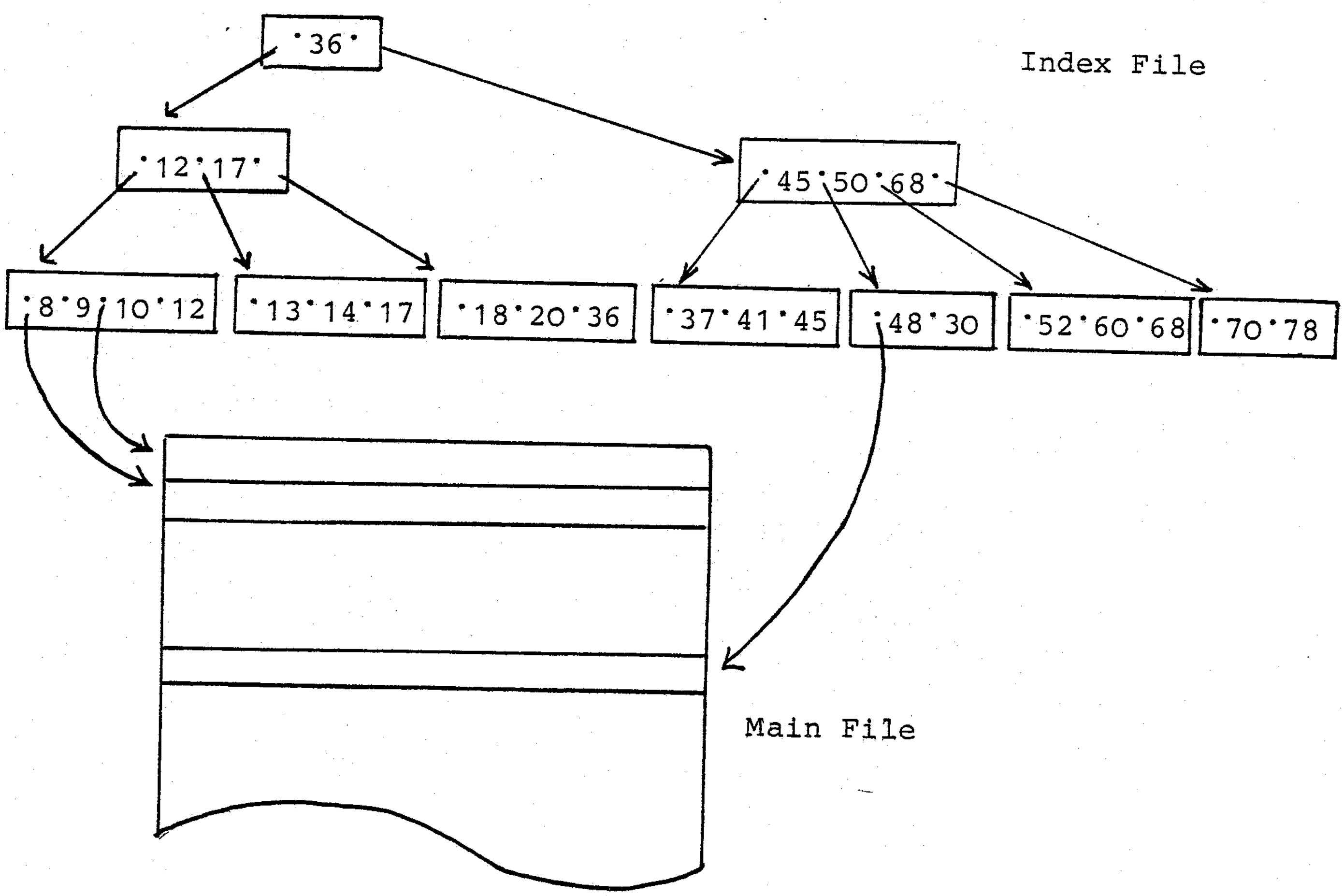


These examples show that the B-trees grow and contract in only one way, namely nodes split off a brother or two brothers are merged or catenated into a single node. The splitting and catenation processes are initiated at the leaves only and are propagated towards the root. If the root node splits a new 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

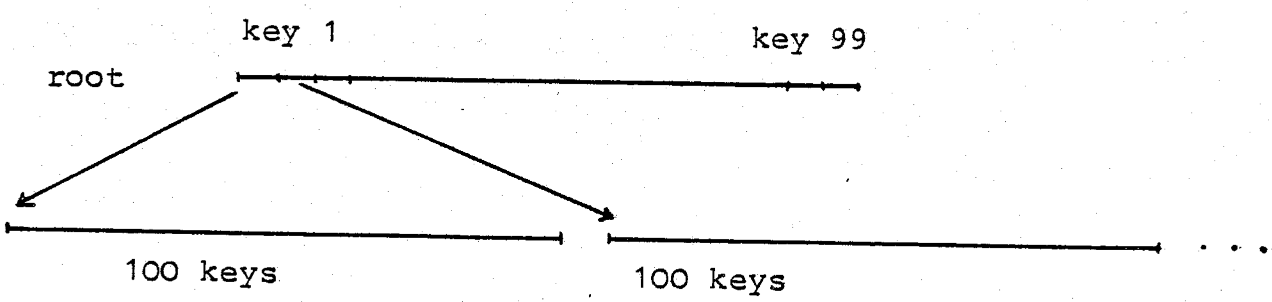
- the maintenance algorithms have a higher complexity
- in case of data base enlargement it would be necessary to reorganize the B^*-tree organization on the index file

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 leafs. 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 <u>The Usage of System Buffers</u>

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.

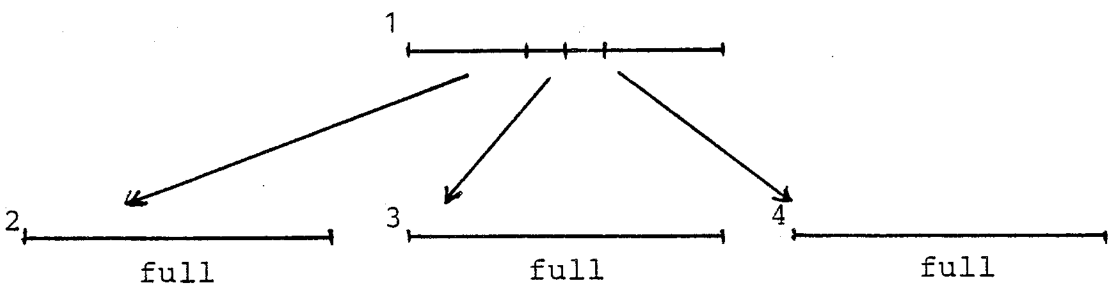
Now if we try to keep the root always in main storage actually only one access is needed to find out that a specified key is or is not in the data base. If the recently used leaf is also kept in main storage many IAS-items (in the best case exactly 100) can be processed without additional access to the 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

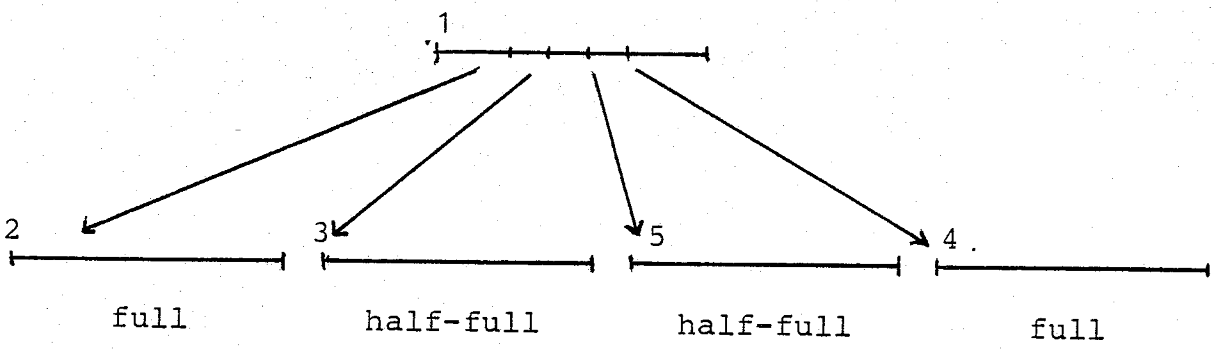
The worst case behavior for insertion is easily outlined by Fig. 9 and Fig. 10

Figure 9:



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.

Figure 10:



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 <u>The File on Unit 11</u>

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

<u>Figure 11</u>

Rec #				
1.	Data base Control Information			
2.	Info 1. IAS-file	Info 2. IAS-file	. . .	Info 10. IAS-file
	Info 11. IAS-file	. . .		
21.	. . .			Info 200. IAS-file
22.	Root of B^*-tree			
23.	1. Leaf of B^*-tree			
⋮	2. Leaf of B^*-tree ⋮			

4.1.1 The First Record

This record contains the data base specific control information, i.e. the array IDBST in the first 40 words and another data base 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*-tree in file 11
IDBST(4)........ Pointer to first leaf in B*-tree in file 11
IDBST(5)........
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(10).......
IDBST(11)....... Factor for computing size of file 12 and 13
IDBST(12)....... Record length of file 12
IDBST(13)....... Number or records of file 12
IDBST(14).......
IDBST(15).......
IDBST(16)....... Record length of file 13
IDBST(17)....... Number of records of file 13
IDBST(18)....... Length of integer info of one record
IDBST(19)....... Length of character info of one record
IDBST(20).......
IDBST(21)....... Pointer to next free record of B*-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(24).......
IDBST(25).......
IDBST(26)....... Number of catalogued IAS-files in file 11
IDBST(27).......
IDBST(28).......
IDBST(29).......
IDBST(30).......
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(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^*-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).......
IROOT(6).......
IROOT(7)....... For further statistical use
IROOT(8).......
IROOT(9).......
IROOT(10)...... 1st pointer (less than 1st key)
IROOT(11)......
IROOT(12)......
IROOT(13)...... 1st key
IROOT(14)......
IROOT(15)...... 2nd pointer (greater than 1st key)
IROOT(16)......
IROOT(17)......
IROOT(18)...... 2nd key
IROOT(19)......
IROOT(20)...... 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^*-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)..... 2rd 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 data base 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.

numeric storage units	
1	Continuation pointer to the next record, if more than one record is needed for storing the data base item
2	backward concatenation pointer to ease the data base 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 data base 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 data base item.
The number of records available is depending on the data base 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 records 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 information 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. Additionally 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 in-termin 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 nonporta-ble "standard modules" which are used in other parts of the IAS-SYSTEM as well, e.g. for the input logfile, output log-file 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 periode and end periode. 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

" ","A",...,"Z","Ø",...,"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
"$" → 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:

1.word	2.word	3.word	4.word
E | C | O | N |	| | | D | S	A | 1 | | |	| | | |

filename · type "DATA" · subtype "SER" · element name · version

Numeric representation

1.word	84734848	$= 5\cdot64^4+3\cdot64^3+15\cdot64^2+14\cdot64$
2.word	275	$= 4\cdot64+19\cdot64^0$
3.word	24117248	$= 1\cdot64^4+28\cdot64^3$
4.word	Ø	

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.

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

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 data base 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(6) and IDBST(7) by changing the PARAMETER statement and recompiling the whole System.

5.3 The I/O Buffers

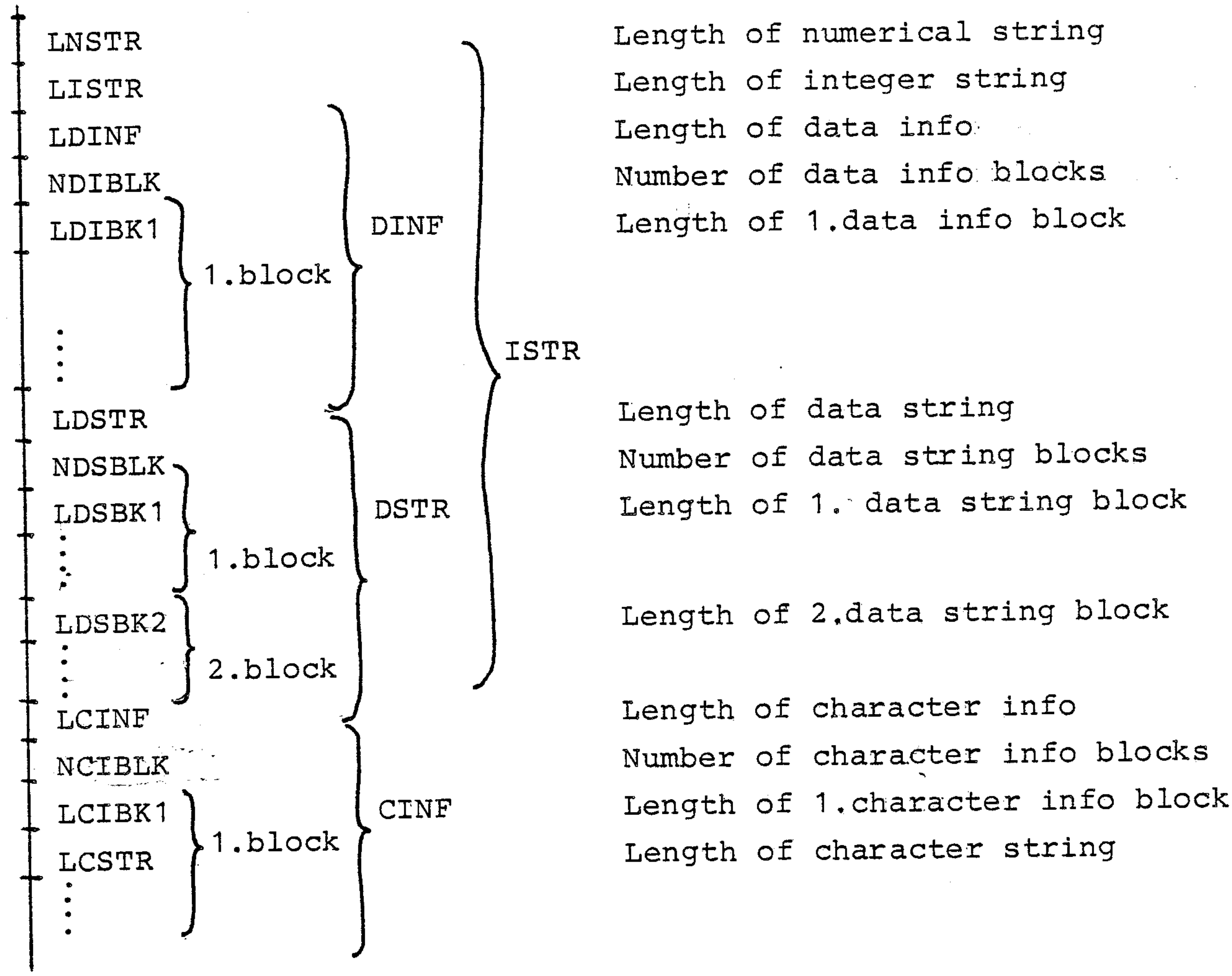
As mentioned above the information transmitted to or from the data base can be divided into numerical values and characters. Because the FORTRAN 77 standard does explicitely not specify the relation between numeric and character storage units, it is necessary to separate these two kinds of information.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



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	DINF
7	LISTR	
3	LDINF	
1	NDIBLK	
1	LDIBK1	
3	LDSTR	
1	NDSBLK	DSTR
1	LDSBK1	
4	LCINF	CINF
1	NCIBLK	
2	LCIBK1	
Ø	LCSTR	

The numerical string for a time series containing three values for the time period 1975 to 1977 may have the following form.

32	LNSTR
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)
Ø	not used
Ø	
Ø	Input from terminal
1	Number of values per year
1975	Start time
1977	End time
Ø	Start time of forecasted values
Ø	End time of forecasted values
Ø	not used
Ø	
Ø	
Ø	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
IHIRCD ... 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 IHIRCD 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

Code	Error condition
5020	illegal hierarchy and/or operation code
5051	no IAS-file assigned at all
5160	specified IAS-file not assigned
5280	read/write permission of IAS-file does not match

The Hierarchy Code

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

Bit	Corresponding identifier part
Ø	version
1	element
2	subtype
3	type
4	file

If the user specified the filename only, e.g. in case of *PRT,T F. then the 4^{th} 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 3^{rd} bit and 4^{th} 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 1^{st} bit and the 3^{rd} 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

Value	Affected data base level	
Ø	OS-file	no identifier is specified
16	IAS-file	the requested operation has to be performed for all IAS-items of an IAS-file either or concerns the handling of an IAS-file
24	type	perform the data base operation for all IAS-items of a special type
28	subtype	
30	element	perform the data base operation for all versions of a certain IAS-item
31	version	fully specified identifier

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.

IHIRCD=Ø	All operations are requested for OS-Files	
IOPTCD		Examples
2	Delete the data base	*DB,D
3	Free data base	*DB,F
4	Catalog new data base	*DB,C
5	Assign data base read-only	*DB,R
6	Assign data base write-enabled	*DB,W
11	Dump of OS-file 11	*DUMP 11
12	Dump of OS-file 12	*DUMP 12
13	Dump of OS-file 13	*DUMP 13

IHIRCD=16	All operations are requested for IAS-Files	
IOPTCD		Examples
-1	Define IAS-file as BFILE	*F,B
-2	Define IAS-file as DFILE	*F,D
-4	Define IAS-file as EFILE	*F,E
-8	Define IAS-file as MFILE	*F,M
-16	Define IAS-file as SFILE	*F,S
-1024	Assign IAS-file write enabled	*F,W
-2048	Assign IAS-file read only	*F,R
-4096	Free IAS-file	*F,F
-8192	Catalog IAS-file	*F,C

IHIRCD=16	All operations are requested for IAS-Files	
IOPTCD		Examples
Ø	Read the characteristics of all cataloged IAS-files	*PRT,C
6	Print all items of an IAS-file	*PRT
7	Delete all items of an IAS-file	*DEL
8	Copy IAS-file to IAS-file	*COPY
9	Copy IAS-file to OS-file	*COPY,O
10	Copy OS-file to IAS-file	*COPY,I

IHIRCD=31 All operations are requested for a single item

IOPTCD		Examples
∅	Find an IAS-item	
1	Read both NUMSTR and CSTR	*SER
2	Read only NUMSTR(*)	*MOD,S
3	Insert an IAS-item	*SER,I
4	Update both NUMSTR and CSTR	*UPD
5	Update only NUMSTR(*)	*MOD,S

IHIRCD≥16

6	Print a table	*PRT
7	Delete a (group of) IAS-item(s)	*DEL,E
8	Copy a (group of) IAS-item(s)	*COPY,D

*) 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.

Value	Operation
∅	The access is done by searching the B^*-tree
1	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

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/IOPTCD 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
IOPTCD	... 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 data base operations, i.e. Ø-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 data base item to be handled but a starting address for the search for the next data base item matching the specified (partial) key.

The parameter KIDNTF must not contain the name of an IAS-file or a data base identifier. Only data base 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

American National Standards Institute, Inc. (ANSI, ed.): ANSI X3.4-1977 American Standard Code for Information Interchange. New York 1977

American National Standards Institute, Inc. (ANSI, ed.): ANSI X3.9-1978 American National Standard Programming Language FORTRAN. New York 1978

BAYER, R. and E. McCREIGHT: Organization and Maintainance of Large Ordered Indices. Acta Informatica 1(3), 1972

DATE, C.J.: An Introduction to Data Base Systems. Reading, Addison Wesley 1976

HÄRDER, T.: Implementierung von Datenbanksystemen. München-Wien, Carl Hanser Verlag, 1978

KNUTH, D.E.: The Art of Computer Programming, Vol. 3: Sorting and Searching. Reading, Addison Wesley 1972

LARMOUTH, J.: Fortran 77 Portability. Software-Practice and Experience 11(10), 1981

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

PLASSER, K.: User Reference Manual - Part One. Institutsarbeit No. 129, Institute for Advanced Studies, Wien, 1980

PLASSER, K., H. SONNBERGER, K. RODLER and W. PHILIPP: On Writing a 'Comprehensive', 'Interactive', 'Portable', 'Data Base Oriented' and 'Reliable' Program System for Econometric Modeling and Corporate Planning (Interim Report, Research Memorandum No. 169, Institute for Advanced Studies, Wien, 1982.

ROSENBERG, A.L. and L. SNYDER: Time- and Space-Optimality in B-Trees. ACM TODS, Vol. 6, 1, 1981

SAVORY, St. E.: CPU Times Spent Searching B^*-trees. Angewandte Informatik 10, 1981

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

ULLMAN, J.D.: Principles of Database Systems. London, Pitman Publishing Limited, 1980

WEDEKIND, H.: Datenbanksysteme I (II). Mannheim-Wien-Zürich, Bibliographisches Institut, 1974 (1976)

A P P E N D I X

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 -
            DBLOOK -
            DBDEL -
            DBCOPY -
          PRTDBI -
          DELDBI -
          COPDBI -
      AGGREG
        COMTM
        AGGSAS
        AGGDEF
    SPØ76
      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 importance.

Common utility routines (e.g. VECCOP for copying a vector of integers) are neither mentioned nor listed.

```
C ... ***  Program for controlling the processing of *SER-COMMAND    ***
C ... ***  the universial command for the handling of time series    ***
C ... ***
C ...
C
C
C ... COMMENT
C
C     The checking of the command string is done in subroutine SERCC
C     (see there also for the allowed options and fields/subfields
C     and their meaning).
C
C     Depending on the specified option for printing or storing
C     different time definitions are taken (see comment of SERSTO).
C
C     As a consequence of the further command requirements, this
C     subroutine can be divided into three parts:
C     - insertion of time serie(s) via batch-mode
C       (IOPTCD=0, INPOPT=1, subroutine SERBTI)
C     - interactive insertion of a single time series
C       (IOPTCD=0, INDPRT>1, subroutine SERSTO)
C     - printing of a time series
C       (IOPTCD><0, subroutine SERPRT)
C
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     IERRCD  I          Error code
C
C
C
C ... RETURN             Normal exit
C ... RETURN1            Error exit
C ...
C
C
      SUBROUTINE ser(*,ierrcd)
C
      CHARACTER*80 calstr
      DIMENSION kidntf(4),kdflsr(8),iadr(5)
C
      ierrcd = 0
C
C ... check command string
C
      calstr = ' '
      CALL vecnul(kdflsr,1,8)
      CALL sercc(*980,ierrcd,ioptcd,indprt,inpopt,kidntf,iadr,kdflsr
     *    ,iaggm,calstr)
C
C ... execute
C
      IF(ioptcd.eq.0) THEN
C
```

```
C ... store option has been specified
C
          IF(inpopt.eq.1) THEN
C
C ... input from data deck in batch mode
C
             CALL serbti(*980,ier0cd)
          ELSE
             CALL sersto(*980,ier0cd,inpopt,kidntf,iadr,kdflsr,iaggm
     *          ,calstr)
             IF(ier0cd.eq.-2) GOTO 970
          ENDIF
       ELSE
C
C ... print specified time series
C
          CALL serprt(*980,ier0cd,ioptcd,indprt,kidntf)
       ENDIF
C
       RETURN
C
C ... error section
C
970    CALL msgprt(*990,1501)
980    ierrcd = ier0cd
990    RETURN1
       END
```

```
C ... ***  Subroutine for reading a time series from mass storage     ***
C ... ***
C ...
C
C
C
C ... COMMENT
C
C     The time series is either read from the data base or from the
C     working-scratch file. The purpose of reading may either be
C     the destination of the time range the series is defined, or
C     the evaluation (aggregation) of the data.
C
C
C
C ... INPUT BY PARAMETERLIST
C
C     KIDNTF  I(4)       - Integer-5-coded time series identifier
C                        - identifier for intrinsic variable
C                          (1), (2) ... 0
C                          (3) ........ neg. log. address of variable
C                          (4) ........ 0
C     INPOPC  I          Operation code
C                        1 ... numeric part and character part is read
C                        2 ... only numeric part is read
C     INPCUR  I          Address currency indicator
C                        0 ... access to be done via KIDNTF
C                        1 ... access to be done via IADR
C
C
C ... INPUT BY PARAMETER STATEMENT COMPUND/pardata/
C
C     LNMSTR  I          Dimension of NUMSTR
C     LCSTR   I          Number of character storage units of CSTR
C
C
C ... INPUT BY COMMON/calc1/
C
C     INTTBL  I(NUMINT) Table of intrinsic variables
C
C
C ... INPUT BY COMMON/status/
C
C     ISWTCH  I(20)      (4) ...Debug switch
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     DATVEC  R(*)       Data vector containing the prepared time series
C                        ATTENTION: DATVEC must not be equal to NUMSTR
C                                   in COMMON /DBNINF/
C     HEADER  C*(*)      Header description of time series
C     MODAGG  I          Aggregation mode
C                        <0 ... undefined
C                         0 ... sum
```

```
C                         1 ... average
C                         2 ... stock
C                         3 ... deflator
C     IERRCD  I          Error code
C
C
C ... OUTPUT BY COMMON/dbcinf/
C
C     CSTR    C*LCSTR    Item header
C
C
C ... OUTPUT BY COMMON/dbninf/
C
C     NUMSTR  I(LNMSTR) Numerical string containing the full item
C                        information
C
C
C ... TRANSPUT BY PARAMETERLIST
C
C     LDATVC  I          I: >0 ... maximum length the prepared
C                                  (aggregated) time series may have
C                           =0 ... no data requested, only the
C                                  common time range
C                        O: number of values in DATVEC
C     ITMDEF  I(3)       Time definition
C                        I: time range desired
C                        O: common time range of desired range and range
C                           the time series is defined for
C     IADR    I(5)       Data base or external working storage access
C                        addresses
C                        I: in case of INPCUR=1
C                        O: in case of INPCUR=0
C
C
C
C ... RETURN             Normal exit
C ... RETURN1            Error exit
C ...
C
      SUBROUTINE serrd(*,ierrcd,kidntf,inpopc,inpcur,datvec,header
     *   ,modagg,ldatvc,itmdef,iadr)
C
      CHARACTER*(*) header
      CHARACTER*21 qidedt
      LOGICAL inddb
      DIMENSION kidntf(4),datvec(*),itmdef(3),itmser(3),iadr(5)
     *   ,kaggid(8)
C
      INCLUDE ias-util.parcalc,LIST
      INCLUDE ias-util.pardata,LIST
      INCLUDE ias-util.comdbninf,LIST
      INCLUDE ias-util.dcldbcinf,LIST
      INCLUDE ias-util.comdbcinf,LIST
      INCLUDE ias-util.comstatus,LIST
      INCLUDE ias-util.dclcommsg,LIST
```

```
      INCLUDE ias-util.comcommsg,LIST
      INCLUDE ias-util.dclcsign,LIST
      INCLUDE ias-util.comcsign,LIST
      INCLUDE ias-util.dclcalc1,LIST
      INCLUDE ias-util.comcalc1,LIST
      ierrcd = 0
      icur = inpcur
      ioprtc = inpopc
C
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,ier0cd,iadr(1),iadr(2),numstr)
         CALL veccop(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,ier0cd,kidntf,ioprtc,icur,numstr,cstr,idatab
     *      ,itmser,modagg,kaggid,iadr)
C
C ... set error code for dummy series
C
         IF(numstr(6).eq.0) ierrcd = -1
         inddb = .TRUE.
      ENDIF
C
C ... DEBUG of NUMSTR and CSTR
C
      IF(iswtch(4).ne.0) THEN
         msgnum = 7920
         CALL ncdeb(*990,numstr,cstr,msgnum)
      ENDIF
C
      IF(ierrcd.eq.0) THEN
C
C ... no time range stored in the fetched time series
C
         IF(itmser(1).le.0) THEN
            ierrcd = 7291
            GOTO 970
         ENDIF
C
```

```
C ... data exists in time series (no dummy series)
C
           IF(ldatvc.gt.0) THEN
C
C ... transfer (with aggregation) data to output vector DATVEC
C
              CALL aggreg(*980,ier0cd,inddb,kidntf,itmser,modagg,kaggid
     *           ,rstr(idatab),datvec,ldatvc,itmdef)
C
           ELSE
C
C ... only time range of time series is requested
C
              IF(itmser(1).lt.itmdef(1)) THEN
                 ierrcd = 7311
                 GOTO 970
              ENDIF
C
              CALL comtm(itmser,itmdef)
           ENDIF
        ENDIF
C
C ... transfer header
C
        IF(inddb.and.inpopc.eq.1) THEN
           ihelp = LEN(header)
           IF(ihelp.gt.76) ihelp = 76
           header(1:ihelp) = cstr(1:ihelp)
        ENDIF
C
        RETURN
C
C ... error section
C
970     IF(inddb) THEN
           msgin = qidedt(kidntf,4)
        ELSE
           msgin = intvtb(ABS(kidntf(3)))
        ENDIF
        msgin(22:22) = msgstp
        CALL msgprt(*990,ierrcd)
C
980     ierrcd = ier0cd
990     RETURN1
        END
```

```
C ... ***    Unique interface to the data base organization module   ***
C ... ***
C ...
C
C
C ... COMMENT
C
C     The data base organization modul performs:
C
C     - handling of data base as a whole (OS-file handling)
C       . catalog
C       . assign
C       . free
C       . delete
C       . enlarge
C       . recover
C       . single record dump
C     - handling of IAS-files
C       . catalog
C       . assign
C       . free
C       . use default file
C       . access permission check
C     - handling of IAS-items
C       . set access (partial key)
C         .. delete
C         .. print
C         .. copy
C       . single item access (full key)
C         .. find
C         .. retrieve
C         .. insert
C         .. update
C
C
C
C ... INPUT BY PARAMETERLIST
C
C     IHIRCD  I         Hierarchy code (bit combination)
C                         bit
C                         set
C                          1  ........ version
C                          2  ........ element
C                          3  ........ subtype
C                          4  ........ type
C                          5  ........ file
C                          no ........ data base
C     IOPRTC  I         Operation code,
C                       must be seen in connection with hierarchy code
C
C                       IHIRCD = 0
C                         1 ........ data base recovery (save)
C                         2,3 ...... data base close
C                         4-7 ...... data base open (catalog)
C                         11-13 .... dump facility
```

```
C                        IHIRCD = 16 and IOPRTC = 0
C                          print list of cataloged files
C                        IHIRCD = 16 and IOPRTC < 0
C                          bit combination indicating the IAS-file
C                          handling operation (see subroutine FBRNCH)
C                        16 <= IHIRCD <= 31 (operation on item set)
C                          6 .... print item list
C                          7 .... delete
C                          8 .... internal copy
C                          9 .... copy out
C                          10 ... copy in
C                        IHIRCD = 31 (operation on single items)
C                          0 .... find
C                          1 .... retrieve whole item
C                          2 .... retrieve numerical information only
C                          3 .... insert item
C                          4 .... update whole item
C                          5 .... update numerical information only
C                          7 .... delete
C                          8 .... internal copy
C                          9 .... copy out
C                          10 ... copy in
C
C
C ... INPUT BY COMMON /status/
C
C      IDBST   I(40)     Data base status array
C                        (3) pointer to root record
C
C
C ... INPUT BY COMMON /intctt/
C
C     IRWPRM  I(0:10)    Read/write permission of assigned IAS-files
C
C
C ... INPUT BY PARAMETERCOMPOUND /parrecl/
C
C      LREC14  I         Record length of file 14
C
C
C ... TRANSPUT BY PARAMETERLIST
C
C     ICUR    I          Currency indicator
C                        if set, access is done via addresses instead
C                        of item identifier
C     IADR    I(*)       Address vector for direct access without search
C     KIDNTF  I(*)       Integer-coded identifier,
C                        Integer-4-coded data base name
C                        Integer-5-coded IAS-file or partial/full
C                        specified item identifier
C     NUMSTR  I(*)       Numerical data base I/O-buffer
C                        in case of print-operation the coded print-size
C                        option is transmitted in (1)
C     CSTR    C*(*)      Character data base I/O-buffer
```

```
      SUBROUTINE dborg(*,ierrcd,ihircd,ioprtc,icur,iadr,kidntf
     *    ,numstr,cstr)
C
      CHARACTER*(*) cstr
      CHARACTER*21 qidedt
      DIMENSION ivalid(0:10),iadr(*),iadr1(1),kidntf(*),numstr(*)
     *    ,indfct(2)
      LOGICAL back11
      INCLUDE ias-util.dclcommsg,LIST
      INCLUDE ias-util.comcommsg,LIST
      INCLUDE ias-util.dclcsign,LIST
      INCLUDE ias-util.comcsign,LIST
      INCLUDE ias-util.comstatus,LIST
      INCLUDE ias-util.comintctt,LIST
      INCLUDE ias-util.parrecl,LIST
      COMMON /idummy/ ibuff(lrec14)
      DATA ivalid /3*1,2,2*0,1,2,1,2,1/
C
      ierrcd = 0
C
      IF(ihircd.eq.0)THEN
C
C ... hierarchy code 0: operation affects whole data base
C
         IF(ioprtc.ge.4.and.ioprtc.lt.7) THEN
C
C ... open data base
C
            CALL dbopen(*9900,ier0cd,ioprtc,kidntf,iadr)
            ierrcd = ier0cd
         ELSE IF(ioprtc.ge.2.and.ioprtc.lt.4) THEN
C
C ... close data base
C
            CALL dbclos(*9900,ier0cd,ioprtc,kidntf)
         ELSE IF(ioprtc.ge.11.and.ioprtc.lt.14) THEN
C
C ... dump (parts) of the data base files
C
            CALL dumpdb(*9900,ier0cd,ioprtc,iadr)
         ELSE
C
C ... error
C
            ierrcd = 5020
         ENDIF
C
       ELSE IF(ihircd.eq.16.and.ioprtc.eq.0) THEN
C
C ... print table of cataloged IAS-SYSTEM-files
C
         CALL prtcat(*9900,ier0cd,numstr(1))
       ELSE
C
C ... search file-control-table for specified file
```

```
C ... get default file if none specified
C
      IF(ioprtc.ne.9) THEN
         ind = 1
      ELSE
         ind = 2
      ENDIF
         CALL fcts(*9900,ier0cd,kidntf((ind-1)*8+1),indfct(ind))
         IF(ioprtc.eq.8) THEN
            CALL fcts(*9900,ier0cd,kidntf(9),indfct(2))
         ELSE
            ind = MOD(ind,2)+1
            indfct(ind) = 0
         ENDIF
C
         back11 = .FALSE.
         IF(inircd.eq.16.and.ioprtc.lt.0) THEN
C
C ... IAS-SYSTEM-file handling is requested
C
            IF(ioprtc.le.-(2**13)) back11 = .TRUE.
            CALL fbrnch(*9900,ier0cd,kidntf,indfct(1),ioprtc)
C
         ELSE
            IF(indfct(1).ge.0.and.indfct(2).ge.0) THEN
C
C ... for each data base operation remaining the specified
C ... IAS-SYSTEM-file must be assigned and the appropriate read/write
C ... permission must satisfy the requirements of the operation
C
               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
C
               IF(indfct(1).ge.0.and.indfct(2).ge.0) THEN
C
C ... checks are ok
C
                  IF(ioprtc.ge.3.and.ioprtc.le.9.and.ioprtc.ne.6)
     *               back11 = .TRUE.
C
                  IF(ioprtc.lt.6.and.ihircd.eq.31) THEN
C
C ... access (read, write, update) to a single item
C
                     CALL dbacc(*9900,ier0cd,ihircd,ioprtc,icur,iadr
     *                  ,kidntf,numstr,cstr)
C
                  ELSE
C
C ... set start values for partial key search
C
                     icur = 2
                     iadr(1) = idbst(3)
```

```
                      iadr(2) = 0
                      iadr(3) = 0
                      iadr(4) = 10
                      iadr(5) = 10
C
                      IF(ioprtc.eq.6.and.(ihircd.lt.31.and.
     *                  ihircd.ge.16)) THEN
C
C ... print table of data base items
C
                        indprt = numstr(1)
                        CALL prtdbi(*9900,ier0cd,ihircd,ioprtc,icur
     *                    ,iadr,kidntf,indprt,numstr,cstr)
C
                      ELSE IF(ioprtc.eq.7.and.(ihircd.lt.28.and.
     *                  ihircd.ge.16.or.ihircd.eq.31)) THEN
C
C ... delete data base item(s)
C
                        CALL deldbi(*9900,ier0cd,ihircd,ioprtc
     *                    ,indfct(1),icur,iadr,kidntf)
C
                      ELSE IF((ioprtc.ge.8.and.ioprtc.le.10).and.
     *                  (ihircd.ge.16.and.ihircd.lt.28.or.
     *                  ihircd.eq.31)) THEN
C
C ... copy data base item(s) to data base item(s) either
C ... or to/from an OS-file
C
                        CALL copdbi(*9900,ier0cd,ihircd,ioprtc,icur
     *                    ,iadr,kidntf,numstr,cstr)
C
                      ELSE
C
C ... access to data base has been tried with an illegal combination
C ... of hierarchy code and operation code
C
                        ierrcd = 5020
                      ENDIF
                    ENDIF
C
                  ELSE
C
C ... read/write permission of IAS-SYSTEM file does not match
C
                    ierrcd = 5280
                  ENDIF
C
                ELSE IF(ier0cd.eq.5051) THEN
C
C ... no IAS-SYSTEM-file at all is assigned
C
                  ierrcd = ier0cd
                ELSE
C
```

```
C ... the specified IAS-SYSTEM-file is not assigned
C
                 ierrcd = 5160
              ENDIF
           ENDIF
C
C ... rewriting of the first record of file 11 is necessary
C ... to keep consistency of IDBST and the data base
C
           IF(back11) THEN
              iadr1(1) = 1
              CALL dbacc(*9900,ier0cd,1024,11,1,iadr1,kidntf,ibuff,cstr)
              CALL veccop(idbst,1,40,ibuff,1)
              CALL dbacc(*9900,ier0cd,1024,21,1,iadr1,kidntf,ibuff,cstr)
           ENDIF
        ENDIF
C
C ... an error has occured
C
        IF(ierrcd.ne.0) GOTO 9800
C
        RETURN
C
C ... error section
C
9800    IF(ierrcd.eq.5160.or.ierrcd.eq.5280) THEN
C
C ... file not assigned or read/write permission wrong
C
           DO 9810 i0=1,2
              IF(indfct(i0).lt.0) THEN
                 msgin = qidedt(kidntf((i0-1)*8+1),2)//msgstp
                 CALL msgprt(*9990,ierrcd)
                 ierrcd = ierrcd+1
              ENDIF
9810       CONTINUE
        ELSE
C
C ... 5020 ... illegal hierarchy and/or operation code
C
           IF(ierrcd.eq.5020)
     *       WRITE(msgin(1:9),FMT='(2I6,A1)') ioprtc,ihircd,msgstp
           CALL msgprt(*9990,ierrcd)
           CALL syserr(*9990,'DBORG')
        ENDIF
C
9900    ierrcd = ier0cd
9990    RETURN1
        END
```

```
C ... *** Program to search file-control-table KFCT for a specific      ***
C ... *** IAS-system-file                                               ***
C ... ***                                                               ***
C ...
C
C
C ... COMMENT
C
C     IF a file name is specified the routine determines it´s
C     position in the file-control-table KFCT.
C     IF none is specified the program determines the default file
C     (the type requested must be specified in the 2nd word of the
C     file identifier according to the common definiton of a full
C     identifier) and the index of this file in KFCT.
C
C
C ... INPUT BY COMMON/status/
C
C     ISWTCH  I(20)    (13) Data base switch
C                            0 ... no data base assigned
C                           >0 ... data base assigned
C     ISTAT   I(20)    (11) Number of files in file-control-table
C
C
C ... INPUT BY COMMON/intctt/
C
C     KFCT    I(2,11)  File-control-table,
C                      contains the integer-5-coded names of all
C                      assigned IAS-system-files
C     INDUCT  I(0:10)  Use-control-table,
C                      each array element contains the index of
C                      the appropriate use (or default) file in the
C                      file-control-table KFCT
C                      (0) .... for the IAS-system-system-file $SYS
C                      (1) .... for the BASE-file
C                      (2) .... for the DATA-file
C                      (3) .... for the EQUATION-file
C                      (4) .... for the MODEL-file
C                      (5) .... for the SOLUTION-file
C                      (6) .... for the TEXT-file
C                      (7)-(10) not yet used
C
C
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     INDFCT  I        Index of specified (or default) file in KFCT
C     IERRCD  I        Error code
C
C
C ... TRANSPUT BY PARAMETERLIST
C
C     KFILE   I(2)     Interger-5-coded file name
C                      - if a file is specified on input, KFILE is a
```

```
C                         pure input parameter
C                       - if none is specified, the default file is
C                         returned, for this purpose it is necessary
C                         that the desired type is stored at the
C                         end of the second word (see common definition
C                         of a full identifier)
C
C
C ... RETURN            Normal exit
C ... RETURN1           Error exit
C ...
C
      SUBROUTINE fcts(*,ierrcd,kfile,indfct)
C
      CHARACTER*5 ctype
      CHARACTER*1 intfil(1:6)
      DIMENSION kfile(2)
C
      INCLUDE ias-util.comstatus,LIST
      INCLUDE ias-util.comintctt,LIST
      DATA intfil /´B´,´D´,´E´,´M´,´S´,´X´/
C
      ierrcd = 0
      indfct = -1
C
C ... is a data base assigned?
C ... if a data base is assigned, is a file assigned?
C
      IF(iswtch(13).eq.0) THEN
         ierrcd = 5041
         CALL msgprt(*990,ierrcd)
      ELSE IF(istat(11).eq.0) THEN
         ierrcd = 5051
         GOTO 900
      ENDIF
C
      kfil2h = MOD(kfile(2),64**2)
      IF(kfile(1).eq.0) THEN
C
C ... get default file if no file is specified
C
         ktype = kfil2h/64
         kstype = kfil2h-ktype*64
C
         CALL st5dec(*980,ier0cd,ktype,1,ctype)
C
C ... check internal file types
C
         i0 = 0
         IF(ctype(5:5).eq.´ ´) THEN
            indfct = 1
         ELSE IF(ctype(5:5).eq.´F´) THEN
            indfct = 0
         ELSE
            DO 200 i0=1,6
```

```
                IF(intfil(i0).eq.ctype(5:5)) THEN
                   indfct = induct(i0)
                   GOTO 300
                ENDIF
200          CONTINUE
C
C ... specified file type cannot be found
C
             ierrcd = 5060
             CALL msgprt(*990,ierrcd)
             CALL syserr(*990,'FCTS')
          ENDIF
C
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 immidiate data base access
C
             CALL addts(*980,ier0cd,'D ',kfile)
          ELSE
             kfile(2) = kfile(2)+ktype*64
          ENDIF
          kfile(2) = kfile(2)+kstype
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 = ier0cd
990    RETURN1
       END
```

```
C ... ***  Subroutine for access to IAS-SYSTEM  DATA BASE           ***
C ... ***
C ...
C
C
C ... COMMENT
C
C     This is the unique interface to the data base files
C
C
C ... INPUT BY PARAMETERLIST
C
C     IHIRCD  I        data base access hierarchy code
C     IOPTCD  I        data base access operation code
C
C     The following combinations of IHIRCD and IOPTCD are possible
C
C     IHIRCD=16-31     Access to elements specified by KEY
C                      IOPTCD= 0     Find   element
C                      IOPTCD= 1     Read   element
C                      IOPTCD= 3     Insert element
C                      IOPTCD= 4     Update element
C                      IOPTCD= 6     Print  element
C                      IOPTCD= 7     Delete element
C                      IOPTCD= 8     Copy   element
C                      IOPTCD= 9     Copin  element
C                      IOPTCD=10     Copout element
C
C     IHIRCD=1024      Access to records specified by IADR
C                      IOPTCD=11     Read  record from system-file 11
C                      IOPTCD=12     Read  record from system-file 12
C                      IOPTCD=13     Read  record from system-file 13
C                      IOPTCD=21     Write record to system-file 11
C                      IOPTCD=22     Write record to system-file 12
C                      IOPTCD=23     Write record to system-file 13
C
C     ICUR    I        Currency indicator
C                      0   No record address specified in array iadr(*)
C                      1   Record address specified in array iadr(*)
C     IADR    I(*)     Address array
C     KEY     I(*)     Key of the element
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     ICUR    I        Currency indicator
C                      0   Element not found
C                      1   Element found (record address specified)
C     NSTR    I(*)     String of the numerical information
C     CSTR    C*(*)    String of character information
C
C
C ... RETURN           normal exit
C ... RETURN 1         error exit
C ...
```

```
C
      SUBROUTINE dbacc(*,ierrcd,ihircd,ioptcd,icur,iadr,key,nstr,cstr)
C
      CHARACTER*21 qidedt
      CHARACTER*(*) cstr
      LOGICAL rchang,rfound,lfound
      DIMENSION key(*),iadr(*),nstr(*)
C
      INCLUDE ias-util.comstatus,list
      INCLUDE ias-util.comdbrecs,list
      INCLUDE ias-util.dclcommsg,LIST
      INCLUDE ias-util.comcommsg,LIST
      INCLUDE ias-util.dclcsign,LIST
      INCLUDE ias-util.comcsign,LIST
C
      ierrcd=0
C
      indhir=ihircd
      indopt=ioptcd
C
      IF((indhir.GE.16.AND.indhir.LE.31).AND.indopt.LE.5)THEN
C
         rchang=.false.
C
C ...
C ... Access to elements
C ...
C
         IF(icur.EQ.0)THEN
C
C ... No address is specified in array iadr(*)
C
            CALL dbfind(*990,ier0cd,indhir,indopt,key,irkpos,ilkpos,
     *         ipleaf,ipnstr,ipcstr,rfound,lfound)
C
            iadr(1) = ipleaf
            iadr(2) = ipnstr
            iadr(3) = ipcstr
            iadr(4) = irkpos
            iadr(5) = ilkpos
C
            IF(lfound)THEN
               icur=1
C
C ... Only find specified
C
               IF(indopt.EQ.0)THEN
                  GOTO 900
               ELSE IF(indopt.EQ.3)THEN
                  ierrcd=4951
                  CALL msgprt(*990,ierrcd)
               END IF
            ELSE
               IF(indopt.EQ.0)THEN
                  GOTO 900
```

```
              ELSE IF(indopt.NE.3)THEN
                 ierrcd=4961
                 msgin=qidedt(key,4)//msgstp
                 CALL msgprt(*990,ierrcd)
              END IF
           END IF
        ELSE
           ipleaf = iadr(1)
           ipnstr = iadr(2)
           ipcstr = iadr(3)
           irkpos = iadr(4)
           ilkpos = iadr(5)
        END IF
C
C ... Read element from data base
C
        IF(indopt.EQ.1.OR.indopt.EQ.2)THEN
           CALL dbread(*990,ier0cd,indopt,key,ipnstr,ipcstr,nstr,cstr)
C
C ... Insert element to data base
C
        ELSE IF(indopt.EQ.3)THEN
           CALL dbinp(*990,ier0cd,indopt,key,irkpos,ilkpos,ipleaf,
     *        ipnstr,ipcstr,nstr,cstr,rchang)
C
C ... Update element in the data base
C
        ELSE IF(indopt.EQ.4.OR.indopt.EQ.5)THEN
           CALL dbupd(*990,ier0cd,indopt,key,ipnstr,ipcstr,nstr,cstr)
C
        END IF
C
      ELSE IF((indhir.GE.16.AND.indhir.LE.31).AND.indopt.GT.5)THEN
C
C
C ... Partial key operation for *PRT, *DEL and *COPY
C
C
        ipleaf=iadr(1)
        irkpos=iadr(4)
        ilkpos=iadr(5)
C
        IF(indopt.LT.8)THEN
C
           CALL dblook(*990,ier0cd,indhir,indopt,key,irkpos,ilkpos,
     *        ipleaf,ipnstr,ipcstr,rfound,lfound)
C
           IF(.NOT.lfound)THEN
              ierrcd=-1
              GOTO 900
           END IF
        END IF
C
C ... Print element(s) of a certain kind
C
```

```
          IF(indopt.EQ.6)THEN
             CALL dbread(*990,ier0cd,indopt,key(5),ipnstr,ipcstr,
     *          nstr,cstr)
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,indopt,key(5),irkpos,ilkpos,ipleaf,
     *          ipnstr,ipcstr,rfound,rchang)
C
C ... Copy element(s) of a certain kind
C
          ELSE IF(indopt.GE.8.OR.indopt.LE.10)THEN
C
             CALL dbcopy(*999,ierrcd,indhir,indopt,key,irkpos,ilkpos,
     *          ipleaf,ipnstr,ipcstr,rfound,lfound,rchang)
C
C ... *COPY,I and IAS-EOF encountered ?
C
             IF(indopt.EQ.9.AND.ierrcd.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)=ipleaf
          iadr(4)=irkpos
          iadr(5)=ilkpos
C
       ELSE IF(indhir.EQ.1024)THEN
C
C ...
C ... Access to address-specified records
C ...
C
          IF(indopt.EQ.11)THEN
             CALL read11(*990,ier0cd,indopt,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,cstr)
C
          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
```

```
               CALL writ13(*990,ier0cd,iadr(1),nstr,cstr)
            ELSE
               ier0cd=4520
               CALL msgprt(*990,ier0cd)
               CALL syserr(*990,'DBACC')
            END IF
C
         ELSE
            ier0cd=4510
            CALL msgprt(*990,ier0cd)
            CALL syserr(*990,'DBACC')
         END IF
C
         IF(rchang)THEN
C
C ... Backwriting of the root
C
            CALL writ11(*990,ier0cd,idbst(3),iroot)
            IF(iswtch(4).GT.0) CALL msgprt(*990,4890)
         END IF
900      RETURN
C
990      ierrcd=ier0cd
999      RETURN 1
C
         END
```

```
C ... ***  Subroutine for finding a specified key in data base        ***
C ... ***
C ...
C
C
C ... INPUT BY PARAMETERLIST
C
C     KEY      I(*)      Specified key
C
C
C ... INPUT BY COMMON/STATUS/
C
C     IDBST    I(40)     IDBST(3)   Pointer to root of B*-tree
C                        IDBST(4)   Pointer to first leaf in B*-tree
C                        IDBST(6)   Maximal number of keys in root
C                        IDBST(7)   Maximal number of keys in leaf
C                        IDBST(21)  Pointer to next free record of file 11
C
C
C ... OUTPUT BY PARAMETERLIST
C
C
C     IRKPOS  I          Position of the according pointer in root
C     ILKPOS  I          Position of the according pointer in leaf
C     IPLEAF  I          Pointer  to leaf
C     IPNSTR  I          Pointer  to numerical string
C     IPCSTR  I          Pointer  to character string
C     RFOUND  L          true if key is stored in root
C     FOUND   L          true if key is found in data base
C
C
C ... RETURN             normal exit
C ... RETURN 1           error exit
C ...
C
      SUBROUTINE dbfind(*,ierrcd,indhir,indopt,key,irkpos,ilkpos,
     *   ipleaf,ipnstr,ipcstr,rfound,found)
C
      LOGICAL   found,rfound,root
      DIMENSION key(*)
C
      INCLUDE ias-util.comstatus,LIST
      INCLUDE ias-util.comdbrecs,LIST
C
      ierrcd=0
      IF(indhir.NE.31)THEN
         ier0cd=4510
         CALL msgprt(*990,ier0cd)
         CALL syserr(*990,'DBFIND')
      END IF
C
C ... Initialization of leaf-pointer
C
      ipleaf=0
C
```

```
      IF((idbst(21).EQ.idbst(4).AND.iroot(1).LE.idbst(7)).OR.
     *    (iroot(1).LE.idbst(6)))THEN
C
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
C ... Only the root-record contains pointers
C
             IF(rfound)THEN
                found=.true.
                CALL ptrdec(*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
          ier0cd=4530
          CALL msgprt(*990,ier0cd)
          CALL syserr(*990,'DBFIND')
       END IF
C
C ... Read the specified leaf
C
       CALL read11(*990,ier0cd,indopt,ipleaf,ileaf)
C
       IF(ileaf(1).le.idbst(7))THEN
C
C ... Search for key in leaf
C
          root=.false.
          CALL keychk(*990,ileaf,root,key,indhir,ilkpos,ipoint,found)
C
          IF(found)THEN
             CALL ptrdec(*990,ipoint,ipnstr,ipcstr)
          ELSE
             ipnstr=idbst(31)
             ipcstr=idbst(36)
          END IF
       ELSE
          ier0cd=4540
          CALL msgprt(*990,ier0cd)
          CALL syserr(*990,'DBFIND')
       END IF
900    RETURN
C
```

```
990     ierrcd=ier0cd
999     RETURN 1
C
        END
```

```
C ... *** Subroutine for checking whether a specified key           ***
C ... *** is in a specified record of file 11                       ***
C ... ***
C ...
C
C ... INPUT BY PARAMETERLIST
C
C ... ISTRNG  I        Record-string to be searched
C ... ROOT    L        is true if key is searched in root
C ... KEY     I        Key of the element
C ... NRKEY   I        Maximal number of keys in ISTRNG
C
C ... INPUT BY COMMON/STATUS/
C
C     IDBST   I(40)    IDBST(4)   Pointer to first leaf in B*-tree
C                      IDBST(21)  Pointer to next free record of file 11
C
C
C ... OUTPUT BY PARAMETERLIST
C
C ... KPOS    I        Start position of key or right key
C ... IPNTR   I        Pointer to the next record in B*-tree
C ... FOUND   L        is true if the specified key is in the record
C
C
C ... RETURN           Normal exit
C ... RETURN 1         Error exit
C ...
C
      SUBROUTINE keychk(*,istrng,root,key,indhir,kpos,ipntr,found)
C
      LOGICAL found,root
      DIMENSION key(4),istrng(*)
      INCLUDE ias-util.comstatus,LIST
C
      found=.false.
      ianf=10
      iend=5*istrng(1)+6
C
      DO 160 i=ianf,iend,5
C
         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 veccop(istrng,i+2,i+2,key,2)
            END IF
            GOTO 900
         END IF
160   CONTINUE
C
```

```
C ... The searched key is the largest in the specified record
C
      kpos=iend+4
      IF(root)THEN
         IF(idbst(21).EQ.idbst(4))THEN
            ipntr=0
         ELSE
            ipntr=istrng(kpos)
         END IF
      ELSE
         ipntr=0
      END IF
      found=.false.
900   RETURN
C
990   RETURN 1
C
      END
```

```
C ... ***  Subroutine for comparing partial keys in data base        ***
C ... ***
C ...
C
C
C ... COMMENT
C
C ... The bits of indhir set reflect the following comparisons
C
C ...    Bit 5: Filename
C ...    Bit 4: Type
C ...    Bit 3: Subtype
C ...    Bit 2: Element
C ...    Bit 1: Version
C
C
C ... INPUT BY PARAMETERLIST
C
C     KEY     I(*)     Specified key
C     INDHIR  I        Data base access hierarchy code
C     ISTRNG  I(*)     ROOT or LEAF
C     ISTART           Start position for Compare-Operation
C
C
C
C ... OUTPUT BY PARAMETERLIST
C
C
C     KSWTCH  I        Switch indicating the relation between
C                      partial key and key of istrng
C
C
C ... RETURN           normal exit
C ... RETURN 1         error exit
C ...
C
       SUBROUTINE keycom(key,indhir,istrng,istart,kswtch)
C
       DIMENSION key(*),istrng(*)
C
       indacc=MOD(indhir,16)
C
C ... For *COPY,I file name may not be specified, e.g. *COPY,I  ,X.
C
       IF(key(1).EQ.0)GOTO 100
C
C ... Test for Filename
C
       IF(key(1).LT.istrng(istart+1))GOTO 200
       IF(key(1).GT.istrng(istart+1))GOTO 400
C
       kchg=(key(2)/64**2)*64**2
       ichg=(istrng(istart+2)/64**2)*64**2
       IF(kchg.LT.ichg)GOTO 200
       IF(kchg.GT.ichg)GOTO 400
```

```
C
C ... Test for Type ?
C
100    IF((indacc/8).EQ.1)THEN
          kchg=((key(2)-kchg)/64)*64
          ichg=((istrng(istart+2)-ichg)/64)*64
          IF(kchg.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
C
          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
       kswtch=0
       GOTO 900
200    kswtch=-1
       GOTO 900
400    kswtch=+1
900    RETURN
       END
```

```
C ... ***  Subroutine for reading an element specified by key        ***
C ... ***
C ...
C
C
C ... INPUT BY PARAMETERLIST
C
C     INDOPT  I        Data base access operation code
C     KEY     I(4)     Specified key
C     IPNSTR  I        Pointer to numerical string
C     IPCSTR  I        Pointer to character string
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     NUMSTR  I(*)     Numerical string to be read
C     CSTR    C(*)     Character string to be read
C
C
C ... RETURN           normal exit
C ... RETURN 1         error exit
C ...
C
      SUBROUTINE dbread(*,ierrcd,indopt,key,ipnstr,ipcstr,numstr,cstr)
C
      CHARACTER*(*) cstr
      DIMENSION key(*),numstr(*)
      INCLUDE ias-util.comstatus,LIST
C
      ierrcd=0
C
C ... Initialize length of numerical string and character string
C
      numstr(1)=2
      numstr(2)=1
C
      IF(indopt.LE.2.OR.indopt.GE.6)THEN
C
C ... Read numerical-string
C
         CALL rdnstr(*990,ier0cd,key,ipnstr,numstr)
         IF(iswtch(4).GT.0) CALL msgprt(*990,4710)
C
         IF(indopt.ne.2)THEN
C
C ... Read character-string
C
            CALL rdcstr(*990,ier0cd,key,ipcstr,numstr,cstr)
            IF(iswtch(4).GT.0) CALL msgprt(*990,4700)
         END IF
C
      ELSE
         ier0cd=4520
         CALL msgprt(*990,ier0cd)
         CALL syserr(*990,'DBREAD')
```

```
      END IF
      RETURN
990   ierrcd=ier0cd
      RETURN 1
C
      END
```

```
C ... ***  Subroutine for reading the numerical string of file 12    ***
C ... ***
C ...
C
C
C ... INPUT BY PARAMETERLIST
C
C     KEY     I(4)      Key of the element
C     IPNSTR  I         Pointer to the next free record in file 12
C
C
C ... OUTPUT TO PARAMETERLIST
C
C     NUMSTR  I(*)      Numerical string
C
C
C ... INPUT BY COMMON/STATUS/
C
C     IDBST   I(40)     IDBST(12)  Record length of file 12
C
C
C ... RETURN            normal exit
C ... RETURN 1          error exit
C ...
C
       SUBROUTINE rdnstr(*,ierrcd,key,ipnstr,numstr)
C
       LOGICAL jkequ
       DIMENSION key(*),numstr(*)
       INCLUDE ias-util.comstatus,LIST
       COMMON/idummy/istack(510),ibuff(108),infchr(20),infdum(20)
C
       ierrcd=0
C
       icon=0
       ibeg=2
C
200    CALL vecnul(ibuff,1,idbst(12))
       CALL read12(*990,ier0cd,ipnstr,ibuff)
C
C ... Testing for right key
C
       IF(jkequ(key,ibuff,4))THEN
C
C ... Create initial information from the first record
C
          IF(icon.EQ.0)THEN
             listr=ibuff(9)
             ires =listr
             ircl =idbst(12)-8
          END IF
C
          ilen=MIN(ircl,ires)
          iend=ilen+8
          CALL veccop(ibuff,9,iend,numstr,ibeg)
```

```
C
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
          ierrcd=4570
          CALL msgprt(*990,ierrcd)
          CALL syserr(*990,'RDNSTR')
       END IF
C
990    ierrcd=ier0cd
       RETURN 1
C
       END
```

```
C ... ***  Program for reading one record of file 12                    ***
C ... ***
C ...
C
C
C ... INPUT BY PARAMETERLIST
C
C     IREC    I        Record number to be read
C
C
C ... INPUT BY COMMON/STATUS/
C
C     IDBST   I(40)    IDBST(12)  Record length of file 12
C                      IDBST(13)  Number of records of file 12
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     IERRCD  I        Error code
C     ISTR    I        Contents of the specified record
C
C
C ... RETURN           Normal exit
C ... RETURN 1         Error exit
C ...
C
       SUBROUTINE read12(*,ierrcd,irec,istr)
C
       DIMENSION istr(*)
       INCLUDE ias-util.comstatus,LIST
C
       ierrcd=0
       lstrng=idbst(12)
C
       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,'READ12')
       END IF
       RETURN
C
900    ierrcd=ier0cd
990    RETURN 1
       END
```

```
C ... ***  Subroutine for aggregation of time series                    ***
C ... ***
C
C
C ... COMMENT
C
C     This routine aggregates time series, if necessary. Four different
C     aggregation procedures are available:
C     Code <0 ... aggregation mode is not defined
C     Code  0 ... aggregate by summing up
C     Code  1 ... aggregate by averaging
C     Code  2 ... take last value of time range as aggregate (for stocks)
C     Code  3 ... deflator aggregation (e.g. deflator defined as
C                 (NOMINAL SERIES)/(REAL SERIES)*100, aggregation is
C                 done by aggregating both series according to their
C                 aggregation mode - accept for another aggregation
C                 mode 3 because of recursion - and repeating the
C                 calculation mentioned above.
C                 This aggregation is only possible for time series
C                 read from the data base. Evidently, also the
C                 nominal and the real series must be items of the
C                 data base and the needed IAS-files must be assigned.
C                 For this reason deflator aggregation is not possible
C                 for intrinsic variables series.
C
C     Disaggregation of time series is not possible and therefore
C     inhibited.
C
C
C
C ... INPUT BY PARAMETERLIST
C
C     INDDB   L        Indicator for time series origin
C                      .T. ... series is stored in data base
C                      .F. ... series is intrinsic variables
C     KIDNTF  I(4)     INDDB = .T. ... integer-5-coded item identifier
C                      INDDB = .F. ... (1)-(2) ... 0
C                                      (3) ....... index of intrinsic
C                                                  variable in intvtb
C                                      (4) ....... 0
C     ITMSER  I(3)     Time range for which the time series is defined
C     MODAGG  I        Aggragation mode (see above in COMMENT)
C     KAGGID  I        Contains the identifier of the nominal and the
C                      real time series for deflator aggregation
C                      (1)-(4) ... integer-5-coded identifier of
C                                  the nominal time series
C                      (5)-(8) ... integer-5-coded identifier of
C                                  the real time series
C     RINPVC  I(*)     Data input vector
C
C
C ... INPUT BY COMMON/calc1/
C
C     INTVTB  I(NUMINT) Table of intrinsic variables
C
```

```
C
C ... INPUT BY COMMON/comstatus/
C
C     ISWTCH  I(20)     (4) ... Debug switch
C                       (9) ... Batch mode switch
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     AGGVEC  I(*)      Aggregated time series
C     LAGGVC  I         Length of aggregated time series
C     IERRCD  I         Error code
C
C
C ... OUTPUT BY COMMON/miss/
C
C     MISSB   I         Number of missing values at the begin
C                       of the aggregated time series
C     MISSE   I         Number of missing values at the end
C                       of the aggregated time series
C
C
C ... TRANSPUT BY PARAMETERLIST
C
C     ITMDEF  I         Time definition
C                       I: time range requested
C                       O: time range possible, when combining
C                          requested time range and time range
C                          of the time series
C
C
C
C ... RETURN            Normal exit
C ... RETURN1           Error exit
C ...
C
      SUBROUTINE aggreg(*,ierrcd,inddb,kidntf,itmser,modagg,kaggid
     *    ,rinpvc,aggvec,laggvc,itmdef)
C
      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.dclcommsg,LIST
      INCLUDE ias-util.comcommsg,LIST
      INCLUDE ias-util.dclcsign,LIST
      INCLUDE ias-util.comcsign,LIST
      INCLUDE ias-util.commiss,LIST
      INCLUDE ias-util.dclcalc1,LIST
      INCLUDE ias-util.comcalc1,LIST
      ierrcd = 0
      logadr = ABS(kidntf(3))
C
```

```
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
C ... evaluate common time range
C
      CALL veccop(itmdef,1,3,itmcom,1)
      CALL comtm(itmser,itmcom)
C
C ... DEBUG of time ranges
C
      IF(iswtch(4).ne.0) THEN
         CALL msgmul(*990,7410,7430)
         WRITE(msgin,FMT='(I2,2(1X,I10),2(2X,I2,2(1X,I10)),A1)')
     *      itmdef,itmser,itmcom,msgstp
         CALL msgprt(*990,0)
      ENDIF
C
      nditem = itmcom(3)-itmcom(2)+1
      IF(nditem.gt.laggvc) THEN
C
C ... output vector AGGVEC can't hold all values
C
         ierrcd = 6601
         CALL msgprt(*990,ierrcd)
      ENDIF
C
C ... evaluate aggregation ratio, missing start and end values,
C ... begin and end of desired data string in input vector RINPVC
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
      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
C
      CALL veccop(itmcom,1,3,itmdef,1)
C
      IF(nquot.eq.1) THEN
C
C ... no aggregation necessary, time series is copied
C
         CALL vecprr(rinpvc,idatab,idatae,aggvec,1)
      ELSE
C
         IF(modagg.ge.0.and.modagg.lt.3) THEN
C
C ... aggregation mode is sum, average or stock
C
            CALL aggsas(modagg,idatab,idatae,nquot,rinpvc,aggvec)
         ELSE IF(modagg.eq.3) THEN
C
C ... aggregation mode is deflator
C
            IF(.not.inddb) THEN
               msgin = intvtb(logadr)//msgstp
               ierrcd = 7321
               CALL msgprt(*990,ierrcd)
            ELSE IF(kaggid(3).eq.0.or.kaggid(7).eq.0) THEN
               ierrcd = 7340
               CALL msgprt(*990,ierrcd)
            ELSE
               CALL aggdef(ierrcd,kaggid,aggvec,itmcom)
            ENDIF
            IF(ierrcd.gt.0) THEN
               msgin = qidedt(kidntf,4)//msgstp
               CALL msgprt(*990,7351)
            ENDIF
         ELSE
C
C ... aggregation mode is not defined or not valid
C
            IF(modagg.lt.0) THEN
               ierrcd = 7391
               ipos = 1
            ELSE
               ierrcd = 7030
               WRITE(msgin,FMT='(I1)') modagg
               ipos = 2
            ENDIF
            IF(inddb) THEN
               msgin(ipos:ipos+20) = qidedt(kidntf,4)
            ELSE
               msgin(ipos:ipos+20) = intvtb(logadr)
            ENDIF
            ipos = ipos+21
            msgin(ipos:ipos) = msgstp
            CALL msgprt(*990,ierrcd)
```

```
            IF(modagg.ge.0) CALL syserr(*990,'AGGREG')
         ENDIF
      ENDIF
C
C ... DEBUG of original and aggregated time series
C
      IF(iswtch(4).ne.0) THEN
         CALL msgmul(*990,7430,7450)
         ncomma = 3
         IF(iswtch(9).eq.0) THEN
            nvalrw = 5
         ELSE
            nvalrw = 10
         ENDIF
         CALL rvcprt(*990,rinpvc,idatab,nditem,nvalrw,ncomma)
         CALL msgprt(*990,10)
         CALL rvcprt(*990,aggvec,1,laggvc,nvalrw,ncomma)
         CALL msgprt(*990,10)
      ENDIF
C
      RETURN
C
990   RETURN1
      END
```

```
C ... ***  Subroutine for printing a time series on standard output   ***
C ... ***  unit with 76 print positions per line                       ***
C ... ***
C ...
C
C
C ... INPUT BY PARAMETERLIST
C
C     IOPTCD  I         Option code
C                       -1 ..... only header output requested
C                       ELSE ... also data is to be printed
C     INDPRT  I         Indicator for print-out size
C                       <1 ..... no header output
C                       >4 ..... print full item information
C     DUMMY   L         Indicates if item is a dummy
C     ITMDEF  I(3)      Time definition for which the series
C                       will be printed
C     KIDNTF  I(4)      Integer-5-coded item identifier
C     DATSTR  R(*)      Vector containing the desired (aggregated)
C                       data values to be printed
C     HEADER  C*76      Header description
C
C
C ... INPUT BY PARAMETER STATEMENT COMPOUND/pardata/
C
C     LNMSTR  I         Dimension of NUMSTR
C
C
C ... INPUT BY PARAMETER STATEMENT COMPOUND/parxdb/
C
C     LKXDB   I         Length of an integer-4-coded item identifier
C                       of an external data base
C
C
C ... INPUT BY COMMON/miss/
C
C     MISSB   I         Number of missing values at the beginning
C     MISSE   I         Number of missing values at the end of the series
C
C
C ... INPUT BY COMMON/dbninf/
C
C     NUMSTR  I(LNMSTR) Numerical string containing the full
C                       information and the unchanged data of an item
C                      (for further documentation see subroutine SERWRT)
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     IERRCD  I         Error code
C
C
C
C ... RETURN            Normal exit
C ... RETURN1           Error exit
```

```
C ...
C
      SUBROUTINE sp076(*,ierrcd,ioptcd,indprt,dummy,itmdef,kidntf
     *    ,datstr,header)
C
      CHARACTER*76 header
      CHARACTER*21 qidedt
      CHARACTER*14 qdtdec
      CHARACTER*10 ttxt,sttxt
      CHARACTER*7 qtmdec
      CHARACTER*5 aggmod(0:3)
      LOGICAL dummy
      DIMENSION itmdef(3),datstr(*)
C
      INCLUDE ias-util.dclcommsg,LIST
      INCLUDE ias-util.comcommsg,LIST
      INCLUDE ias-util.dclcsign,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.dclorigin,LIST
      INCLUDE ias-util.comorigin,LIST
C
      DATA aggmod /'sum','aver.','stock','defl.'/
C
      ierrcd = 0
C
      IF(indprt.gt.0) THEN
C
C ... complete and print header line
C
         CALL tstedt(*980,ier0cd,'DS',ttxt,sttxt,indst)
         msgin = sttxt//qidedt(kidntf,4)//qdtdec(numstr(8))
     *      //msgstp
         CALL yupcon(*980,ier0cd,msgin(1:1))
         CALL msgprt(*990,7510)
         msgin = header//msgstp
         CALL msgprt(*990,0)
      ENDIF
C
      IF(indprt.gt.4) THEN
C
C ... L-option is specified
C
         CALL msgprt(*990,10)
C
C ... print time range of values in series and date/time last
C ... header update
C
         IF(dummy) THEN
            msgin = 'dummy  item'
         ELSE
            msgin = qtmdec(numstr(12),numstr(13))//qtmdec(numstr(12)
```

```
     *          ,numstr(14))
        ENDIF
        msgin(15:30) = qdtdec(numstr(numstr(2)+6))//msgstp
        CALL msgprt(*990,7530)
C
C ... print time range of prognosted values in series and date/time
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 msgprt(*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
        ELSE
           CALL syserr(*990,'SP076')
        ENDIF
        WRITE(msgin,FMT='(I1,A5,A21,A1)')
     *     numstr(20),aggmod(numstr(20)),origin(ind1),msgstp
        CALL msgprt(*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 msgprt(*990,7560)
        ELSE
           ind0 = 21
        ENDIF
C
C ... print in case of generation type 'interface' the name of the
C ... data item in the external data base
C
        IF(numstr(11).ge.90) THEN
           ind1 = lkxdb*4
           msgin = ' '
           CALL st4dec(*980,ier0cd,numstr(ind0),lkxdb,msgin(1:ind1))
           msgin(65:65) = msgstp
           CALL msgprt(*990,7570)
           ind0 = ind0+lkxdb
```

```
          ENDIF
C
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
C
       IF(ioptcd.ne.-1) THEN
          CALL msgprt(*990,10)
          IF(dummy) THEN
             msgin = qidedt(kidntf,4)//msgstp
             CALL msgprt(*990,7650)
          ELSE
C
C ... print of data
C
             msgnum = 7610
             CALL spdata(*980,ier0cd,msgnum,itmdef,datstr)
          ENDIF
       ENDIF
C
       RETURN
C
C ... error section
C
980    ierrcd = ier0cd
990    RETURN1
       END
```

THE SEMIPORTABLE ROUTINES WITHIN THE IAS-SYSTEM

YBLKDT
YCLOSE
YDATIM
YFILE
YINQU
YIOERR
YOPEN
YQMACD
YSET
ZCSFPT

```
C ... ***  BLOCKDATA PROGRAM OF THE IAS-SYSTEM VERSION UNIVAC          ***
C ... ***
C ...
C
C

      BLOCK DATA yblkdt
C
C
C
      INCLUDE ias-util.dclcsign,LIST
      CHARACTER*2 iaslev
      CHARACTER*1 star,blank,comma,apost,sfsep,mast,pound,lbrak
     *    ,rbrak,msgvar,msgstp,msgcon,lsg1,lsg2,lsg3,slash,pointc
     *    ,syssgn,msschr
C
      INCLUDE ias-util.comcsign,LIST
C
C ... COMMON CSIGN contains special characters
C
C     STAR.............sign indicating an IAS-command
C     BLANK............seperator between option(command) and data string
C     COMMA............seperator between command and option and
C                      between two fields in input string
C     APOST............sign to be used for marking a comment
C     SFSEP............seperator between two subfields
C     MAST.............sign starting an system command
C     POUND............sign starting a data input
C     LBRAK............left pointed braket
C     RBRAK............right pointed braket
C     MSGVAR...........sign to be replaced by message
C     MSGSTP...........sign indicating the end of a message
C     MSGCON...........sign indicating continuation of a message
C     LSG1...LSG3......signs additionally allowed to be used in a name
C     SLASH............sign used as division symbol
C     POINTC...........sign to mark a special position in the line above
C     SYSSGN ..........sign starting a SYSTEM-command
C     IASLEV           release level of IAS-SYSTEM  (C*2)
C     CHRL.............latest character read in calcstring
C     MSSCHR...........character to indicate a missing value
C
      COMMON /csign/ star,blank,comma,apost,sfsep,mast,pound,lbrak
     *    ,rbrak,msgvar,msgstp,msgcon,lsg1,lsg2,lsg3,slash,pointc
     *    ,syssgn,iaslev,msschr
      DATA pound,lbrak,rbrak /'#','(',')'/
      DATA msgvar,msgstp,msgcon /'^','~','&'/
      DATA lsg1,lsg2,lsg3,slash,pointc,syssgn/'$','%','&','/','^','$'/
      DATA iaslev,chrl,msschr/'1 ',' ','.'/
C
C
C
      INCLUDE ias-util.comiordc,LIST
C
C ... COMMON IORDC containing computer depending values
C
```

```
C      IAC..............ordinal number of upper code A
C      IIC..............ordinal number of upper code I
C      IJC..............ordinal number of upper code J
C      IRC..............ordinal number of upper code R
C      ISC..............ordinal number of upper code S
C      IZC..............ordinal number of upper code Z
C      I9C..............integer number representing character 9
C      I0C..............integer number representing character 0
C      IDIFC............difference of lower code a to upper code A
C      IBLC.............ordinal number of character blank
C
       COMMON /iordc/ iac,iic,ijc,irc,isc,izc,i0c,i9c,idifc,iblc
       DATA iac/65/,iic/73/,ijc/74/,irc/82/,isc/83/,izc/90/
       DATA i0c/48/,i9c/57/,idifc/-32/,iblc/32/
C
C
C
       INCLUDE ias-util.dclcomand,LIST
       CHARACTER*80 striin,spcstr
       CHARACTER*6 option
       CHARACTER*4 comand
C
       INCLUDE ias-util.comcomand,LIST
C
C ... COMMON COMAND includes the whole input string as well as
C      seperated command/option and the packed specification string
C
C      STRIIN...........whole input string
C      COMAND...........non-encoded command
C      OPTION...........non-encoded option
C      SPCSTR...........packed specification string
C
       COMMON /comand/ striin,comand,option,spcstr
C
C
C
       INCLUDE ias-util.dclcmdtab,LIST
       CHARACTER*4 cmdtbl
C
       INCLUDE ias-util.comcmdtab,LIST
C
C ... COMMON CMDTAB includes all available IAS-commands
C
C      CMDTBL...........table of commands - length in parameter ICTL
C
       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','PRT','W','WAIT',' ',' ',' ',' ',' ',' ',
     * ' ',' ',' ',' ',' ',' ',' ',' ','D','DATA',
     * 'OLS','O','IV','TSLS','KLS','LIML','2SLS','3SLS','LIVE','FIVE',
     * 'FIML','TEST',' ',' ',' ',' ',' ',' ',' ',' ',
     * 'DEL','EDIT','COPY','INFO','LINK','PLT','PCH','TAPE','MAT','LP',
     * 'DB','EXIT',' ',' ',' ',' ',' ',' ',' ',' ',
```

```
     * ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
     * ' ',' ','CHG','DUMP',' ',' ',' ',' ',' ',' '/
C
C
C
      INCLUDE ias-util.comstrcd,LIST
C
C ... COMMON STRCD includes encoded command/option and
C ... vectors of pointers to fields/subfields in string SPCSTR
C
C     ICMDNR...........encoded command
C     IOPTNR(2)........encoded option containing character and
C                      numerical options
C     IFLDVC(10).......vector pointing to the last character in a field
C     ISFDVC(40).......vector pointing to the last character in a subfield
C
      COMMON /strcd/ icmdnr,ioptnr(2),ifldvc(10),isfdvc(40)
C
C
C
      INCLUDE ias-util.comstatus,LIST
C
C     ISTAT(1).........read status
C     ISTAT(2).........iruntp
C     ISTAT(3).........numcmd
C     ISTAT(4).........numinp
C     ISTAT(5).........isups
C     ISTAT(6).........numio
C     ISTAT(10)........last input line not yet checked if .gt.0
C     ISTAT(11)........number of files in file control table FCT
C     ISTAT(14)........continuation record was read at last READ from
C                      MSG-file if ISTAT(15) > 0
C     ISTAT(17)........continuation of algebraic string encountered
C     ISTAT(18)........record number of EXWS-File record in buffer
C     ISTAT(19)........number of changes ocurred to the record of
C                      EXWS-File actually in buffer
C
C
C     ISWTCH(1)........input log on (=1) or off (=0)
C     ISWTCH(2)........echo mode on (=1) or off (=0)
C     ISWTCH(3)........walk in case of system errors. ON/OFF (see above)
C     ISWTCH(4)........debug mode ON/OFF
C     ISWTCH(5)........output also to alternate print file. 0,1,2
C                      0 = no output log
C                      1 = all output to output log
C                      2 = as above, additionally all input (like echo
C                          mode) to output log
C     ISWTCH(6)........print error message if output is truncated.ON/OFF
C     ISWTCH(7)........expert mode. ON/OFF
C     ISWTCH(8)........quantity of print output. 0,1,2,3,4,5,6
C     ISWTCH(9)........batch mode. ON/OFF
C     ISWTCH(10).......assign total set of files if ISWTCH(10).EQ.1
C                      assign only minimal number of files  if
C                      ISWTCH(10).EQ.0
C     ISWTCH(11).......switch for writing results to the mass storage
```

```
C                             file EXWS (extended working storage) for re-use.
C                             If 'ON' all records are written to file EXWS,
C                             if 'OFF' records <= ISIZE(6) are not written.
C     ISWTCH(13).......DB-assign. 0,1,2,3
C                             0 = No data base assigned
C                             1 = Data base assigned read-only
C                             2 = Data base assigned write-only
C                             3 = Data base assigned read/write enabled
C     ISWTCH(14).......EXWS assigned. ON/OFF
C                             0 = File EXWS not assigned
C                             1 = File EXWS assigned
C     ISWTCH(15).......internal OS-file assigned. ON/OFF
C     ISWTCH(16).......message file assigned. ON/OFF
C     ISWTCH(17).......input-log assigned. ON/OFF
C     ISWTCH(18).......output-log assigned. ON/OFF
C
C
C     ISIZE(4).........record length of file EXWS (extended working
C                             storage on mass storage)
C     ISIZE(5).........number of records in file EXWS
C     ISIZE(6).........number of records in file EXWS which are
C                             sensitive to ISWTCH(11)
C     ISIZE(11)........pointer indicating type of computer
C     ISIZE(12)........number of records in MSG-file
C     ISIZE(13)........maximum number of bits which can be used for
C                             a positive integer, i.e. 2**ISIZE(13)-1 is
C                             the largest integer for this implementation
C     ISIZE(14)........number of lines per page
C     ISIZE(15)........number of characters per print line for
C                             interactive mode, not counting the vertical
C                             spacing character
C     ISIZE(16)........number of characters per print line for
C                             batch mode, not counting the vertical spacing
C                             character
C     ISIZE(17)........number of bits per computer word
C                             (see also ISIZE(13))
C     ISIZE(18)........character length in bits
C     ISIZE(19)........number of significant bits per character
C     ISIZE(20)........number of characters per word
C
C
C     IDBST(1)......... record length of file 11
C     IDBST(2)......... number of records of file 11
C     IDBST(3)......... pointer to root record of tree in file 11
C     IDBST(4)......... pointer to first leaf in B*-tree
C     IDBST(6)......... max. number of keys in root
C     IDBST(7)......... max. number of keys in leaf
C     IDBST(8)......... middle key in leaf
C     IDBST(9)......... position of middle key in leaf
C     IDBST(11)........ factor for computing size of file 12 and 13
C     IDBST(12)........ record length of file 12
C     IDBST(13)........ number of records of file 12
C     IDBST(16)........ record length of file 13
C     IDBST(17)........ number record in file 13
C     IDBST(18)........ length of integer info. of one record
```

```
C     IDBST(19)........ length of character info. of one record
C                       (in character storage units)
C     IDBST(21)........ pointer to next free record of tree in file 11
C     IDBST(22)........ number of keys stored in file 11 (B*-tree)
C     IDBST(23)........ number of records used in file 11 for B*-tree
C     IDBST(26)........ number of catalogued IAS-files in file 11
C     IDBST(31)........ pointer to next free record of file 12
C     IDBST(32)........ number of elements stored in file 12
C     IDBST(33)........ number of records used in file 12
C     IDBST(36)........ pointer to next free record of file 13
C     IDBST(37)........ number of elements stored in file 13
C     IDBST(38)........ number of records used in file 13
C
       COMMON /status/ istat(20),iswtch(20),isize(20),idbst(40)
     *    ,input,iout,kexws,kosint,msguni,loginp,logout
C
C
C
       INCLUDE ias-util.parrecl,LIST
       PARAMETER (lrec11=510,lrec12=108,lrci13=20,lrcc13=120,lrec14=500)
C
       INCLUDE ias-util.comdbrecs,LIST
C
C ... contains data base records of file 12 and file 13
C
C     IROOT(1).......number of keys in data base root
C          (2).......free record concatenation pointer
C          (3).......not used
C          (4).......not used
C          (5)......./
C          (6)......./
C          (7)......./ for further statistical use
C          (8)......./
C          (9)......./
C          (10)......1st pointer (less than 1st key)
C          (11)....../
C          (12)....../
C          (13)....../1st key
C          (14)....../
C          (15)......2nd pointer (greater than 1st key)
C          (16)....../
C          (17)....../
C          (18)....../2nd key
C          (19)....../
C          (20)......3rd pointer (greater than 2nd key)
C             .
C             .
C             .
C
C     ILEAF(1).......number of keys in data base leaf
C          (2).......free record concatenation pointer
C          (3).......pointer to left leaf
C          (4).......pointer to right leaf
C          (5)......./
C          (6)......./
```

```
C          (7)......./ for further statistical use
C          (8)......./
C          (9)......./
C          (10)......1st pointer (less than 1st key)
C          (11)....../
C          (12)....../
C          (13)....../1st key
C          (14)....../
C          (15)......2nd pointer (greater than 1st key)
C          (16)....../
C          (17)....../
C          (18)....../2nd key
C          (19)....../
C          (20)......3rd pointer (greater than 2nd key)
C            .
C            .
C            .
C
      COMMON /dbrecs/ iroot(lrec11),ileaf(lrec11)
C
C
C
      INCLUDE ias-util.comintctt,LIST
C
C     INDUCT.......use control table, contains pointer to file in FCT
C           (1)....base file
C           (2)....data file
C           (3)....equation file
C           (4)....model file
C           (5)....solution file
C           (6)....text file
C
C     KFCT.........keyed names of assigned IAS-files
C
C     IRWPRM.......read/write permission indicator, corresponding
C                  to files in KFCT
C                  =-1...array element not in use
C                  = 0...file assigned read enabled, write protected
C                  = 1...file assigned write enabled, read protected
C                  = 2...file assigned read/write enabled
C
      COMMON /intctt/ kfct(2,0:10),irwprm(0:10),induct(10),itct(3,40)
      DATA induct,kfct,irwprm /10*1,22*0,11*-1/
C
C
C
      INCLUDE ias-util.comsys,LIST
C
C ... COMMON SYS contains tolerance values for calculation
C
C     tol(1) ... EXP(tol(1)) is greatest possible value for single
C                precision
C     tol(2) ... allowed declination for a value to be treated as
C                integer
C     tol(3) ... greatest possible value for single precision
```

```
C
C     tol(4) ... greatest possible value for trigonometric functions
C
      COMMON /sys/ tol(10)
      DATA tol /80.0,.1E-20,1.0E32,1.0E6,6*0.0/
C
C
C
      INCLUDE ias-util.parcalc,LIST
      PARAMETER (maxcal=12,ninvar=7,ninftn=8,maxscl=400,ipoll=800,
     *   maxlbl=800,maxfl=10,maxrft=30,maxval=2000,maxscv=2400,
     *   maxpar=20,maxkey=4,lclc=500)
C
      INCLUDE ias-util.dclcalc1,LIST
      CHARACTER*8 intvtb,infnct
      CHARACTER*5 keytbl
      CHARACTER*(lclc) clcchr
      CHARACTER*1 chrl
C
      INCLUDE ias-util.comcalc1,LIST
      COMMON /calc1/ intvtb(ninvar),infnct(ninftn),keytbl(maxkey),
     *   clcchr,chrl
      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'/
C
C
C
      INCLUDE ias-util.comfnpar,LIST
C
C ... COMMON FNPAR contains the number of parameters for
C                 intrinsic functions
C
      COMMON /fnpar/ ninpar(ninftn)
      DATA ninpar/1,1,1,1,1,1,2,2/
C
C
C
      INCLUDE ias-util.dcltpstp,LIST
      PARAMETER (ntyp=5,nsytyp=1,nstyp=20)
      CHARACTER*1 tchr(ntyp),stchr(nstyp)
      CHARACTER*10 ttxt(ntyp),sttxt(nstyp)
C
      INCLUDE ias-util.comtpstp,LIST
C
C ... COMMON TPSTP allows the conversion of the type-subtype-code
C ... to text
C
C     TCHR ..... type-characters (4 type-characters +
C                                 1 system type-character)
C     STCHR .... subtype-characters
C     TTXT ..... text for the single types (e.g. 'DATA' for 'D')
C     STTXT .... text for the single subtypes (e.g. 'SERIES' for 'S')
C     IPTTST ... pointer to the first subtype-character
C                of a specific type (TCHR) within STCHR
```

```
C
      COMMON /itpstp/ ipttst(ntyp)
      COMMON /ctpstp/ tchr,ttxt,stchr,sttxt
      DATA ipttst /1,4,8,15,19/
      DATA tchr /'X','M','E','D','F'/
      DATA stchr /3*' ',4*' ',7*' ','V','S','M',' ',' ',' '/
      DATA ttxt /'text','model','equation','data','$sys'/
      DATA sttxt /3*' ',4*' ',7*' '
     *    ,'value','series','matrix',' ','function',' '/
C
C
C
      INCLUDE ias-util.dclorigin,LIST
      CHARACTER*21 origin(0:16)

      INCLUDE ias-util.comorigin,LIST
C
C ... COMMON ORIGIN contains the text for the different origin types
C
      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',3*' ','estim.endogenous var.'
     *    ,'estimated residuals',2*' '
     *    ,'interface to ext. db','interf.ext.db/transf.'/
C
C
C
      INCLUDE ias-util.comexwsbf,LIST
C
C ... Table of segments in EXWS
C
      COMMON /exwsbf/ ibuff(lrec14),iexwst(100)
      DATA iexwst /1,2,3,4,5,6,7,8,9,10,40*0,51,52,53,15*0,69,70,
     *     75,29*0/
C
C
C
      INCLUDE ias-util.dclchrsys,LIST
      CHARACTER*20 osflnm(13:18)
C
      INCLUDE ias-util.comchrsys,LIST
C
C ... COMMON CHRSYS containing IAS-SYSTEM character information
C
C     OSFLNM...... Table of OS file names
C           (13) ... data base name
C           (14) ... external working storage file
C           (15) ... internal OS file for (user) communication
C           (16) ... message file
C           (17) ... input log
C           (18) ... output log
C
      COMMON /chrsys/ osflnm
```

```
      DATA osflnm /' ','EXWSFL','INTCOM','IAS-80*IAS-MSG'
     *    ,'INPLOG','OUTLOG'/
C
      END
```

```
C ... ***  Program for closing and de-assigning operating system     ***
C ... ***  files                                                     ***
C ... ***
C ...
C
C
C ... COMMENT
C
C     This is an unportable routine of the first kind because
C     of the configuration dependent handling of the facility
C     return codes.
C
C     The reader should refer to ANSI X3.9-1978 FORTRAN and to
C     UP-8244.1 SPERRY UNIVAC series 1100 FORTRAN(ASCII) or
C     to appropriate documentation of other vendors if this
C     program is to be implemented on other machine ranges.
C
C     All character variables must be uppercase on input!!!
C
C
C ... INPUT BY PARAMETERLIST
C
C     IUNIT   I         unit number to be closed
C     FILE    C*20 <    file name to be closed (may be space)
C     FILSTA  C*7  <    close-status: 'KEEP' or 'DELETE'
C     IFREE   I         free (deassign) file if IFREE .le. 0,
C                       but keep cataloged if STATUS = 'KEEP'
C     INDPRT  I         print error messages if INDPRT .ge. 1
C                       print warnings if INDPRT .ge. 2
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     IERRCD  I         error code
C
C
C ... RETURN            normal exit
C ... RETURN 1          error exit
C ...
C
      SUBROUTINE yclose(*,ierrcd,iunit,file,filsta,ifree,indprt)
C
      CHARACTER*(*) file,filsta
      CHARACTER*20 lofile
      CHARACTER*20 yqmacd,cerrcd
      CHARACTER*7 lostat
      CHARACTER*6 rdkey,wrkey
      INCLUDE ias-util.dclcommsg,LIST
      INCLUDE ias-util.comcommsg,LIST
      INCLUDE ias-util.dclcsign,LIST
      INCLUDE ias-util.comcsign,LIST
C
      ierrcd = 0
      lofile = file
      lostat = filsta
```

```
C
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
C
C ... closing error
C
         ierrcd = -4220
         IF(indprt.ge.2) THEN
            cerrcd = yqmacd(ier0cd)                                SEM
            msgin(1:20) = cerrcd                                   SEM
            WRITE(msgin(21:22),FMT='(i2)') iunit                   SEM
            msgin(23:23) = msgstp                                  SEM
            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='(i2)') 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
```

```
C ... ***  Program for providing date and time                        ***
C ... ***  (1)  in CHARACTER FORMAT    YYYYMMDDHHMMSS  (see below)    ***
C ... ***  (2)  in BIT-FORMAT    YYYYYMMMMDDDDDHHHHHMMMMMMSSSSSS      ***
C ... ***                                           (see below)    ***
C ... ***
C ...
C
C
C ... COMMENT
C
C     This is an unportable routine of first order
C
C
C ... INPUT BY PARAMETERLIST
C
C     No input from parameterlist
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     DATTIM  C*14 >    date and time in CHARACTER FORMAT
C                       YYYYMMDDHHMMSS
C
C     IDATIM  I         date and time in BIT FORMAT
C
C                       32    26  22   17   12     7    1
C                         YYYYYYMMMMDDDDDHHHHHMMMMMMSSSSSS
C                       Y(6) M(4) D(5) H(5) M(6) S(6)
C
C                       IDATIM = (Y-1980) * 2**26
C                              +     M    * 2**22
C                              +     D    * 2**17
C                              +     H    * 2**12
C                              +     M    * 2**6
C                              +     S
C
C
C ... RETURN            unique exit
C                       no error exit
C
C
      SUBROUTINE ydatim (dattim,idatim)
C
      CHARACTER*14 dattim
      CHARACTER*8  date,time
C
      CALL zzdate(date,time)
C
C ... Calculate INTEGER-value of date & time
C
C ... UNIVAC 1100 presents date in format MMDDYY
C
      READ(date,2)m2,m1,m3                                              SEM
2     FORMAT(3I2)                                                       SEM
      m3=m3-80
```

```
      isum=m3*2**26 + m2*2**22 + m1*2**17
C
C ... UNIVAC 1100 presents time in format HHMMSS
C
      READ(time,2)m3,m2,m1                                          SEM
      isum=m3*2**12 + m2*2**6 + m1 + isum
      idatim=isum
C
C ... Calculate CHARACTER-variable with date & time
C
      dattim='19'//date(5:6)//date(1:4)//time(1:6)
C
      RETURN
C
      END
```

```
C ... ***  Program for cataloging, assigning, freeing and deleting   ***
C ... ***  of operating system files                                 ***
C ... ***
C ...
C
C
C ... COMMENT
C
C     That is an unportable routine of first kind
C
C     The reader should refer to ANSI X3.9-1978 FORTRAN and to
C     UP-8244.1 SPERRY UNIVAC Series 1100 FORTRAN (ASCII) or the
C     appropriate documentation of other vendors if this program
C     is to be implemented on other machine ranges.
C
C     All character variables must (currently) be uppercase on input !!
C
C     This routine contains explicit WRITE-statements rather then
C     calling the messages from the message file. This cannot be
C     avoided, however, as the message file might not be available
C     before having been assigned by this program.
C
C
C ... INPUT BY PARAMETERLIST
C
C     FILE    C*20     operating system file name
C     RDKEY   C*6      read key for specified file
C     WRKEY   C*6      write key for specified file
C     IFILST  I        status for OPEN- or CLOSE-statement
C                      IFILST = 4  <=>  STATUS = 'NEW'      (OPEN)
C                      IFILST = 3  <=>  STATUS = 'EXTEND'   (OPEN)        SEM
C                      IFILST = 2  <=>  STATUS = 'OLD'      (OPEN)
C                      IFILST = 1  <=>  STATUS = 'UNKNOWN'  (OPEN)
C                      IFILST =-1  <=>  STATUS = 'KEEP'     (CLOSE)
C                      IFILST =-2  <=>  STATUS = 'DELETE'   (CLOSE)
C     ITRKS   I        number of tracks to be requested
C     INDEXC  I        indicator if file is to be assigned exclusively
C                      (INDEXC=1) or not (INDEXC=0)
C     IWAIT   I        indicator if program should be kept in a wait
C                      position if the file is temporarily not
C                      available (wait if IWAIT=1, don't wait if
C                      IWAIT=0)
C     IFREE   I        Free file before assigning it
C     INDPRT  I        Print error messages if INDPRT .ge. 1
C                      Print warnings if INDPRT .ge. 2
C                      Print program trace if INDPRT .ge. 3
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     IERRCD  I        error code (facility key code)
C
C
C ... RETURN           normal exit
C ... RETURN 1         error exit
```

```
C ...
C
      SUBROUTINE yfile (*,ierrcd,file,rdkey,wrkey,ifilst,itrks,indexc
     *     ,iwait,ifree,indprt)
C
      CHARACTER*52 asgstr
      CHARACTER*30 frestr
      CHARACTER*20 file
      CHARACTER*6  key(2),rdkey,wrkey
C
      INCLUDE ias-util.comstatus,LIST
C
      ierrcd = 0
      asgstr = ' '
      frestr = ' '
C
C ... Execute only if UNIVAC
C
      IF(isize(11).NE.1)GOTO 900
C
      key(1) = rdkey
      key(2) = wrkey
C
C ... evaluate exact file name (size)
C
C     IPOS    I        last character of PROJ-ID*FILE[.] in FILE
C     IPOS2   I        position of last character in ASGSTR
C
      ipos = jlastc(file)
      IF (ipos.le.0) GOTO 950
C
C ... generate general part of asgstr
C
      IF (ifilst.GE.2) THEN
         asgstr(1:5) = '@ASG,'                                        SEM
         ipos2 = 10+ipos                                              SEM
         asgstr(11:ipos2) = file(1:ipos)                              SEM
C
C ... transfer of keys
C
         DO 400 i0 = 1,2
            ipos2 = ipos2+1
            asgstr(ipos2:ipos2) = '/'                                 SEM
            iposc = jlastc(key(i0))
C
C ... non-blank key ?
C
            IF (iposc.gt.0) THEN
               ipos1 = ipos2+1
               ipos2 = ipos2+iposc
               asgstr(ipos1:ipos2) = key(i0)(1:iposc)                 SEM
            END IF
C
400      CONTINUE
C
```

```
C ... number of tracks specified ?
C
          IF(itrks.gt.0) THEN
             ipos1 = ipos2+1
             ipos2 = ipos2+8
             asgstr(ipos1:ipos2) = ',///     '                              SEM
             WRITE(asgstr(ipos1+4:ipos2),'(i4)') itrks                      SEM
          ENDIF
          ipos2 = ipos2+3                                                   SEM
          asgstr(ipos2-2:ipos2) = ' . '                                     SEM
C
C ... file to be catalogued ?
C
          IF (ifilst.EQ.4) THEN
             asgstr(6:10) = 'UP'                                            SEM
             CALL zcsfpt(*990,ier0cd,asgstr,indprt)                         SEM
          ENDIF
       ENDIF
C
C ... free file if IFREE.gt.0
C
       IF (ifree.GT.0.OR.ifilst.EQ.4) THEN
          frestr = '@FREE    '//file(1:ipos)//' . '                         SEM
          IF(ifilst.EQ.-2) frestr(6:7) = ',D'                               SEM
          CALL zcsfpt(*990,ier0cd,frestr,indprt)
       END IF
C
C ... assignment of file, if status = 'NEW' or 'OLD'
C
       IF (ifilst.GE.2) THEN
          asgstr(6:10) = 'A'                                                SEM
          IF(indexc.eq.1) THEN
             asgstr(7:7) = 'X'                                              SEM
             iposc = 8
          ELSE
             iposc = 7
          ENDIF
          IF(iwait.ne.1) asgstr(iposc:iposc) = 'Z'                          SEM
          CALL zcsfpt(*990,ier0cd,asgstr,indprt)                            SEM
       ENDIF
C
900    ierrcd = ier0cd
       RETURN
C
C ... Error messages
C
950    WRITE(*,*)'File specification missing'
       ier0cd=1
990    ierrcd=ier0cd
       RETURN 1
C
       END
```

```
C ... ***  Program for inquiring about files (calling FORTRAN        ***
C ... ***  INQUIRE).                                                 ***
C ... ***
C ...
C
C
C ... COMMENT
C
C     In order to separate all file handling into special modules with a
C     tendency to non-portability the INQUIRE-statement is located only
C     in this program unit and in no other program unit throughout the
C     whole program.
C
C     Compare ANSI X3.9-1978, chapter 12.2.1, for the definition of
C     file existence. File existence does not necessarily imply that
C     the file is contained in some file directory. On the other hand
C     a file may be known to the processor but does not exist for the
C     program.
C
C
C ... INPUT BY PARAMETERLIST
C
C     FILE    C*(*)    name of the file being inquired about
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     IERRCD  I        error code (currently not used)
C     IEXIST  I        existence indicator
C                      <0 : file does not exist
C                      >0 : file exists
C     IDUMMY  I        currently not used
C
C
C ... RETURN           unique exit
C ...
C
      SUBROUTINE yinqu (ierrcd,file,iexist,idummy)
C
      CHARACTER*(*) file
      LOGICAL exist
C
      INQUIRE(IOSTAT=ierrcd,FILE=file,EXIST=exist)
C
      IF (exist) THEN
         iexist=1
      ELSE
         iexist=-1
      END IF
C
      idummy=0
C
      END
```

```
C ... ***  Program for editing and printing I/O-errors                ***
C ... ***
C ...
C
C
C ... COMMENT
C
C     This is an unportable routine of first order
C
C
C ... INPUT BY PARAMETERLIST
C
C     IREC    I        Record number (-99999 if not applicable)
C     INDPRT  I        Print  indicator.
C                      Print an error message if the machine
C                      independent error code is .LE. INDPRT*10.
C                      Nothing is printed if INDPRT.LE.0,
C                      everything is printed if INDPRT.GE.9
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     No output by parameterlist
C
C
C ... TRANSPUT BY PARAMETERLIST
C
C     IOERCD  I        I/O error code
C                      on input:  machine dependent
C                      on output: machine independent
C                       1: other error
C                      21: attempted to read from unassigned mass
C                          storage area and the record read was not
C                          the first one. This means mostly that the
C                          file had no ENDFILE record.
C                      31: attempted to read from unassigned mass
C                          storage area and the record read was
C                          the first one. This means mostly that a
C                          READ was performed on an empty file.
C                      41: attempted to read an empty record from
C                          a direct access file
C
C
C ... RETURN           unique exit
C ...
C
      SUBROUTINE yioerr (ioercd,irec,indprt)
C
      CHARACTER*8 arec
C
      INCLUDE ias-util.comstatus,LIST
C
      IF (ioercd.eq.0) RETURN
      IF (iswtch(4).GT.0) THEN
         locprt=9
```

```
      ELSE
         locprt=indprt
      END IF
C
      ios=ioercd/2**27                                              SEM
      irest=ioercd-ios*2**27                                        SEM
      iou=irest/2**18                                               SEM
      ioc=irest-iou*2**18                                           SEM
C
      IF (ioc.EQ.1.AND.ios.EQ.5) THEN                               SEM
         IF (irec.EQ.1) THEN
            ioercd=31
         ELSE
            ioercd=21
         END IF
C
      ELSE IF (ioc.EQ.1053) THEN                                    SEM
         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)
      END IF
C
      RETURN
      END
```

```
C ... ***  Program for assigning and opening files and i/o-units.     ***
C ... ***  This Program calls file-handling procedure YFILE for       ***
C ... ***  assignment of an operating system file.                    ***
C ... ***
C
C
C
C
C ... COMMENT
C
C     This is an unportable routine of the first kind because
C     of the configuration dependent computation of the number
C     of needed tracks on mass-storage and the handling of the
C     facility return codes
C
C     The reader should refer to ANSI X3.9-1978 FORTRAN and to
C     UP-8244.1 SPERRY UNIVAC series 1100 FORTRAN(ASCII) or
C     to appropriate documentation of other vendors if this
C     program is to be implemented on other machine ranges.
C
C     All character variables must be uppercase on input!!!
C
C     This routine contains explicit WRITE-statements rather
C     than calling the messages from the message file. This
C     cannot be avoided, however, as the message file is not
C     available before being opened by this program.
C
C
C ... INPUT BY PARAMETERLIST
C
C     IUNIT   I        file reference number
C     FILE    C*20 <   file name to be associated with the spec. unit
C     RDKEY   C*6 <    read key for specified file
C     WRKEY   C*6 <    write key for specified file
C     FILSTA  C*7 <    status for OPEN-statement.
C                      Status may be 'OLD', 'NEW' or 'UNKNOWN'.
C                      'SCRATCH' is not implemented.
C                      In accordance with the proposals for FORTRAN 8x
C                      status 'EXTEND' has been added.                    SYN
C                      For a D/A file NRCDS may be increased with         SYN
C                      status 'EXTEND'. A sequential file will be         SYN
C                      positioned after the current last record           SYN
C                      (before the ENDFILE-record, if one exists)         SYN
C     ACCESS  C*10 <   access method: 'SEQUENTIAL' or 'DIRECT'
C     FORM    C*11 <   format: 'FORMATTED' or 'UNFORMATTED'
C     IRECL   I        record length for DIRECT ACCESS FILES
C     NRCDS   I        maximum number of records in a D/A file            SYN
C                      (RCDS = nrcds is used in the OPEN-statement        SYN
C                      but this clause is not standard conforming)        SYN
C     INDWRT  I        indicator if writing is prohibited (INDWRT=0)
C                      or allowed (INDWRT=1)
C     INDEXC  I        indicator if file is to be assigned exclusively
C                      (INDEXC=1) or not (INDEXC=0)
C     IWAIT   I        indicator if program should be kept in a wait
C                      position if the file is temporarily not
```

```
C                         available (wait if IWAIT=1, don't wait if
C                         IWAIT=0)
C     IFREE   I           Free file before assigning it if IFREE .gt. 0.
C                         This parameter is not used if filsta = 'NEW'.
C                         A file with status 'NEW' is always freed
C                         -- after -- having been cataloged and than
C                         considered to have status 'OLD'
C     LMSIZE  I           Indicator if (processor dependent) mass storage
C                         size is to be computed (LMSIZE.gt.0) or not
C     INDPRT  I           Print error messages if INDPRT .ge. 1
C                         Print warnings if INDPRT .ge. 2
C                         Print program trace if INDPRT .ge. 3
C                         or if ISWTCH(4) .GT. 0
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     IERRCD  I           error code
C                         attention to
C                         -1 ... read key missing
C                         -2 ... write key missing
C
C
C ... RETURN              normal exit
C ... RETURN 1            error exit
C ...
C
      SUBROUTINE yopen(*,ierrcd,iunit,file,rdkey,wrkey,filsta,access
     *    ,form,irecl,nrcds,indwrt,indexc,iwait,ifree,lmsize,indprt)
C
      CHARACTER*(*) file,rdkey,wrkey,filsta,access,form
      CHARACTER*7 lostat
      CHARACTER*6 key(2)
C
      INCLUDE ias-util.comstatus,LIST
C
C ... transferring dummy variables to local variables
C
      locprt=MAX(indprt,3*iswtch(4))
      ierrcd = 0
      lostat = filsta
      key(1) = rdkey
      IF(indwrt.eq.0) THEN
         key(2) = ' '
      ELSE
         key(2) = wrkey
      ENDIF
C
C ... compute numerical index for STATUS-specifier
C
      IF (lostat.EQ.'NEW') THEN
         ifilst = 4
      ELSE IF (lostat.EQ.'EXTEND') THEN
         ifilst = 3
      ELSE IF (lostat.EQ.'OLD') THEN
```

```
            ifilst = 2
         ELSE IF (lostat.EQ.'UNKNOWN') THEN
            ifilst = 1
         ELSE
            GOTO 920
         ENDIF
C
C ... compute machine dependent size (number of tracks) to be requested
C
         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                    SEM
               itrks = iwords/1792+5                                  SEM
C
            ELSE
               WRITE(*,*)'Warning: File not write-enabled, mass '
     *            ,' storage size not calculated'
               itrks=0
            END IF
C
         ELSE
            itrks = 0
         ENDIF
C
C ... generate specification string for assignment and assign
C ... operating system file
C
         CALL yfile(*990,ier0cd,file,key(1),key(2),ifilst,itrks
     *      ,indexc,iwait,ifree,locprt)
C
         IF (isize(11).EQ.1.AND.ifilst.eq.4) lostat='OLD'
C
C ... FORTRAN-opening for assigned file
C ... N.B.: Status = 'EXTEND' is allowed but not standard conforming    SYN
C
         IF(access.eq.'SEQUENTIAL') THEN
            OPEN(UNIT=iunit,IOSTAT=ier0cd,FILE=file,STATUS=lostat
     *      ,ACCESS=access,FORM=form)
         ELSE IF(access.eq.'DIRECT') THEN
            OPEN(UNIT=iunit,IOSTAT=ier0cd,FILE=file,STATUS=lostat
     *      ,ACCESS=access,FORM=form,RECL=irecl,RCDS=nrcds)           SYN
         ELSE
            WRITE(*,*)'Invalid ACCESS specifier ',access
            CALL syserr(*990,'YOPEN')
         ENDIF
C
         IF(ier0cd.eq.0) THEN
```

```
          ierrcd=0
C
       ELSE
C
C ... opening error
C
          WRITE(*,*)'Error when opening unit ',iunit
          CALL yioerr (ier0cd,-99999,9)
          GOTO 990
       ENDIF
C
       RETURN
C
C ... other error messages
C
920    WRITE(*,*)'Invalid STATUS specifier ',lostat
       ier0cd=101
       CALL syserr(*990,'YOPEN')
C
990    ierrcd=ier0cd
       RETURN1
       END
```

```
C ... ***  FUNCTION subprogram for translating an integer variable    ***
C ... ***  to a character variable (machine representation of the    ***
C ... ***  binary code) with length of 20 char's                      ***
C ... ***
C ...
C
C
C ... COMMENT
C
C     This is a FUNCTION subprogram.
C
C     This is an unportable routine of first kind.
C
C     However, if octal code is used to represent machine code, it
C     is very likely that this program runs on the corresponding
C     machine. On byte machines it has to be changed to hexadecimal
C     code.
C
C
C ... INPUT BY PARAMETERLIST
C
C     INTEG   I        input variable
C
C
C ... OUTPUT BY FUNCTION VALUE
C
C     YQOCT   C*20 >   octal code output variable
C
C
C ... RETURN           normal and error exit
C ...
C
      CHARACTER*20 FUNCTION yqmacd (integ)
C
      WRITE (yqmacd,FMT='(O20)',IOSTAT=ierrcd) integ                  SYN
C
      IF (ierrcd.ne.0) yqmacd='777777777777'
C
      RETURN
      END
```

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

```
C                     If IMATH<=0 or if more than IMATH math-faults
C                     have occured, the program will terminate.
C
C     IEXCEP  I       max number of erroneous exceptions
C                     captured by the exception handling program.
C                     If IEXCEP<=0 or if more than IEXCEP erroneous
C                     exceptions have occured, the program will terminate.
C
C ... OUTPUT BY PARAMETERLIST
C
C     No output by parameterlist
C
C
C ... RETURN          unique exit
C ...
C
      SUBROUTINE yset (iundfl,iovfl,idiv,imath,iexcep)
C
      CALL zundst(iundfl)                                                SEM
      CALL zovfst(iovfl)                                                 SEM
      CALL zdivst(idiv)                                                  SEM
      CALL zcmlst(imath)                                                 SEM
      CALL zexcep(iexcep)                                                SEM
C
      RETURN
      END
```

```
C ... ***  Program to prepare the calling of routine ZCSF in order    ***
C ... ***  to perform an ACSF$ executive request (e.g. assign,         ***
C ... ***  catalog and free operating system files)                    ***
C ... ***
C ...
C
C
C ... COMMENT
C
C     This is an unportable routine of second order
C
C
C ... INPUT BY PARAMETERLIST
C
C     IMAGE   C*80 <   image of executive control statement.            SEM
C                      This image must be terminated by the character  SEM
C                      sequence : blank period blank                    SEM
C
C                      EXAMPLE:
C                      '@ASG,AXZ MYPROG*OLDFILE . '                     SEM
C
C     INDPRT  I        Print error messages if INDPRT .ge. 1
C                      Print warnings if INDPRT .ge. 2
C                      Print program trace if INDPRT .ge. 3
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     IERRCD  I        Error code. The following special codes apply :
C                        -1 : write key missing
C                        -2 : read key missing
C                       423 : facility error, including read and write
C                             key (both) missing
C                      -424 : facility warning other than read or write
C                             key missing
C
C
C ... RETURN           normal exit
C ... RETURN 1         error exit
C ...
C
      SUBROUTINE zcsfpt (*,ierrcd,csfstr,indprt)
C
      CHARACTER*(*) csfstr
      CHARACTER*20 cerrcd,yqmacd
      CHARACTER*1  rdwrky
C
      INCLUDE ias-util.comstatus,LIST
      ipos=len(csfstr)
C
C ... Call actual executive request  ER ACSF$                           SEM
C
      CALL zzcsf (ier0cd,csfstr(1:ipos))
C
      IF (ier0cd.lt.0) THEN                                             SEM
```

```
            ierrcd=423
         ELSE IF (ier0cd.GT.0) THEN                                      SEM
            ierrcd=-424
         ELSE
            ierrcd=0
         END IF
C
         IF (indprt.lt.3.and.ier0cd.eq.0) RETURN
C
C ... Print error messages
C
         cerrcd=yqmacd(ier0cd)                                           SEM
C
C ... Read/write keys missing ?
C
         rdwrky=cerrcd(12:12)
         IF (rdwrky.eq.'3') THEN                                         SEM
            cerrcd(9:9)='4'                                              SEM
            ierrcd=423
         else IF (rdwrky.eq.'1') THEN                                    SEM
            ierrcd=-1
         else IF (rdwrky.eq.'2') THEN                                    SEM
            ierrcd=-2
         END IF
C
C ... Printout of CSF-string ?
C
         IF (indprt.ge.3.or.
     *      (indprt.ge.2.and.ierrcd.lt.0).or.
     *      (indprt.ge.1.and.ierrcd.gt.0)) THEN
            ipos1 = INDEX(csfstr(1:ipos),'/')
            IF (ipos1.GT.0) ipos=ipos1
            WRITE (iout,92) csfstr(1:ipos)
92          FORMAT(1X,A)
         END IF
C
         IF (ierrcd.gt.0.and.indprt.ge.1) WRITE (iout,94) cerrcd
94       FORMAT (' Facility error ',A)
         IF (ierrcd.lt.0.and.indprt.ge.2) WRITE (iout,96) cerrcd
96       FORMAT (' Facility warning ',A)
C
         IF (ierrcd.gt.0) RETURN 1
         RETURN
         END
```

THE CONVERSION ROUTINES WITHIN THE IAS-SYSTEM

ST5ENC

ST5DEC

ST4ENC

ST4DEC

```
C ... ***  Subprogram for converting a character string to a vector  ***
C ... ***  of numeric keys                                            ***
C ... ***
C ...
C
C
C ... INPUT BY PARAMETERLIST
C
C     STRING  C        Character string
C     KEYDIM  I        Dimension of KSTR
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     KSTR    I(*)     Numeric keystring
C
C
C ... Special Case
C
C     STRING  C*20     Contains the file name, type, element name and
C                      version name starting at positions 1,9,11 and 19
C
C     KSTR             Contains the numeric keys calculated from STRING
C
C
C ... RETURN           Normal exit
C ... RETURN 1         Error exit
C ...
C
      SUBROUTINE st5enc(*,ierrcd,string,keydim,kstr)
C
      CHARACTER*(*) string
      CHARACTER*(1) chr
      DIMENSION kstr(keydim)
C
      INCLUDE ias-util.dclcommsg,LIST
      INCLUDE ias-util.comcommsg,LIST
      INCLUDE ias-util.dclcsign,LIST
      INCLUDE ias-util.comcsign,LIST
C
      ier0cd=0
      length=len(string)
C
      kdim=((length-1)/5)+1
      IF(kdim.LE.keydim)THEN
         mpos=0
         DO 220 i=1,kdim
            ikey=0
200         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
220      CONTINUE
      ELSE
         ier0cd=2011
         CALL msgprt(*990,ier0cd)
         CALL syserr(*990,'ST5ENC')
      END IF
      RETURN
990   ierrcd=ier0cd
      RETURN 1
      END
```

```
C ... ***  Subprogram for converting a vector of numeric keys          ***
C ... ***  to a character string                                       ***
C ... ***
C ...
C
C
C ... INPUT BY PARAMETERLIST
C
C     KSTR    I        Numeric keystring
C     KEYDIM  I        Dimension of KSTR
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     STRING  C*(*)    Character string
C
C
C ... Special Case
C
C     KSTR    I        Is a vector of four keys
C
C     KSTR(1) I        Is calculated from the first 5 characters of file
C     KSTR(2) I        Is calculated from the last 3 characters of file
C                      and the type of the element
C     KSTR(3) I        Is calculated from the first 5 characters of element
C     KSTR(4) I        Is calculated from the last 3 characters of element
C                      and the 2 characters of version
C
C
C ... RETURN           normal exit
C ... RETURN 1         error exit
C ...
C
      SUBROUTINE st5dec(*,ierrcd,kstr,keydim,string)
C
      CHARACTER*(*) string
      CHARACTER*(1) qchr
      DIMENSION kstr(keydim)
C
      INCLUDE ias-util.dclcommsg,LIST
      INCLUDE ias-util.comcommsg,LIST
      INCLUDE ias-util.dclcsign,LIST
      INCLUDE ias-util.comcsign,LIST
C
      ier0cd=0
      length=len(string)
C
C ... Conversion of kstr to string
C
      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
```

```
               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.i.EQ.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   ierrcd=ier0cd
      RETURN 1
      END
```

```
C ... ***  SUBROUTINE to encode an arbitrary string in processor     ***
C ... ***  dependent code into an IAS-SYSTEM code by use of          ***
C ... ***  FORTRAN 77 function ICHAR                                 ***
C ... ***
C ...
C
C
C ... COMMENT
C
C     Four characters are encoded in every word, using eight bits per
C     character. The sign bit has to be treated in a special way
C     for the sake of words with only 32 bits, one of which is the sign
C     bit.
C
C
C ... INPUT BY PARAMETERLIST
C
C     STRING  C*(*)   String to be converted
C     IDIM    I       Dimension of vector ISTRCD
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     ISTRCD   I(IDIM) Converted string
C
C
C ... RETURN           Normal exit
C ... RETURN 1         Error exit
C ...
C
      SUBROUTINE st4enc (*,ierrcd,string,idim,istrcd)
C
      CHARACTER*(*) string
      LOGICAL minus
      DIMENSION istrcd(idim)
C
      INCLUDE ias-util.comstatus,LIST
C
      ierrcd=0
      iword=1
      ibyte=0
      CALL vecnul(istrcd,1,idim)
C
      length=LEN(string)
      numwrd=(length+3)/4
C
      IF (idim.NE.numwrd) THEN
         CALL syserr(*990,'ST4ENC')
      END IF
C
C ... Loop over number of characters in input string
C
      DO 600 i0=1,length
         i0char=ICHAR(string(i0:i0))
C
```

```
C ... Character out of range ?
C
         IF (iOchar.GT.254) THEN
            iOchar=0
            ierrcd=-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 (iOchar.GT.127.AND.isize(13).LT.32) THEN
               minus=.TRUE.
               iOchar=254-iOchar
            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=iOchar*2**(8*(4-ibyte))
         IF (minus) THEN
            ivalue=ivalue-iadd
         ELSE
            ivalue=ivalue+iadd
         END IF
         istrcd(iword)=ivalue
600   CONTINUE
C
      RETURN
C
990   RETURN 1
      END
```

```
C ... ***  SUBROUTINE to convert the code generated by subroutine     ***
C ... ***  ST4ENC back to a character string in processor dependent  ***
C ... ***  code, using FORTRAN 77 CHAR-function                      ***
C ... ***
C ...
C
C
C ... COMMENT
C
C
C
C ... INPUT BY PARAMETERLIST
C
C     ISTRCD  I(IDIM) Code generated by ST4ENC
C     IDIM    I       Number of words to be converted
C
C
C ... OUTPUT BY PARAMETERLIST
C
C     STRING  C*(*)   Output string
C
C
C ... RETURN          Normal exit
C ... RETURN 1        Error exit
C ...
C
      SUBROUTINE st4dec (*,ierrcd,istrcd,idim,string)
C
      CHARACTER*(*) string
      LOGICAL minus
      DIMENSION istrcd(idim)
C
      ierrcd=0
      iword=1
      ibyte=0
      string=' '
C
      length=LEN(string)
      numwrd=(length+3)/4
      IF (idim.NE.numwrd) THEN
         CALL syserr(*990,'ST4DEC')
      END IF
C
      DO 600 i0=1,length
         ibyte=ibyte+1
C
C ... New word (but not first 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=istrcd(iword)
         IF (iword.LT.0) THEN
            minus=.TRUE.
            ivalue=-ivalue
         ELSE
            minus=.FALSE.
         END IF
      END IF
C
      idiv=2**(8*(4-ibyte))
      i0char=ivalue/idiv
      ivalue=ivalue-i0char*idiv
      IF (ibyte.EQ.1.AND.minus) THEN
         i0char=254-i0char
      END IF
C
      IF (i0char.GT.0) THEN
         string(i0:i0)=CHAR(i0char)
      ELSE
         string(i0:i0)=' '
      END IF
600   CONTINUE
C
      RETURN
C
990   RETURN 1
      END
```