Introduction to the Use of the UNIX-Version of the KArlsruhe PROgram System KAPROS FZKA 6280

D. Woll

September 18, 2013

Rebuild from original resources July 1999
by C.H.M. Broeders September 2013

Contents

1 Introduction

The goal of this report is to provide a **brief introduction** to the **UNIX**-version of the **KA**rlsruhe **PRO**gram **S**ystem **KAPROS** so that the user becomes quickly acquainted with KAPROS and can appreciate the specific features and the possible advantages of KAPROS compared to other systems. It is not intended to give a detailed description of KAPROS. For this purpose several reports of the MVS-version running until 1998 on IBM computers exist, which are essentially valid for the UNIX-version too. There exists also a comprehensive **computer internal documentation** accessible e.g. by the UNIX script "**ksinfo**" including more detailed information also of aspects not covered in this report.

2 History of the KAPROS-Development

Since 1965 at KfK program systems were used, that manage sequencing of programs and organize the flow of data between the programs in datablocks.
KAPROS was developed since 1973 as a successor of the first NUclear program SYStem NUSYS at KfK for an IBM/370-168 computer with its relatively small storage capacity usual at that time (a KAPROS-job was able to use about 300 kbyte storage) [1a, 1b, 1c]. Most parts of KAPROS were written in **FORTRAN** except for some Assembler routines that interacted closely with the operating system OS-MVT (Multiprogramming with a Variable number of Tasks). The storage available for a KAPROS-job was managed by KAPROS itself. To make optimal use of the precious storage capacity, datablocks could be divided into parts for storing them in the main storage. The management of this datablock-parts in extensive and complicated tables turned out to be a considerable disadvantage. For saving storage KAPROS has been designed in an overlay-structure.
In 1982 a computer SIEMENS Fujitsu 7890 running under the operating system OS-MVS (Multiple Virtual Storage) was installed. Making better use of the new capabilities of this computer generation a new version of KAPROS was developed [2]. The overlay-structure was no longer necessary because a storage of 1 Mbyte now could be used.
For use on a CYBER 205 computer, it was necessary to develop a revised version of KAPROS, KSSK, using special CYBER-FORTRAN-features instead of IBM-Assembler-routines [3]. Taking advantage of the available large storage, subdividing of datablocks was no longer necessary. In KSSK the possibility of storing data on disk was not provided. In order to facilitate the portability of KAPROS, the importance increased to avoid subroutines written in Assembler as far as possible. Therefore on the basis of KSSK the KAPROS-version KSSKBU [4] was developed for IBM 3090 again using external storage. Management of the storage region available for KSSKBU was done by the operating system, therefore only few Assembler-routines were still necessary further on. Later KSSKBU was extended to KAPROS3 [5]. Because of addressing restrictions this version could utilize only about 8 Mbyte of the available storage. In order to make use of the whole available storage volume of more than 64 Mbyte upgraded versions had been developed using the extended area XA of the MVS-sytem [4], [6].
In order to enable burnup-calculations with KARBUS [7] on workstations running under **UNIX** the current version KSSKUX was developed [8] on the basis of KSSKBU-XA-version and KAPROS3 using the programming language **C** instead of Assembler for interacting with the operating system.

3 Introduction to the UNIX-Version of KAPROS

KAPROS enables the user to **call** calculation **programs**, the so called **modules**, in freely chosen order and **organizes** the **flow of data** between the modules in the so called **lifeline**. Furthermore KAPROS allows the user to write own modules in the programming language **FORTRAN** which may be used to combine existing **modules** or to carry out calculations not yet included in the existing modules. They can be added in an easy way to the already existing package of modules. **Archives** allow to store calculated data so that they can be used subsequently in other jobs.

The **UNIX**-version of **KAPROS** consists of **two main parts**,

- the so called system "**kernel**" and

interacting with but **independent of the kernel**,

- the **modules** and the associated application **libraries** (cross-sections etc.),

where executables, e.g. UNIX-scripts, can be processed in the same way as modules.

The **kernel** is of a rather general nature and can be used for **any task**, that requires a flexible sequence of program calls and an associated organisation of the flow of data.

For **storing data** the so called **lifeline** is used. It consists of two parts,

(1) the **internal** lifeline in the workspace of KAPROS extended by external files if necessary, working with transfer of data between modules and lifeline,

(2) and the so called **pointer**-lifeline, to which the modules can have access by pointers without actual datatransfer.

Data are stored in so called **datablocks**, labeled by the datablock**name** and an **index**.

UNIX-**scripts** make it easy to **run KAPROS** and to **compile** and **link modules**. Scripts may also be useful for postprocessing of results.

The KAPROS information system "**ksinfo**" enables the user to get an extensive documentation about the KAPROS-kernel and modules in a simple way.

4 Survey of the Existing Modules and the Associated Libraries

4.1 Modules

For **nuclear calculations** modules are available in the **KAPROS-library KSLIB** for the following tasks (the list in the annex shows them more detailed):

- Determination of atomic **number densities** for material compositions,
- Calculation and manipulation of multigroup **cross-sections**,
- **Cell calculations and/or heterogeneity correction**,
- **Neutron Diffusion** calculations,
- **Neutron Transport** calculations,
- Iterative determination of buckling or fuel enrichment,
- **Evaluations**, i.e reaction rates and combinations of them,
- Reactor **kinetics** parameters, based on perturbation theory,
- **Burnup** and **depletion** calculations,
- **Plot**-modules,

and independent from the nuclear calculations:

- Auxiliary modules for manipulating datablocks.

4.2 Libraries

4.2.1 Material-Dependent Microscopic Group Cross-Section Libraries

Libraries with material-dependent microscopic group cross-sections (mainly for neutron induced nuclear reactions) are available at the Forschungszentrum for following groups:

Number of Energy Groups	Identifications
11	SIMMER
26	SIMMER, KFKINR, KFKINR2
69	WIMSLIB
75	GR75LIB
208	GR208
275	GR275
334	GR334
100 neutron/23 photon	NEUTPHOT

These files are mainly used by the module GRUCAL. In order to get information about the included materials and cross-section types the user may use ”ksinfo”.

4.2.2 Burnup libraries

For burnup calculations with the module BURNUP following files are available containing data for:

light elements
heavy isotopes
fission products

These files are based on the latest versions of the libraries used by the stand-alone code KORIGEN (for details see corresponding input descriptions).

5 Preparing Input for a KAPROS-Job

Input for a simple KAPROS-job consists of **two parts**,

(1) the **input-datablocks** with KAPROS-commands to transfer them to the lifeline and

(2) the KAPROS-commands to **call** the **modules**.

In the following text words written in capital letters are key words.
Parameters enclosed in [ ] may be omitted.
$USER means the identification of the user.
Text written in *italics* is of minor interest.

Input-datablocks are stored in the lifeline by the KAPROS-command

***KSIOX DBN**=dbn,**IND**=ind,**TYP=CARD,PMN**=pmn

where

dbn is the name of the datablock
(up to 16 characters, right-handed blanks may be omitted),
ind is the index of the datablock (in simple cases usually 1) and
pmn is the name of a check-module.

The input **data** following the *KSIOX-command have to be written **freeformatted**, characterstrings have to be enclosed in apostrophes. The structure of the input-datablocks is described in the input-descriptions of the modules (in general stored in the computer and available e.g. by "ksinfo").
The input-data of a datablock have to be terminated by

***$*$**

Modules are **call**ed by

***GO SM**=module[,*MPARM=mparm*]

where

module is the name of the module, see the list in the annex.
mparm are up to 20 parameters transferred to the module.
They may contain the index of the input datablock,
a character-string up to 80 characters or
parameters for controlling the run of the KAPROS-job.

A KAPROS-job may contain a **sequence of *GO**-commands.
Instead of such a sequence of *GO-commands the user may also establish an own module containing a sequence of KSEXEC-calls (see chapter 7.2).
Annotations may be included in the input after ***$** followed by at least one blank.

6 Running a KAPROS-Job

To **run** a KAPROS-job the user may call the script ksuxgo:

ksuxgo input [additional parameters]

where input is the name of the input file prepared by the user containing datablocks and KAPROS-commands for calling modules.
The meaning of the additional parameters for special applications can be obtained by call of ksuxgo without arguments.

Additional files, for example archives, are usually expected by KAPROS under the name KSUX.$USER.FTnt connected by a symbolic link with the real file.
nt means the unit-number for using the file (two digits with leading zeros).

The user can observe the KAPROS-run on screen, where all important messages will be shown.

KAPROS will create the standard-**output**file KSUX.$USER.FT07 containing the results of the calculations and the protocol-file KSUX.$USER.FT42 containing special messages, which may be useful especially for the interpretation of erroneous KAPROS-runs.

For using KAPROS it is necessary to extend the file .profile in following way:
- *storing the directory containing files belonging to KAPROS in the variable KAPROS_PATH (currently at the Forschungszentrum KAPROS_PATH=/fzk/inr/rs_aix41/KAPROS)*
- *including EXPORT=$KAPROS_PATH*
- *extending the sytem-variable $PATH by $PATH=$PATH:$KAPROS_PATH/bin*

7 Preparing Modules

7.1 KAPROS-Subroutines for Datatransfer

Modules usually have to be written as a FORTRAN-**subroutine** (see 9.2), the main-program will be included when linking the module. If one wants to develop own modules one has to consider two aspects:

(1) **Input/output of data** in datablocks of the lifeline and
(2) **calling** other **modules**.

The meaning of the variables appearing in the following calls of subroutines is explained at the end of this section.

Communication with the lifeline is possible in two ways:

(1) with **datatransfer** by call of the KAPROS-subroutines
CALL **KSGET** (DBN,IND,NF,K,N,IQ)
CALL **KSPUT** (DBN,IND,NF,K,N,IQ)
CALL **KSCH** (DBN,IND,NF,K,N,IQ)

KSGET will transfer data from the datablock DBN into the data array NF in the module (NF may also be a variable when N=1),
KSPUT will store the content of the variable or array NF into the datablock DBN,
KSCH will change already existing data in the datablock DBN, it works like KSPUT.

(2) in **pointer-technique** by call of the KAPROS-subroutines
CALL **KSGETP** (DBN,IND,N,A,IP,IQ)
CALL **KSPUTP** (DBN,IND,N,A,IP,IQ)

KSGETP will return a pointer IP, i.e. a number pointing to the first value of the datablock referred to a reference array A specified in the module. A(IP) contains the first value of the datablock. If necessary the datablock will be moved from the internal or external lifeline to the pointer-lifeline.

KSPUTP will return a pointer to free space for storing data in the pointer-lifeline referred to a reference array. A(IP) is the first address that can be used for storing data.
By means of **KSPUTP** it is possible, to **allocate workspace** used by the module for storing calculated data.

The **pointe**r of pointer-datablocks can be **released** by
CALL **KSCHP** (DBN,IND,IQ)
At the same time the datablock will be moved from the pointer-lifeline to the internal or external lifeline.

Datablocks can be **deleted** by
CALL **KSDLT** (DBN,IND,IQ)

The arguments of the subroutines mentioned above have the following meaning:

DBN	Name of the datablock (16 capital characters)
IND	Index of the datablock
NF	Array provided by the module, from which (KSPUT) or into which (KSGET) data are to be transferred
K	Position within the datablock DBN from which data are to be transferred
N	Number of words to be transferred or to be allocated
A	Reference array
IP	Pointer to the data relative to the reference array A A(IP) contains the first value of the datablock
IQ	Errorcode, 0 if no error has been detected (see also chapter 7.3 error handling)

7.2 KAPROS-Subroutines for Calling Modules

Modules can be **called** (even recursive) by use of the KAPROS-subroutine KSEXEC, for example in the simplest manner by:

CALL **KSEXEC** (MODUL,NDB, 0 ,DBNC_1,DBN_1,..,DBNC_{ndb},DBN_{ndb},IQ)

with

MODUL	Name of the module in capital letters (character*8 word)
NDB	Number of datablocks to be transferred to the called module
DBNC_i	Name of the datablock in the called module (i=1,NDB)
DBN_i	Name of the datablock in the calling module
IQ	Errorcode, 0 if no error has been detected (see also chapter 7.3 error handling)

In this case the indices of all transferred datablocks are supposed to be 1.

It is also possible to attach indices or to displace the indices of the datablocks between the calling and the called module. This allows e.g. to use a sequence of datablocks with the same name but different indices or to use a datablock, stored with an index different from 1, by a called module using index 1:

CALL KSEXEC (MODUL,NDB,NIND,$DBNC_1$,DBN_1,..,$DBNC_{ndb}$,DBN_{ndb},
$INDD_1$,..,$INDD_{nind}$,IQ)

with

NIND	*Number of datablocks with index-handling, NIND $\leq$ NDB,* *for the first NIND datablocks $DBNC_j$.* *The indices of the remaining (NDB - NIND) datablocks are supposed to be 1.*
$INDD_j$	*Arrays for index-handling of datablocks (j=1,NIND)* *$INDD_j$(1) first index of $DBNC_j$* *$INDD_j$(2) last index of $DBNC_j$* *$INDD_j$(3) Index-displacement from $INDC_j$ to IND_j*

This means that the index $INDC_j$ for a sequence of datablocks with the same name $DBNC_j$ for $INDD_j(1) \leq INDC_j \leq INDD_j(2)$ in the called module will correspond to $IND_j = INDC_j + INDD_j(3)$ in the calling module for the first declared NIND datablocks. For example NINDB datablocks with the same name and the indices 1 to NINDB can be transferred to the called module by INDD=(1,NINDB,0).

A datablock stored e.g. during an iteration using the number of the iteration NITER as index can be used in a called module with index 1 by means of INDD=(1,1,NITER-1).

A datablock-table will show the attachment of datablocks and corresponding indices for each call of a module in the output.

7.3 Error Handling

If an error occurs during a call of a KAPROS-subroutine (for example during a call of KSGET for a datablock that does not exist), it is possible to **reset** the **errorcode** IQ by call of KSCC and, if reasonable, to continue (for example by reading another datablock):

CALL **KSCC** (1,IQ)

If the errorcode will not be deleted, KAPROS will stop execution at the next call of a KAPROS-subroutine (except a call of KSCC).

On the other hand if the module has detected an error, for example an input error, it is possible to **define** an **errorcode** IQ by the module:

CALL **KSCC** (-1,IQ)

For ($90 \leq \text{IQ} \leq 99$) KAPROS will stop execution at the next call of a KAPROS-subroutine.

7.4 Initialization of a KAPROS-Module

In preceding versions of KAPROS it was necessary to initialize the start of the module by call of KSINIT. Currently KSINIT may be used to transfer the numbers of the I/O units and parameters for the task-time management from the kernel to the module:

CALL KSINIT (TC,DTC,NTIN,NTMESS,NTOUT)

with

TC, DTC	Time parameters depending on the system environment
NTIN	Number of the input unit
NTMESS	Number of the protocol unit
NTOUT	Number of the output unit

7.5 Compiling and Linking of KAPROS-Modules

In order to compile and link modules provided by users there exist two different ways:

(1) Compiling and linking during the KAPROS-job

(2) Storing load-modules in a user-library

KAPROS-modules of general interest should be included in the KAPROS-library KSLIB.

7.5.1 Compiling and Linking during the KAPROS-Job

Short control-modules may be compiled and linked during execution of the KAPROS-job, the load-module will be stored in the working directory.
The sourcecode of the module has to be included after the KAPROS-command

***COMPILE**

it has to be terminated by

***$*$**

*It is possible to include the sourcecode on an external file by use of the UNIT-parameter in the *COMPILE-command*

**COMPILE UNIT=nt*

where nt is the unit-number containing the sourcecode connected by symbolic link with the file KSUX.$USER.FTnt.

Several successive *COMPILE-commands can be concatenated. The sourcecode included by the preceding *COMPILE-commands will be compiled and linked by

***LINK**
MODNAM
***$*$**

where MODNAM (in this case no key-word) is the name of the module in capital characters, attributed to the module for calling it by a subsequent *GO-command or by another module.

7.5.2 Storing the Load-Module in a User-Library

For modules frequently used it is recommended to store the loadmodules in the user-library **$USER/KSLIB** (established previously by the user), using the script

ksuxcl MODNAM [additional parameters]

where MODNAM is the name of the module. It is expected, that a file MODNAM.f exists in the working directory.
The meaning of the additional parameters for special applications can be obtained by call of ksuxcl without arguments.

The sourcecode of each module has to start with the subroutine **ksskbu***:*
*(When using *COMPILE this subroutine will be generated automatically.)*

```
      subroutine ksskbu (mparm)
      integer mparm(*)
      call MODNAM (mparm)
      return
      end
```

*MPARM allows to transfer parameters defined in the *GO-command (see section 5).*

8 Using Archives

Archives provide a convenient way for long-term storage of results, e.g. for restart or postprocessing purposes.

By means of the parameter
  TYP=ARCI (**I**nto the lifeline) and
  TYP=ARCO (**O**ut of the lifeline) in the ***KSIOX**-command
it is possible to read datablocks from an archive into the lifeline or to store datablocks from the lifeline into an archive at the **end** of the KAPROS-job.
By means of the parameter
  SPEC=spec
the user may characterize the origin and content of the archive-datablock in the specification spec (up to 100 characters). It is possible to store several datablocks with the same name and index, distinguished by date and time of creation and by the specification.

When using TYP=ARCO for an archive including several datablocks with the same name and index but without specification, the newest datablock will be chosen.

During execution of the KAPROS-job data can be exchanged between lifeline and archive too by call of the KAPROS-subroutines KSARCI and KSARCO

  CALL **KSARCI** (DBN,IND,NT,SPEC,IQ)
  CALL **KSARCO** (DBN,IND,NT,SPEC,IQ)

with

DBN	Name of the datablock (16 capital characters)
IND	Index of the datablock
NT	unit-number of the archive connected by symbolic link with the file KSUX.$USER.FTnt > 0 direct access archive < 0 sequential archive
SPEC	Specification of the archive-datablock (up to 100 characters)
IQ	Errorcode, 0 if no error has has been detected

KSARCI will transfer archive-datablocks from the archive into the lifeline,
KSARCO from the lifeline into the archive.

Applying KSARCO during jobs running for a long time may be advisable for storing datablocks in an archive for later use in successive KAPROS-jobs.
Within a sequence of *GO-commands datablocks can be stored in an archive by using the module ARCO.
The file KSUX.ARCIO contains information about the datablocks transferred during the KAPROS-job.

Standard versions of sequential **archives** usually used may be **generated** in a simple way by the module ARCHIV using the following KAPROS-command
  ***GO SM=ARCHIV,MPARM**=nt,**'SEQ','GEN'**
where nt is the unit-number of the file including the archive connected by a symbolic link with the file KSUX.$USER.FTnt.

9 Examples for KAPROS-Jobs

9.1 Simple KAPROS-Job using *GO-commands

The following list shows an example for a simple KAPROS-job performing a zero-dimensional (fundamental mode) diffusion calculation determining k$_{eff}$ and neutron group fluxes for one mixture.
When calling GRUCAL the real file-names are expected instead of connecting the necessary libraries to files of the type KSUX.$USER.FTnt by symbolic link as usually used in KAPROS.

```
*KSIOX DBN=GRUCAL,TYP=CARD,IND=1,PMN=PRGRUC
*$ file definitions for GRUCAL
'CONTROL '
'/fzk/inr/rs_aix41/KAPROS/data/CONTROL '
'GRUBA   '
'/fzk/inr/rs_aix41/KAPROS/data/KFKINR '
'STEUER  '
'/fzk/inr/rs_aix41/KAPROS/data/F26 '
*$ input for GRUCAL
'GRUCAL  '
'KFKINR  ' '        ' '        '
'MISCH   '
1
7  '        ' '        ' '        '
'C       '  300.  1.360-5
'CR      '  300.  1.200-3
'FE      '  300.  3.955-3
'MO      '  300.  9.970-6
'NI      '  300.  9.845-4
'U 238   '  300.  3.994-2
'U 235   '  300.  1.625-4
'GRUCEND '
*$*$
*KSIOX DBN=INPUT DIFF0,IND=1,TYP=CARD,PMN=PRDUM
0  26  1  1  0.   1
*$*$
*GO SM=GRUCAL
*GO SM=DIFF0U
```

9.2 KAPROS-Job Including a User-Module

Instead of calling GRUCAL and DIFF0U by *GO-commands a user-module can be used calling GRUCAL and DIFF0U by means of the KAPROS-routine KSEXEC. It is assumed, that the job will be started from the working directory JOB of the user inr067.

```
*COMPILE
      subroutine ksskbu                       |  This subroutine
      call calc0                              |  may be omitted
      return                                  |  when using
      end                                     |  *COMPILE
      subroutine calc0
      call ksinit (tc,dtc,ntin,ntmess,ntout)
      call ksexec ('GRUCAL  ',2,0,
     * 'GRUCAL          ','GRUCAL          ',
     * 'SIGMN           ','SIGMN           ',
     * iq)
      if (iq.ne.0) go to 99
      call ksexec ('DIFF0U  ',3,0,
     * 'INPUT DIFF0     ','INPUT DIFF0     ',
     * 'SIGMN           ','SIGMN           ',
     * 'FLUX0           ','FLUX0           ',
     * iq)
      if (iq.ne.0) go to 99
      call ksget ('FLUX0           ',1,xkeff,3,1,iq)
      if (iq.ne.0) go to 99
      write (*,'(/''   CALC0: keff='',1pe10.4/)') xkeff
      write (ntout,'(/''   CALC0: keff='',1pe10.4/)') xkeff
      go to 90
   99 write (*,'(''Error in CALC0'')')
   99 write (ntmess,'(''Error in CALC0'')')
   90 return
      end
*$*$
*LINK CALC0
*$*$
*KSIOX DBN=GRUCAL,TYP=CARD,IND=1,PMN=PRGRUC
. . . same as in the preceding example
*$*$
*KSIOX DBN=INPUT DIFF0,IND=1,TYP=CARD,PMN=PRDUM
. . . same as in the preceding example
*$*$
*GO SM=CALC0
```

This job will produce the following output on screen:

```
 KSUXKERN: LIFELINE-SIZE = 8192000 bytes
 KSUXKERN: Separator 0
 KSUXKERN: SHARED MEMORY ATTACHED FROM 30000000 TO 307d0000

 VERSION KSUX-1.7

 PROTOKOLL-UNIT KSSKUN=42
 INPUT-UNIT     MODIN = 8
 OUTPUT-UNIT    MODOUT= 7


 K A P R O S - R U N :

      *COMPILE
      *LINK CALCO
 Level  1: Call of Module: PRGRUC   in Library: /fzk/inr/rs_aix41/KAPROS/KSLIB
 Level  1: Module: PRGRUC   ended

 Call of Module: PRDUM skipped

      *GO SM=CALCO
 Level  1: Call of Module: CALCO    in Library: /fzk/inr/home/inr067/JOB
 Level  2: Call of Module: GRUCAL   in Library: /fzk/inr/rs_aix41/KAPROS/KSLIB
 Level  2: Module: GRUCAL   ended
 Level  2: Call of Module: DIFFOU   in Library: /fzk/inr/rs_aix41/KAPROS/KSLIB
 Level  2: Module: DIFFOU   ended

   CALCO: keff=3.9328E-01

 Level  1: Module: CALCO    ended
```

The results of this job contained in the file KSUX.inr067.FT07 (reduced) are as follows:

```
*****  REAL   PROBLEM *****

FLUX OF COMPOSITION  1  (NORMALIZED:  SUM OF FLUX = 1)
 1.321328E-03  7.384848E-03  1.461104E-02  2.438733E-02  5.575961E-02
               1.979004E-01  2.294791E-01  1.899359E-01  1.590762E-01
 8.698145E-02  2.599444E-02  5.730472E-03  9.900557E-04  3.554254E-04
               8.237824E-05  9.162600E-06  8.934241E-07  3.477950E-08
 8.956729E-10  5.088124E-11  4.580754E-12  5.894109E-12  1.785415E-12
               3.023754E-13  3.456881E-14  2.675159E-15

K-EFF  =  3.932832E-01
```

9.3 KAPROS-Job Using an Archive

An archive file (for example ARC01) can be generated in the working directory using e.g. the following UNIX-script (the unit of the archive, in this example 30, can be chosen freely):

```
touch ARC01
ln -s ARC01 KSUX.$USER.FT30
ksuxgo archiv.gener
```

where the file archiv.gener (freely chosen file name) contains

```
*KSIOX DBN=INPUT ARCHIV,TYP=CARD,PMN=PRDUM
30  'SEQ' 'GEN'
0
'ARCHIVE FOR BURNUP-CALCULATIONS '
5
'USED FOR FZKA 6280'
*GO SM=ARCHIV
```

It is supposed, that the datablocks GRUCAL and INPUT DIFF0 of the first example have been written into the archive ARC01 on unit 30 by means of the *KSIOX-commands:

```
   *KSIOX DBN=GRUCAL,TYP=ARCO,UNIT=30,FORM=SEQ,SPEC=JOB1
   *KSIOX DBN=INPUT DIFF0,TYP=ARCO,UNIT=30,FORM=SEQ,SPEC=JOB1
```

With the following inputfile for KAPROS the datablock SIGMN will be stored in the archive by the module ARCO immediately after its generation by the module GRUCAL, the datablock FLUX0 will be stored in the archive at the end of the KAPROS-job by the KAPROS-kernel:

```
   *KSIOX DBN=GRUCAL,TYP=ARCI,UNIT=30,FORM=SEQ,SPEC=JOB1
   *KSIOX DBN=INPUT DIFF0,TYP=ARCI,UNIT=30,FORM=SEQ,SPEC=JOB1
   *KSIOX DBN=FLUX0,TYP=ARCO,IND=1,UNIT=30,FORM=SEQ,SPEC=JOB2
   *GO SM=GRUCAL
   *GO SM=ARCO,MPARM=30,'SEQ','SIGMN',1
   *GO SM=DIFF0
   *GO SM=ARCHIV,MPARM=30,'SEQ','LIST'
```

In this example MPARM transfers input parameter to the modules ARCO and ARCHIV, the first component contains the unit-number of the archive.

The following list of contents of the archive, contained in the outputfile KSUX.$USER.FT07, will be produced by the call of the module ARCHIV. The datablock FLUX0 is not yet included, it will be stored to the archive at the end of the job after execution of the module ARCHIV:

```
PRINTOUT OF CONTENTS OF SEQUENTIAL ARCHIVE
  USER          : inr067
  IDENTIFICATION: ARCHIVE FOR BURNUP-CALCULATIONS
  COMMENT       : USED FOR FZKA 6280
==========================================
NR. DBN            INDEX NREC NWORD    SPECIFICATION
-------------------------------------------------------------------------

  1 GRUCAL             1   1     89  inr067  -99.02.22-10:38:29-JOB1
  2 INPUT DIFF0        1   1      6  inr067  -99.02.22-10:38:29-JOB1
  3 SIGMN              1   1    818  inr067  -99.02.22-11:04:04-
```

Acknowledgements

The author wishes to thank several colleagues for their support and in particular Dr. E. Kiefhaber for his incessant efforts in preparing an acceptable edition of this "Introduction to the Use of the UNIX-Version of KAPROS".

Special thanks are due to Dr. G. Buckel for his help to improve this English report and to Mr. D. Thiem for his helpful comments and the following discussions, which caused a improved insight of the author to some details of KAPROS.

References

[1a] G. Buckel, W. Höbel:
Das Karlsruher Programmsystem KAPROS
Teil I: Übersicht und Vereinbarungen
KFK 2253 (1976)

[1b] H. Bachmann, S. Kleinheins:
Das Karlsruher Programmsystem KAPROS
Teil Ia: Kurzes KAPROS-Benutzerhandbuch
KFK 2317 (1976)

[1c] H. Bachmann, S. Kleinheins:
Das Karlsruher Programmsystem KAPROS
Teil II: Dokumentation des Systemkerns
KFK 2254 (1976)

[2] N. Moritz:
Die FORTRAN-77 Version des Karlsruher Programmsystems KAPROS
KFK 3860 (1985)

[3] W. Höbel et al.:
KAPROS-SUBSYSTEM-KERN KSSK
private communication (1987)

[4] C.H.M. Broeders:
Development of the KAPROS-Versions KSSKBU and KSSKXA
private communication (1993)

[5] J. Braun, D. Woll:
Einführung in Arbeitsweise und Benutzung von KAPROS3
private communication (1991)

[6] J. Braun:
Development of an XA-Version of KAPROS3
private communication (1993)

[7] C.H.M. Broeders:
Entwicklungsarbeiten für die neutronenphysikalische Auslegung
von Fortschrittlichen Druckwasserreaktoren (FDWR)
mit kompakten Dreiecksgittern in hexagonalen Brennelementen
KfK 5072 (1992)

[8] C.H.M. Broeders:
Development of the KAPROS-Version KSSKUX
private communication (1995)

Survey of available KAPROS-Modules[1]

Determination of Atomic Number Densities for Material Compositions

Module	Purpose
GRUMIX	Modifying the material names, densities and temperatures of material compositions in MISCH-structure and preparing input for GRUCAL
MIMI	Blending mixtures
NDCALC	Determining number densities and cell geometry data
NDWIMS	e.g. for GRUCEL, included in GRUCAL
SIMI	Replacing materials not yet included on a GRUBA-file by other ones
TCAL	Calculation of atomic number densities

Calculation and Manipulation of Multigroup Cross-Sections

Module	Purpose
CHICOR	Iterative improvement of isotope-averaged neutron fission spectra
COLLUP	Group collapsing SIGMN-datablocks including upscattering
COLRAB	Collapsing group cross-sections without upscattering using real, adjoint and bilinear flux weighting
ENERGY	Energy group boundary information of group constant libraries
GRUCAL	Calculation of microscopic and macroscopic group cross-sections in SIGMN-structure; Heterogeneity correction for a homogenized reactor zone (GRUCAH); Heterogeneity correction for lattices of reactor cells (GRUCEL)
ONEHOM	Determination of cell-averaged effective group constants
REMCOR	Sigma-removal correction for upscattering
RESABK	Improved calculation of group constants in the resonance energy range
SIGMNC	Collapsing group cross-sections without upscattering using conventional real flux weighting
SIGMUT	Creating and Modification of datablocks in SIGMN-structure
TRANSX	Provision of self-shielded group-constants from SIGMN-datablocks for computer codes like DIAMANT or codes using the Los Alamos cross-section format like DANTSYS (including e.g. ONEDANT and TWODANT)

[1]Affected by the adaptation of the modules from MVS to UNIX it might be possible, that not all listed modules are operational, but the some important modules are running.

Cell Calculations and/or heterogeneity correction

Module	Purpose
KAPER4	Calculation of unit-cells of fast reactors
RATES	Superposition of a fine structure of reaction rates calculated by KAPER4 by a global neutron flux distribution
WEKCPM	Onedimensional collision probability code based on WIMS using the interface files prepared by WEFILE

Neutron Diffusion Codes

Module	Purpose
CHECK1	Checks adequacy of input mesh sizes with respect to diffusion length and mean free path composition- and group-wise (1- and 2-dimensional)
DIFF0U	Solution of the zero-dimensional multigroup-diffusion equation (fundamental mode with buckling)
DIF1D	1-dim. multigroup diffusion code
DIXCON	Accelerating multigroup DIXY-calculations
DIXY2	Solution of the multigroup diffusion equation in xy- or rz- or r-theta-geometry, evaluation of neutron flux distributions, integral and/or local reaction rates and/or densities, application of first order or "exact" perturbation theory and gamma-sources at the spot.
D3DG D3DR D3EG D3ER	Branches of a FORTRAN Program for the Solution of the Stationary Three-Dimensional Multigroup Neutron Diffusion Equations in Rectangular, Cylindrical and Triangular Geometry

Neutron Transport-Codes

Module	Purpose
HEXNOD	Nodal diffusion and transport code in (hex,z)-geometry
ONETRA	Onedimensional Sn-code It is recommended to use the more sophisticated neutron transport code ONEDANT[2]

[2]A list of the most important stand-alone codes for fission- and fusion-application is available at the Forschungszentrum.

Iterative Determination of Buckling and Fuel Enrichment and Evaluation of Neutron Flux Distributions

Module	Purpose
BUCITU	Buckling calculation by iterations of 0-dimensional diffusion neutron flux calculations allowing neutron-upscattering
IBUCK0	Buckling calculation by iterations of 0-dimensional diffusion neutron flux calculations
IBUCK1	Buckling calculation by iterations of 1-dimensional diffusion neutron flux calculations
YIT0	Adaption of fuel enrichment to a given criticality value by 0-dimensional diffusion neutron flux calculations
YIT1	Adaption of fuel enrichment to a given criticality value by 1-dimensional diffusion neutron flux calculations

Evaluations, i.e. Reaction Rates and/or Combinations of them

Module	Purpose
AUDI3	Evaluation of 3-dimensional neutron flux distributions and application of first order or "exact" perturbation theory
AUTOBU	Determination of pin powers in specified fuel elements
BILANZ	Calculation of zone- and energy dependent macroscopic reaction rates and balances
DXEVA2	Evaluation of 2-dimensional neutron flux distributions
RATKOM	Combining of reaction rates (calculated by RAT1)
RAT1	Calculation of 1-dimensional reaction rates

Reactor Kinetics Parameter, based on Perturbation Theory

Module	Purpose
AUDI3	Application of first order or "exact" perturbation theory in 3 dimensions (diffusion theory)
BETA1	Calculation of 1-dimensional beta-eff values
DIXDYN	Calculation of reactor dynamics parameters
DXPRT2	Application of first order or "exact" perturbation theory in 2 dimensions (diffusion theory)
LAMBDA	Calculation of the effective decay constant for delayed neutrons
LIFET1	Calculation of the neutron generation time by 1-dimensional neutron flux calculations
PERT1	Perturbation calculations in 1-dimensional neutron flux calculations (diffusion theory)
DSIGDT	Calculation of cross section temperature derivatives
DOPPL1	Calculation of Doppler Coefficients

Burnup and Depletion Calculations

Module	Purpose
ARCOSI	Advanced Reactor COre SImulator [7]
HXSLIB	Creation of ARCOSI libraries
BURNUP	Numerical solution of the burnup equation, Group collapsing with upscattering
BURN0D	0-dimensional burnup calculations with DIFF0U and BURNUP
DXBURN	Burnup calculations using DIXY and BURNUP
EVAHEX	Determination of significant nuclear parameters of a reactor and characteristic quantities for its burnup behavior from HEXABU-results
HEXABU	Burnup calculations with output of macroscopic cross-sections in SIGMN-structure for burnt-up compositions (mainly for fast reactor applications)
KARBUS	Procedure for whole core burnup calculations including KORIGEN [7][3]
KORINT	Preparation of input data for KORIGEN
SIG2EV	Creation of burnup dependent KORIGEN libraries
MIXMAN	Simulation of fuel management during burnup calculations

Plot-Modules[4]

Module	Purpose
PLFLUX	Presentation of energy dependent spectra stored in FLUX0 structur
PLOTKS	PLOTEASY plotting module for KAPROS
PLOT1V	Presentation of results given in multidimensional data arrays depending from energy and space as twodimensional data depending from energy or space
PLO3D	Presentation of 3-dimensional perspective plots
QUAPLO	Presentation of reactor cross-sectional views and contour lines in rectangular geometry
TRIPLO	Presentation of reactor cross-sectional views and contour lines in triangular geometry

[3]Several other burnup related modules are described in some detail in Ref.[7] and in online documentations.

[4]For the presentation of results new plot software, e.g. TECPLOT©, will be applied, therefore no effort was devoted to adapting the corresponding MVS-modules to UNIX. But in principle they are running but need to be tested.

Auxiliary Modules for Manipulating Datablocks

Module	Purpose
ARCHIV	Generating of archives, listing of contents of archives, selecting or printing of datablocks of archives
ARCO	Archiving a KAPROS-datablocks during execution ot the job
DELDB	Deleting of KAPROS-datablocks
KOPDB	Combining several KAPROS-datablocks of the same structure into a single one
PRINDB	Printing of KAPROS-datablocks with data-depending format
RENDB	Changing name and/or index of KAPROS-datablocks
UTKS	Transferring of data between KAPROS-lifeline and external storage devices