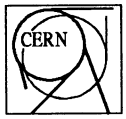
ALEPH 89-11
DATACQ 89-2
30th JANUARY 1989

A.Aimar University of Torino
J.Harvey RAL/CERN-EF
M.Lubich University of Innsbruck
G.Waltermann MPI/Munich



ORGANISATION EUROPÉENNE POUR LA RECHERCHE NUCLÉAIRE
EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
Laboratoire Européen pour la Physique des Particules
European Laboratory for Particle Physics

G P H - User Guide

Version 1.1

Abstract

Hierarchical Graphics Package

- Structuring and storing Graphical Data on a Database
- Retrieving Data from the Database
- Drawing and Pick facilities

GPH is a software-package to simplify an hierachical structure of graphical data, storing the data on a 'graphical database', retrieve easily the information to be manipulated (rotated, shifted, scaled) and draw on the screen. A 'Pick' facility is provided. Usage and the problems with various Graphics Systems are hidden away from the GPH user.

Contents

1 Introduction

The *'Basic Ideas'* to the *'Structure'* of GPH originally came from John Harvey. Executed into a completely working, usable package on top of *GKS* by Alberto Aimar - who also did the great job of really typing most of the code in its basic form.
It's up to you now to work with the manual, on your application and possibly find all these little bugs still crawling around within the package..... *(Thanks!)*.
New ideas, approvements and complaints are wellcome! Please send mail to AL1W00::WALTERMANN.

1.1 WHAT IS GPH?

GPH - an *'Hierarchical Graphics Package'* consists of two basic elements:

a) *GRAPHICAL DATABASE* (DB):

where descriptions of any kind of object are kept in terms of graphics primitives (like polylines, polymarkers, text etc..) commonly used by standard graphics packages (like GKS, UIS etc..) and following an hierarchical structure to be non-redundant and very modular to construct any kind of complicated setup.

b) *SUBROUTINE LIBRARY* :

consisting of routines to:

- build the graphical database
- retrieve data from the graphical database
- draw to the screen
- define or modify graphical aspects
- interact with the display (pick)

Please see the chapter 2 and the 'GPH REFERENCE MANUAL' for all the details about its usage.

1.2 BASIC IDEAS

GPH has been created to enable you to draw even complex graphics without knowing anything about basic graphic-packages like *GKS* or *UIS*. The most important you really have to think about is to divide your graphics into several small basic structures called *'ICONs'*. Once these Icons are defined you can start to 'play' with them by using

transformations like scaling, rotation, shifting and duplicating and continue constructing like children use their *'LEGO'*.

This scheme describes in principle almost all graphics one is able to produce by the relation of its parts. The central part called *'GraphOBJ'* is any kind of drawing composed by one or more Primitives. The parameters of these primitives are separately stored in the primitive-coordinate tables (Text, Polyline, Polymarker, Circle, Fillarea). They are separated because different primitives have different formats of coordinates. These primitives describe the technical part of a graphical object.

The scheme allows you then to built a more sophisticated structure based on these graphical objects. First you specify some primitives with corresponding coordinates and declare that graphical object as an *ICON*. The Icons are the basic parts of drawing graphics in GPH. (The construction of these Icons is the one and only time the user has to think about the x,y,z-coordinates of its primitives!) An Icon can be e.g. a TPC-sector, a box or similiar.

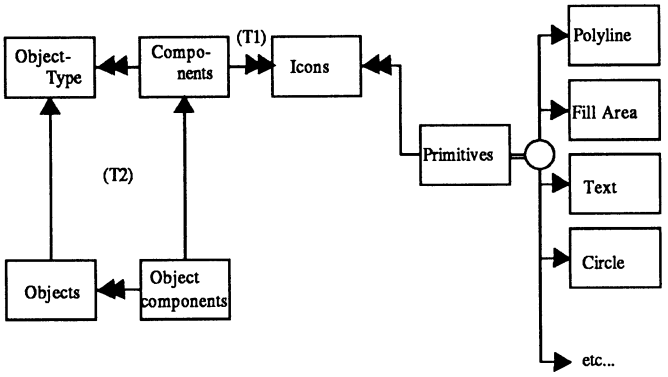
Once having defined a set of *Icons* the user can go one step further : he can now define an *ObjectType*. An ObjectType is in principle similiar to an Icon but has usually more complexity. It is constructed out of predefined Icons which have gone through a *'first level transformation'* (rotation, shift and scale) and are then called *Components* of the ObjectType.

So in fact an *ObjectType* represents the basic structure of any final *Objects* of the same family (e.g. one TPC-endplate) and is built out of several (or only one) *Components* related to exactly one basic Icon each, represented in its first transformation. The ObjectType also usually has to carry all the information of this special type of object in the sense of graphical aspects like linestyle, colour etc.

The user is now able to build out of his several ObjectTypes - by using the same kind of transformations mentioned above - *Objects* in different instantiations , which are therefore composed of the same components like its mother ObjectType but gone through the *'second level transformations'* .

Maybe a diagram is easier to read:

Layout of Graphics Database



(T1 = first level transformation
T2 = second level transformation)

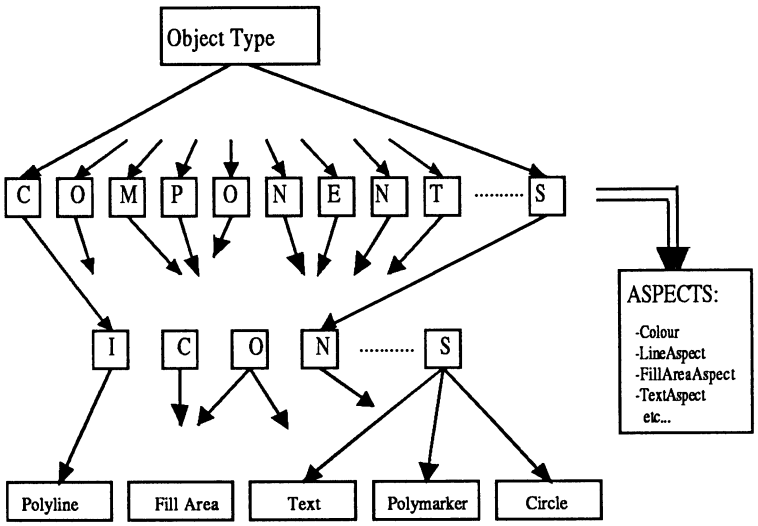
The scheme also shows that there are more data related with a single graphical object : MarkerAspect, LineAspect, FAreaAspect, TextAspect and Colour. These tables define for each graphical object the different aspects; so it is possible to draw e.g. a 'broken' TPC-sector in a different colour or linestyle..

2 USER GUIDE

2.1 STRUCTURE OF GRAPHICAL DATABASE

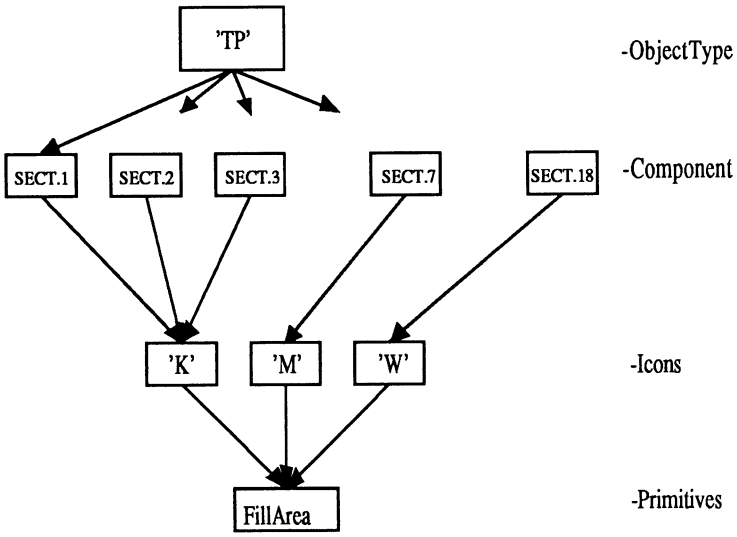
Everybody working with 'graphics' wants to read and learn by looking at a drawing - so rather than words a layout to explain the structure of the database:

STRUCTURE OF DATABASE



Defining the basic parts of the DB - the *'ICONs'* - is the only time you have to think about x,y,z-coordinates and graphics primitives. Then you scratch your head about the 'first level transformation' (rotation,scaling,shifting) and the beauty of the 'aspects' (colour, linestyle etc.) of these Icons and call them *'COMPONENTs'*. From these you can construct your *'OBJECT-TYPEs'*, the major parts of the DB.
Just in case you are familiar with the ALEPH-DETECTOR an example using 'real world terminology' could help to understand a bit better:

EXAMPLE TPC



Right?
And since it's quite a job to think about this structure and set up the database it is the most logical that you:

- *Build it only ONCE !*
- *Use it MANY TIMES !*

2.2 PROGRAMMING WITH GPH

2.2.1 BUILDING OF A GRAPHICAL DATABASE

The following shows the basic sequence of routines needed to build a DB. You should just take care of mapping to a global section holding then the database (please see the related example app. A.1 and routine descriptions).

```
C....initialize the package
GPH_INIT( )

C....map to the global section to be created
GPH_CREATE_GLOBAL(file_name,section_name)

C....define first icon built out of polyline and fillarea
GPH_OPEN_ICON('Icon_Name1',ICON_ID1)
—GPH_INSERT_POLYLINE(N,X,Y,Z)
—GPH_INSERT_FILLAREA(N,X,Y,Z)
—etc.
GPH_CLOSE_ICON( )

C....define second icon built out of circle
GPH_OPEN_ICON('Icon_Name2',ICON_ID2)
—GPH_INSERT_POLYLINE(N,X,Y,Z)
—GPH_INSERT_CIRCLE(XC,YC,ZC,RAD,ANG1,ANG2)
—etc.
GPH_CLOSE_ICON( )


C....construct first ObjectType out of icon1 and icon2
C....using the same transformations (T(1))
GPH_OPEN_OBJTYPE('ObyType_Name1',OBJTYPE_ID1)
—GPH_CREATE_COMPONENT(ICON_ID1,T(1),'CompName',COMP_ID1)
—GPH_CREATE_COMPONENT(ICON_ID2,T(1),'CompName',COMP_ID2)
—etc.
GPH_CLOSE_OBJTYPE( )

C....construct second ObjectType out of icon1 using different
C....transformations
GPH_OPEN_OBJTYPE('ObyType_Name2',OBJTYPE_ID2)
—GPH_CREATE_COMPONENT(ICON_ID1,T(2),'CompName',COMP_ID3)
—GPH_CREATE_COMPONENT(ICON_ID1,T(3),'CompName',COMP_ID4)
—etc.
GPH_CLOSE_OBJTYPE( )


etc...
```

C....close properly the package...
GPH_CLOSE( )

(P.S. - T(x) = transformation: shift, rotate, scale)

2.2.2 RETRIEVING DATA FROM DATABASE

Once the DB exists you should map to the global section file and you can retrieve information on graphics primitives to go on working (the only thing you should remember from creating the DB are the names you gave to icons and ObjectTypes !)

C....initialize the package...
GPH_INIT( )

C....map to the global section holding the DataBase
GPH_MAP_GLOBAL(file_name,section_name)

C....define the data by an internal IDentifier
GPH_GET_ICONID('Icon_Name2',ICON_ID)
GPH_GET_OBJTYPEID('ObjType_Name1',OBJTYPE_ID)

C....retrieve the information on the primitives
GPH_GETPRIM_ICON(ICON_ID,NPRIM,NPT,X,Y,Z)
GPH_GETPRIM_OBJTYPE(OBJTYPE_ID,NPRIM,NPT,X,Y,Z)

etc..

Where 'nprim' is the number of primitives, 'npt' the number of points per primitive and 'x,y,z' the arrays with all coordinates.

For people interested on drawing the *ALEPH-DETECTOR* (or parts of it) a 'Graphical Database' describing the detector elements like TPC, ECAL, HCAL etc. exists already. It is derived from the official *ALEPH-DATABASE* (ADBS.DAT) and is called

- A_GRAPH$DB:GPH_DETECTOR.DAT

The ObjectType names usually define the detector parts down to their 'subcomponent' level and the icons down to their 'slot' level. Sorry for exceptions, there might be reasons for it.
There is a special routine in GPH to retrieve the primitives describing the detector elements without knowing their lower level construction on

the DB. The first argument is a character string describing the detector elements and should follow the ALEPH naming convention (be careful, this in the process of changing, the GPH package will follow!) :

```
GPH_INIT( )

C....map to the global section describing the ALEPH detector
GPH_MAP_GLOBAL('A_GRAPH$DB:GPH_DETECTOR.DAT','MY_COPY)

C....retrieve the primitives of a slot:
GPH_GETPRIM_DETELEMENT('EC_EA_12',NPRIM,NPT,X,Y,Z)

C....retrieve the primitives of a subcomponent:
GPH_GETPRIM_DETELEMENT('HC_BL',NPRIM,NPT,X,Y,Z)
```

(P.S. Please see example in app. A.2)

2.2.3 DRAWING ROUTINES

When you want to use the drawing routines within GPH you should think of the 'second' transformation of an ObjectType (defining the final instantation) and declare it as an 'OBJECT' to be drawn.
The setup of the screen with its graphics window(s) also comes into place. GPH makes a graphics window to appear on the screen by the call to 'GPH_OPEN_OUTPUT' (for size and location please read the particular section within 'Implementation Details' chapter 2.3) and defines the 'world coordinate system' with the call to 'GPH_OPEN_WINDOW'. (Sorry about the confusion within the routine names. This problem is graphics world wide!)

```
C....initialize the package
GPH_INIT( )

C....map to the existing global section
GPH_MAP_GLOBAL(file_name,section_name)

C....create a window with its world coordinates on the screen
GPH_OPEN_OUTPUT(wtype,'title',connID,wkID)
GPH_OPEN_WINDOW(wkID,WX1,WX2,WY1,WY2)

C....define the data by an internal IDentifier
GPH_GET_OBJTYPEID('ObjType_Name',OBJTYPE_ID)

C....set graphics aspects
GPH_SET_COLOUR(colour_ID)
GPH_UPDATECOLOUR_OBJTYPE(OBJTYPE_ID,EVERY)
```

```
C....and create the final object (transformation possible!)
C....with an internal IDentifier
GPH_CREATE_OBJECT(OBJTYPE_ID,T(x),'Object_Name',OBJECT_ID)

C....draw the object to the screen
GPH_DRAW_OBJECT(OBJECT_ID)
.
.
.
GPH_CLOSE( )
```

(P.S.
- WX1,WX2,WY1.WY2 = world-coordinate-system
- EVERY = set to -1 means all components of the ObjectType
- T(x) = transformation: shift, rotate, scale)

2.2.4 INTERACTION WITH THE DISPLAY

As soon as the picture is drawn the interaction with the display can start: a 'Pick-facility' is implemented in it's most simple form which returns (after you have 'picked' an item with the mouse) the internal Object-ID and Component-ID. There are other routines where you can change all sorts of graphical aspects like linestyle, colour, fillpattern etc. and redraw an Object or its parts (Components).

```
.
.
.
C....pick on an object and find it's internal ID's
GPH_PICK_OBJECT(Object_ID,Comp_ID)

C....i.e. change some graphics aspect
GPH_SET_COLOUR(colour_ID)
GPH_UPDATECOLOUR_OBJECT(Object_ID,Comp_ID)

GPG_SET_FILLASP(fillaspect_ID)
GPH_UPDATEFASP_OBJECT(OBJECT_ID,EVERY)

C....redraw in the new style...
GPH_DRAW_COMPONENT(OBJECT_ID,COMP_ID)
.
.
.
```

(P.S. EVERY = set to -1 means all components of the object)

Please see the Reference Manual for the complete list of routines available.....

2.3 IMPLEMENTATION DETAILS

The very important part of GPH - the *DATAMANAGEMENT* - is handled by VAX-Structure.
The file containing the *GRAPHICAL DATABASE* being created or read is mapped to the program as a *GLOBAL SECTION*.
The *ERROR-messages* are produced by calls to VAX-VMS LIB$SIGNAL.
So - at the moment - the package is restricted to VAX-computers!!!!
(But be optimistic. It's the big project for version 1.2 of the package to be 'transportable'!)

The *GRAPHICAL PART* of GPH at the time has two lower level interfaces:
- to GKS (the Graphics Kernel Standard)
- to UIS (Graphics System used at VAX-Workstations)
but can be changed quite easily to be on top of another system.

SEGMENTATION and *PICKING* is done on the level of drawn Objects and their Components.
Warning: within the UIS-Version the 'segmentation' is not really made for being 'picked' with the 'mouse'. Be careful when picking on an area or overlapping lines.

MULTIPLE WINDOWING on the same workstation is possible but restricted (by GKS/GTS-Gral-Version for VAX-Workstations) to a maximum of 6 windows simultaneously.
Therefore a file *WINDOW.DAT* has to exist within the working directory when you use the graphics facilities of GPH. It defines (in metres !!!) the *size* (length in X, length in Y), the *position on the screen* (lower left corner X/Y) plus the *title* for all required windows openend (read in by GPH_OPEN_WINDOW). The format is : (4F5.3,A20) and the sequence is corresponding the workstation-type-numbers (8601 to 8606).

Example:
0.15 0.15 0.02 0.02 Title for WKTYP 8601
0.10 0.10 0.10 0.10 Title for WKTYP 8602
0.05 0.25 0.20 0.00 Title for WKTYP 8603
etc.

Creation of *METAFILES* is provided. But since metafiles are produced in very different ways within the various graphics systems more than one GPH routine is necessary. Please see the related routines descriptions.

Using *UPI* (the Online-MENU-package) as a menu-driven steering-facility for GPH is highly recommended for all GPH-applications wanting to talk to other programs running within the ALEPH-world (via the *SWITCHER and SCHEDULER*).
There is just one warning: To run UPI on top of the GKS-version of GPH - you have to use the old version of UPI for the time being! (Please consult the UPI-specialists for more details....)

2.4 Installation and Use

To make life easier - logical names are defined (and please get sure they will be defined at new installation !) :

A_GRAPH$SRC - for all *SOURCE* and *INCLUDE* files
= DISK$COMMON:[ONLINE.ONLDB.GRAPHICS.SOURCE] on ALONL
= DISK$ONLINE:[ONLINE.GPH.SOURCE] on AL1W00

A_GRAPH$DIR - for all *OBJECT* files and *LIBRARIES*
= DISK$COMMON:[ONLINE.ONLDB.GRAPHICS.NODEB] on ALONL
= DISK$ONLINE:[ONLINE.GPH.NODEB] on AL1W00

A_GRAPH$DB - for *DataBases*
= DISK$COMMON:[ONLINE.ONLDB.GRAPHICS.DBASE] on ALONL
= DISK$ONLINE:[ONLINE.GPH.DBASE] on AL1W00

The *GPH source* is a *CMS*-LIBRARY ON:
ALONL::DISK$COMMON:[ONLINE.ONLDB.GRAPHICS.CMS]

The *GPH libraries* are called:
A_GRAPH$DIR:GPH_GKS.olb for the GKS-Version
A_GRAPH$DIR:GPH_UIS.olb for the UIS-Version
and the *OPT*-file:
A_GRAPH$DIR:GPH.OPT

The existing DataBase describing the *ALEPH-Detector* :
A_GRAPH$DB:GPH_DETECTOR.DAT

The *LINK* should be:

```
$ LINK/EXE=myprog   myprog,-
  a_graph$dir:GPH_MESSAGE,GPI_MESSAGE,-
              GBLSECUFO,CREATE_GLOBAL,-
              GPH_UIS/LIB,GPH/OPT,-
              'CERN$LIBS'
```

The mentioned *EXAMPLES* you will find under:
A_GRAPH$SRC:EX_BUILD.FOR
A_GRAPH$SRC:EX_GET.FOR
A_GRAPH$SRC:EX_DRAW.FOR
The command file to link:
A_GRAPH$DIR:EX.COM

Using *GPH with UPI to draw the ALEPH-DETECTOR* :
an example to draw all parts of the detector existing on the GPH_DETECTOR.DAT - database on different windows on the screen and controlled by UPI-menus - you will find on:
AL1W00::DISK$USER:[WALTERMANN.GPH] or
ALONL::DISK$ONLINE:[WALTERMANN.GPH]
(with link-files and an example for *window.dat*!)

A EXAMPLES

A.1 HOW TO BUILD A DATABASE

```
C=====================================================================
C
      PROGRAM ECAL_BUILD
C
C---------------------------------------------------------------------
C    Example to build Database for ECAL Barrel and Endcap A
C---------------------------------------------------------------------
C Declarations.
C -
      IMPLICIT NONE

      INCLUDE 'A_GRAPH$SRC:ADBS.VLB'
      INCLUDE 'A_GRAPH$SRC:GPH.VLB'

      REAL    X(100),Y(100),Z(100),R
      REAL ROT(2)
      REAL NOSCALE(3) /1.,1.,1./
      REAL NOSHIFT(3) /0.,0.,0./
      INTEGER N,LP1
      INTEGER SC_ID,SL_ID
      INTEGER BL_ICON,      EA_ICON
      INTEGER EC_BL_OT,     EC_EA_OT
      INTEGER ECB_SLOT(12), ECE_SLOT(12)
C
      CHARACTER*16 VOLNAM(3)
      INTEGER LEPLAN,LECON, LEFACE
      INTEGER PTRFCE (10)
      REAL PLANES(4,12)
c
      CHARACTER*2 MOD_NAME(12) /'01','02','03','04','05','06',
     &                          '07','08','09','10','11','12'/

      DATA VOLNAM/'E external','B external','E external'/

C + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C Entry point.
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
C     Map the Graphical-Database to be created:
C
      CALL GPH_CREATE_GLOBAL
     &         ('DISK$COMMON:[WALTERMANN.DBASE]MY_DETECTOR.DAT',
     &          'ECAL_GBL')
C
C     Initialize GPH and map to Aleph-Database
C
      CALL GPH_INIT ()
      CALL ADBS_INIT ()
C
C     Create the ECAL-barrel-Icon
C
      SC_ID = 2
      SL_ID = 1
      CALL EVOLPL(VOLNAM(SC_ID),SC_ID,SL_ID,LEPLAN,PLANES)
      CALL EVOLCT(VOLNAM(SC_ID),LEPLAN,PLANES,N,X,Y,Z)
C
      CALL GPH_SET_COLOUR(6)
```

```
      CALL GPH_OPEN_ICON ('EC_BL_MOD', BL_ICON)
        CALL GPH_INSERT_FILLAREA (N,X,Y,Z)
      CALL GPH_CLOSE_ICON ()
C
C     Now the Icon for ECAL-endcap A
C
      SC_ID = 1
      SL_ID = 1

      CALL EVOLPL(VOLNAM(SC_ID),SC_ID,SL_ID,LEPLAN,PLANES)
      CALL EVOLCT(VOLNAM(SC_ID),LEPLAN,PLANES,N,X,Y,Z)

      CALL GPH_SET_COLOUR(6)

      CALL GPH_OPEN_ICON ('EC_EA_MOD', EA_ICON)
        CALL GPH_INSERT_FILLAREA (N,X,Y,Z)
      CALL GPH_CLOSE_ICON ()
C
C     Define the Barrel-ObjectType
C
      CALL GPH_OPEN_OBJTYPE ('EC_BL',EC_BL_OT)

      ROT(1) = 0.
      DO LP1 = 1,12
        ROT(2) = -(LP1 -1)*30.
        CALL GPH_CREATE_COMPONENT (BL_ICON,
     &                             NOSHIFT,ROT,NOSCALE,
     &                             MOD_NAME(LP1),
     &                             ECB_SLOT(LP1))
      END DO

      CALL GPH_CLOSE_OBJTYPE ()

C
C     Define the Endcap A-ObjectType
C
      CALL GPH_OPEN_OBJTYPE ('EC_EA',EC_EA_OT)

      ROT(1) = 0.
      DO LP1 = 1,12
        ROT(2) = -(LP1 -1)*30.
        CALL GPH_CREATE_COMPONENT (EC_ICON,
     &                             NOSHIFT,ROT,NOSCALE,
     &                             MOD_NAME(LP1),
     &                             ECE_SLOT(LP1))
      END DO

      CALL GPH_CLOSE_OBJTYPE ()
C
      CALL GPH_CLOSE()
C
      END
```

A.2 HOW TO RETRIEVE GRAPHICAL INFORMATION

```
C=======================================================================
C
      PROGRAM GPH_GET
C
C-----------------------------------------------------------------------
C     Example to retrieve Primitives for TPC-sector E08 Endplate B
C-----------------------------------------------------------------------
C Declarations.
C -
      IMPLICIT NONE

      INCLUDE 'a_graph$src:GPH.VLB'
      INCLUDE 'a_graph$src:GPH_UIS.INC'

      INTEGER STATUS
      INTEGER NPRIM,NPT(10)
      REAL X(100),Y(100),Z(100)
C
      CHARACTER*80 FILE_NAME
      CHARACTER*16 SECTION_NAME
C
      INTEGER I,NXYZ
      INTEGER WKID,CONID
      INTEGER TP_ID,  TP_E08_ID
C
C + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C Entry point.
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
C--Map to Graphical-Database (Global Section)
C
      FILE_NAME = 'A_GRAPH$DB:GPH_DETECTOR.DAT'
      SECTION_NAME = 'MY_GBL'
      CALL GPH_MAP_GLOBAL (FILE_NAME,SECTION_NAME)
C
C--Initialize GPH and Open Window on Screen
C
      CALL GPH_INIT ()
C
        WKID=1
        CONID=1
        CALL GPH_OPEN_OUTPUT(8601,'TEST_GPH_GET',CONID,WKID)
        CALL GPH_OPEN_WINDOW(1,-500.,500.,-500.,500.)
C
C--Get Primitives
C
        CALL GPH_GETPRIM_DETELEMENT('TP_B_E08',NPRIM,NPT,X,Y,Z)
C
C--which is equivalent to:
C      CALL GPH_GET_OBJTYPEID ('TP_B',  TP_ID)
C      CALL GPH_GET_COMPONENTID ( TP_ID, 'E08', TP_E08_ID)
C      CALL GPH_GETPRIM_OTCOMP ( TP_ID, TP_E08_ID, NPRIM, NPT,X,Y,Z)
C
C--use the 'polylines' i.e. to draw in your own way....
C
      NXYZ=1
      DO I=1,NPRIM
        CALL UIS$PLOT_ARRAY(ID_DISP,0,NPT(I),X(NXYZ),Y(NXYZ))
        NXYZ=NXYZ+NPT(I)
      ENDDO
C
        CALL GPH_CLOSE()
      STOP
      END
```

A.3 HOW TO DRAW and PICK

```
C=======================================================================
C
      PROGRAM ECAL_DRAW
C
C-----------------------------------------------------------------------
C   Example to use the full Graphics-facilities of GPH
C-----------------------------------------------------------------------
C Declarations.
C -
      IMPLICIT NONE

      INCLUDE 'A_GRAPH$SRC:GPH.VLB'

      INTEGER STATUS
      INTEGER CONID,WKID
C
      REAL NOROT(2) /0.,0./
      REAL NOSCALE(3) /1.,1.,1./
      REAL NOSHIFT(3) /0.,0.,0./

      INTEGER ECBL_OT, ECEA_OT
      INTEGER OB_EA_ID,OB_BL_ID
      INTEGER OBJECT_PICK,COMP_PICK
      INTEGER LINE_ID_SOLID,LINE_ID_DASHED
      LOGICAL COLOURS
C
      CHARACTER*80 FILE_NAME
      CHARACTER*16 SECTION_NAME
C
C + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
C Entry point.
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
C--Map to existing Graphical Database
C
      FILE_NAME = 'A_GRAPH$DB:GPH_DETECTOR.DAT'
      SECTION_NAME = 'MY_GBL'
      CALL GPH_MAP_GLOBAL (FILE_NAME,SECTION_NAME)
C
C--Initialize
C
      CALL GPH_INIT ()
      CALL GPH_INQUIRE_COLOUR_SETUP(8601,COLOURS)
C
      IF(COLOURS) CALL GPH_SET_BORDER(.TRUE.,5)
C
      CALL GPH_PUT_LINEASP(1,1.,LINE_ID_SOLID)
      CALL GPH_PUT_LINEASP(2,1.,LINE_ID_DASHED)
      CALL GPH_SET_LINEASP(LINE_ID_SOLID)
C
C--Define Objects
C
      CALL GPH_GET_OBJTYPEID ('EC_BL', ECBL_OT)
      CALL GPH_GET_OBJTYPEID ('EC_EA',  ECEA_OT)
C
      CALL GPH_CREATE_OBJECT ( ECBL_OT, NOSHIFT, NOROT, NOSCALE,
     &                         'EC_BL', OB_BL_ID)
      CALL GPH_CREATE_OBJECT ( ECEA_OT, NOSHIFT, NOROT, NOSCALE,
     &                         'EC_EA', OB_EA_ID)
C
C--Open Windows and Draw Objects
C
        WKID=1
        CONID=1
        CALL GPH_OPEN_OUTPUT(8601,'ECAL-BARREL',CONID,WKID)
        CALL GPH_OPEN_WINDOW(1,-300.,300.,-300.,300.)
C
        CALL GPH_DRAW_OBJECT (OB_BL_ID)
```

```
C
        WKID=2
        CONID=2
        CALL GPH_OPEN_OUTPUT(8602,'ECAL-ENDCAP',CONID,WKID)
        CALL GPH_OPEN_WINDOW(2,-300.,300.,-300.,300.)
C
        CALL GPH_DRAW_OBJECT (OB_EA_ID)
C
C--PICK on Component and redraw
C   (at pick outside the object - change window or go on)
C
        WKID=1
  1     CALL GPH_PICK_OBJECT (WKID, STATUS, OBJECT_PICK,COMP_PICK)
        IF (STATUS.EQ.0 .OR. OBJECT_PICK.EQ.0) THEN
          IF (WKID.EQ.2) GO TO 9
          WKID=2
          GO TO 1
        ENDIF
C
        IF (COLOURS) THEN
          CALL GPH_SET_COLOUR(4)
          CALL GPH_UPDATECOLOUR_OBJECT(OBJECT_PICK,COMP_PICK)
        ELSE
          CALL GPH_SET_LINEASP(LINE_ID_DASHED)
          CALL GPH_UPDATELASP_OBJECT(OBJECT_PICK,COMP_PICK)
        ENDIF
C
        CALL GPH_DRAW_COMPONENT(OBJECT_PICK,COMP_PICK)
        GO TO 1
C
C--Create METAFILE (UIS version)...
C
  9     CONTINUE
        CALL GPH_WRITE_METAFILE('META1.DAT',WKID)
C
C--End
C
        CALL GPH_CLOSE()
      STOP
      END
```

That's it!

When you like it and want to use it get the

- 'GPH REFERENCE MANUAL'
- ALEPH 89-11
- DATACQ 89-3

YOU SHOULD NEED IT !