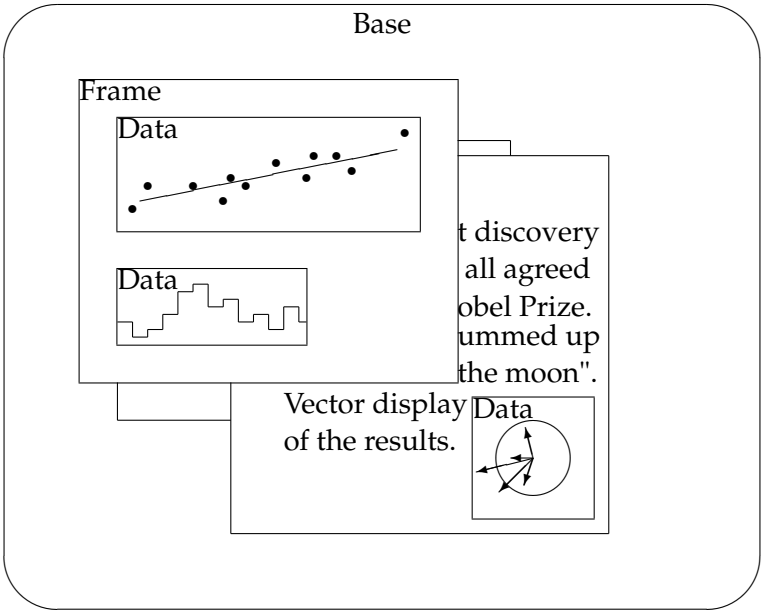
SUN/48.8

Starlink Project
Starlink User Note 48.8

Nicholas Eaton
Brian McIlwrath

24 April 2013

AGI — Applications Graphics Interface Library
Version 2.2
Programmer's Manual



Abstract

AGI is a graphics database system that can be used to store information about the size and position of a plot on a graphics device. This document explains how to interact with this database using the AGI library.

Contents

1 Introduction

AGI is a graphics database system. It can be used to store information about the size and position of a plot on a graphics device. This can be used by subsequent applications to find out where the plot has been drawn and how the coordinate system maps onto the plotting area.

One obvious use of the database is to allow an application to overlay its output on top of that produced by a different application, for instance plotting radio contours onto an optical image. The first application draws its image and informs AGI the size and position of the plot. The second application uses AGI to set up its graphics coordinate system to match that of the first application so that the contour map appears in the correct position on the screen.

Another common use is to allow an application to put up an interactive cursor to read off positions from a previously drawn image.

Although designed for use within ADAM, AGI can also be used in non-ADAM applications.

2 Brief Description

The AGI database system implementation has changed in the current version. It now exists as two subroutine libraries – one, which is unchanged from earlier versions, to handle all graphics interactions where GKS is the underlying graphics package (the **AGI** library) and a new library which interfaces to native PGPLOT only (the **AGP** library). In both these libraries the routines are split into two functional types. One set of routines deals with the manipulation of, and navigation around, the database; these routines are prefixed with **AGI_**. The other set of routines provides an interface between the database and the graphics package used by the application. In the AGI library these routines are prefixed **AGS_** if the graphics package is SGS (SUN/85), **AGP_** if the package is GKS-based PGPLOT (SUN/15) and **AGD_** if the package is IDI (SUN/65). The new AGP library only interfaces to native PGPLOT (SUN/15) and so only contains those graphics routines prefixed with **AGP_**. The two libraries can create identical database files on disk and thus new native-PGPLOT applications can be arranged to co-exist with older applications based on GKS. Current Starlink policy is that GKS is being phased out and so all new applications should be designed to use the AGP library only.

The basic structure of the database is an object called a picture. This represents a rectangular area on the display surface and contains information about the coordinate system in use when the picture was created. Pictures are usually created in the database by one of the interface routines specific to a particular graphics package. For instance the routine **AGS_SZONE** will save the extent and coordinate system of the current SGS zone as a picture in the database. However pictures in the database are independent of the graphics package that created them. This means that, for example, a picture created by one application using SGS graphics can be used by another application employing PGPLOT. It is even possible, with care, to use two graphics packages, such as SGS and IDI, in one application by using AGI to mediate between them.

Although pictures are independent of a graphics package they are not independent of the graphics device to which they refer. When a picture is created it contains an implicit reference to

the current device, even if nothing has been plotted on the screen or even if a graphics package has not been activated on the device. The pictures in the database are grouped into a larger unit, here called a workstation structure, that contains all the pictures on a given device, stored in the order they were created.

Pictures are given a name which gives an indication of what the plotting space contains. For effective communication between applications it is important that the picture names are standardised. It is customary for a picture containing other pictures to be referred to as a 'FRAME' picture, and a picture containing a plot to be referred to as a 'DATA' picture. A 'BASE' picture refers to the whole plotting space and this name should not be used for other purposes.

As well as a name a picture structure contains a one line comment field which can be used to further identify a picture in the database. The name, comment and coordinate system are mandatory components of a picture in the database, but there are some other components which are optional. The first is a reference to the data associated with a picture. This is a complete description of where the data is located, containing both the file name and the location within the file. This can be used by a later application to access the same data set without having to prompt the user again.

The second optional component is a transformation structure. The default coordinate system of a picture stored in the database is a simple orthogonal linear set that increases left to right and bottom to top. If the coordinate system of a plot on the screen does not conform to this rule then a transformation which defines the relationship can be saved in the database. For example, an application may have displayed an image and saved a transformation from pixel coordinates to right ascension and declination in the database. A later cursor application can then use the transformation to report its results in these non-linear coordinates.

The last optional component of a picture is a label. This is basically a short character string which can be used by an application to uniquely identify a picture. Unlike the other components of a picture the label contents can be changed at any time, the only restriction being that the label is unique on a given device.

If it is necessary to store additional information in a picture structure, that cannot be accommodated by any of the defined components, then a MORE component is available for this purpose. It is important to remember, however, that the structure and contents of the MORE component can only be interpreted by applications that know how to use it, and a general purpose application will ignore the MORE structure.

3 Summary of AGI Calls

3.1 Interface to PGPLOT (both libraries)

AGP_ACTIV(status)
Initialise PGPLOT

AGP_ASSOC(param, acmode, pname, border, picid, status)
Associate a device with AGI and PGPLOT through ADAM

AGP_DEACT(status)
Close down PGPLOT

AGP_DEASS(param, parcan, status)
De-associate a device from AGI and PGPLOT

AGP_NVIEW(border, status)
Create a new PGPLOT viewport from the current picture

AGP_SVIEW(pname, coment, picid, status)
Save the current PGPLOT viewport in the database

3.2 Interface to SGS (only in AGI library)

AGS_ACTIV(status)
Initialise SGS

AGS_ASSOC(param, acmode, pname, picid, newzon, status)
Associate a device with AGI and SGS through ADAM

AGS_DEACT(status)
Close down SGS

AGS_DEASS(param, parcan, status)
De-associate a device from AGI and SGS

AGS_NZONE(newzon, status)
Create a new SGS zone from the current picture

AGS_SZONE(pname, coment, picid, status)
Save the current SGS zone in the database

3.3 Interface to IDI (only in AGI library)

AGD_ACTIV(status)
Initialise IDI

AGD_ASSOC(param, acmode, pname, memid, picid, dispid, xsize, ysize, xoff, yoff, status)
Associate a device with AGI and IDI through ADAM

AGD_DEACT(status)
Close down IDI

AGD_DEASS(param, parcan, status)
Deassociate a device from AGI and IDI

AGD_NWIND(memid, dispid, xsize, ysize, xoff, yoff, status)
Define an IDI window from the current picture

AGD_SWIND(dispid, memid, xsize, ysize, xoff, yoff, pname, coment, wx1, wx2, wy1, wy2, picid, status)
Save an IDI window in the database

3.4 Control (both libraries)

AGI_ANNUL(picid, status)
Annul a picture identifier

AGI_ASSOC(param, acmode, picid, status)
Associate an AGI device with an ADAM parameter

AGI_BEGIN
Mark the beginning of a new AGI scope

AGI_CANCL(param, status)
Cancel the ADAM device parameter

AGI_CLOSE(status)
Close AGI in non-ADAM environments

AGI_END(picid, status)
Mark the end of an AGI scope

AGI_NUPIC(wx1, wx2, wy1, wy2, pname, coment, newx1, newx2, newy1, newy2, picid, status)
Create a new picture in the database

AGI_OPEN(wkname, acmode, picid, status)
Open an AGI device in a non-ADAM environment

AGI_PDEL(status)
Delete all the pictures on the current device

AGI_SELP(picid, status)
Select the given picture as the current one

AGI_SROOT(status)
Select the root picture for searching

3.5 Recall (both libraries)

AGI_RCF(pname, picid, status)
Recall first picture of specified name

AGI_RCFP(pname, x, y, picid, status)
Recall first picture embracing a position

AGI_RCL(pname, picid, status)
Recall last picture of specified name

AGI_RCLP(pname, x, y, picid, status)
Recall last picture embracing a position

AGI_RCP(pname, pstart, picid, status)
Recall preceding picture of specified name

AGI_RCPP(pname, pstart, x, y, picid, status)
Recall preceding picture embracing a position

AGI_RCS(pname, pstart, picid, status)
Recall succeeding picture of specified name

AGI_RCSP(pname, pstart, x, y, picid, status)
Recall succeeding picture embracing a position

3.6 Labels (both libraries)

AGI_ILAB(picid, label, status)
Inquire label of a picture

AGI_SLAB(picid, label, status)
Store label in picture

3.7 Reference (both libraries)

AGI_GTREF(picid, mode, datref, status)
Get a reference object from a picture

AGI_PTREF(datref, picid, status)
Store a reference object in a picture

3.8 Transformations (both libraries)

AGI_TCOPY(trnloc, picid, status)
Copy a transformation structure to the database

AGI_TDDTW(picid, nxy, dx, dy, wx, wy, status)
Transform double precision data to world coordinates

AGI_TDTOW(picid, nxy, dx, dy, wx, wy, status)
Transform data to world coordinates

AGI_TNEW(ncd, ncw, dtow, wtod, picid, status)
Store a transformation in the database

AGI_TRUNC(status)
Truncate the AGI database file by removing unused space

AGI_TWTDD(picid, nxy, wx, wy, dx, dy, status)
Transform double precision world to data coordinates

AGI_TWTOD(picid, nxy, wx, wy, dx, dy, status)
Transform world to data coordinates

3.9 Inquiries (both libraries)

AGI_IBASE(picid, status)
Inquire base picture for current device

AGI_ICOM(coment, status)
Inquire comment for the current picture

AGI_ICURP(picid, status)
Inquire the current picture

AGI_INAME(pname, status)
Inquire name of the current picture

AGI_IPOBS(picid, lobs, status)
Is current picture obscured by another?

AGI_ISAMD(picid, lsame, status)
Inquire if pictures are on same device

AGI_ISAMP(picid, lsame, status)
Inquire if two pictures are the same

AGI_ITOBS(nxy, x, y, ltobs, status)
Inquire if test points are obscured

AGI_IWOCO(wx1, wx2, wy1, wy2, status)
Inquire world coordinates of current picture

3.10 More (both libraries)

AGI_IMORE(picid, lmore, status)
Inquire if a MORE structure exists

AGI_MORE(picid, acmode, morloc, status)
Return an HDS locator to a MORE structure

3.11 Example skeleton applications

The lengthy subroutine list suggests that any application utilising AGI could end up looking complicated, but this need not be the case. By using the wrap-up routines simple applications need only call two or three routines from the list. As an example consider first an application that plots something on the screen and makes an entry in the database. A second application then uses a cursor to read a position from the graph. The second application is quite general and will work with any application that creates a DATA picture in the database, such as KAPPA:DISPLAY. Note that, as this example uses PGPLOT, it can be linked with either the GKS-based AGI library or the native-PGPLOT AGP library. However Starlink policy is that GKS is being phased out and so all new production applications should be linked with the **AGP** library as described in Section 8.

```
      SUBROUTINE PPLOT( STATUS )
      INCLUDE 'SAE_PAR'
      INTEGER I, ID1, ID2, N, STATUS
      PARAMETER ( N = 100 )
      REAL XP( N ), YP( N )

*   Check inherited global status
      IF ( STATUS .NE. SAI__OK ) GOTO 99

*   Define the function to plot
      DO I = 1, N
         XP( I ) = REAL( I )
         YP( I ) = REAL( I )
      ENDDO
```

```
*   Open AGI and PGPLOT through the ADAM interface
      CALL AGP_ASSOC( 'DEVICE', 'WRITE', ' ', .TRUE., ID1, STATUS )     ...1
      IF ( STATUS .NE. SAI__OK ) GOTO 99

*   Define the world coordinates of the viewport to match the data
      CALL PGSWIN( 0.0, 100.0, 0.0, 100.0 )

*   Create a box and plot the data
      CALL PGBOX( 'BCNST', 0.0, 0, 'BCNST', 0.0, 0 )
      CALL PGLINE( N, XP, YP )

*   Save the PGPLOT viewport as a picture in the database
      CALL AGP_SVIEW( 'DATA', 'PGLINE output', ID2, STATUS )            ...2

*   Close down AGI and PGPLOT cancelling the parameter
      CALL AGP_DEASS( 'DEVICE', .TRUE., STATUS )                        ...3

  99  CONTINUE
      END
```

The program requires an interface file to define the DEVICE parameter. The following example will do the job:-

```
interface PPLOT
   parameter DEVICE
      access 'READ'
      vpath 'PROMPT'
      prompt 'Display device '
   endparameter
endinterface
```

Program notes:

1. The routine **AGP_ASSOC** is used to access the database and open PGPLOT. It is a wrap-up routine for a number of commonly called AGI functions. The mode is set to 'WRITE' so that the plotting area is cleared and the border argument is set true to allow room for annotation of the axes. Passing an empty string in the name argument results in the current database picture being used to define the PGPLOT viewport.
2. The current PGPLOT viewport is saved as an AGI picture in the database.
3. The database and PGPLOT are closed using another of the wrap-up routines. This also tidies up any picture identifiers returned from AGI routines, and in this example cancels the parameter association.

```
      SUBROUTINE PCURS( STATUS )
      INCLUDE 'SAE_PAR'
      INTEGER ID, STATUS
      CHARACTER CH, TEXT*64
      REAL XC, YC
```

```
*   Check inherited global status
      IF ( STATUS .NE. SAI__OK ) GOTO 99

*   Open AGI and PGPLOT through the ADAM interface
      CALL AGP_ASSOC( 'DEVICE', 'READ', 'DATA', .FALSE., ID, STATUS )   ...4
      IF ( STATUS .NE. SAI__OK ) GOTO 99

*   Request a PGPLOT cursor
      CALL PGCURSE( XC, YC, CH )

*   Report the result
      WRITE( TEXT, '( ''Cursor position ='', 2F6.1 )' ) XC, YC
      CALL MSG_OUT( 'PCURS', TEXT, STATUS )

*   Close down AGI and PGPLOT cancelling the parameter
      CALL AGP_DEASS( 'DEVICE', .TRUE., STATUS )                        ...5

  99  CONTINUE
      END
```

Program notes:

4. The mode argument in **AGP_ASSOC** is set to 'READ' to ensure that the plot is not cleared. The name argument specifies that the last picture of type DATA should be recalled. The border argument is set false to ensure that the PGPLOT viewport is created to exactly match the DATA picture.
5. The call to **AGI_END** within **AGP_DEASS** resets the current picture to be the one current when the application began.

4 Control

4.1 Opening the Database

AGI can be used independently of a graphics system. When AGI is opened, no specific graphics system is associated with the database. When it is required to display something on a device, a graphics package is selected and the plotting done. The opening and closing of the package is performed separately from the opening and closing of the database [1]. The following simplified block diagram illustrates this operation.

```
OPEN AGI ( e.g. AGI_ASSOC )

   OPEN the graphics package ( e.g. AGP_ACTIV )
   OPEN the graphics device ( e.g. AGP_NVIEW )
   plot with the graphics package
   CLOSE the graphics package ( e.g. AGP_DEACT )

CLOSE AGI ( e.g. AGI_CANCL )
```

[1] There are routines, such as **AGP_ASSOC** and **AGP_DEASS**, which package both operations in one unit.

The first step of opening the database is done by calling either **AGI_ASSOC** or **AGI_OPEN**. **AGI_ASSOC** is used in ADAM tasks and obtains the name of the device through the ADAM parameter system.

```
CALL AGI_ASSOC( PARAM, ACMODE, PICID, STATUS )
```

PARAM is the name of the parameter in the ADAM interface file through which the device name is to be obtained. It should be stressed that **AGI_ASSOC** does not open the specified device for plotting, it opens the database entry for that device. The opening of the device for plotting is handled by the routines specific to a particular graphics package, for instance the **AGS_** routines. ACMODE is an access mode which has a particular meaning for AGI. It does not refer to the access mode of the database, i.e. whether an application can update the database or not, but instead it refers to the access mode of the graphics device. It is used to control whether the device should be cleared when it is opened by a graphics package. The possible modes are 'READ', 'UPDATE', and 'WRITE'. The modes 'READ' and 'UPDATE' leave the display as it is; the 'WRITE' mode clears the area of the display specified by the current picture returned in PICID. PICID returns an identifier to the current picture in the database; on opening the database this will be the picture that was current when the database was last accessed. If there is no previous entry in the database for this device then a 'BASE' picture will be created. This 'BASE' picture will fill the display area and will be created with a default world coordinate system, in the same way as the SGS base zone.

AGI_OPEN is used in non-ADAM tasks.

```
CALL AGI_OPEN( WKNAME, ACMODE, PICID, STATUS )
```

WKNAME is the device name. The other arguments are the same as those in **AGI_ASSOC**.

The given device name, whether passed through the parameter system or the argument list, should obey the syntax rules described in the next Section (4.2).

Devices with windowing capabilities allow the user to change the size of the display window between applications. This has a detrimental affect on AGI because the size of the base picture, and the relationship between pictures changes. Rather than considering this an error status, which could terminate an application, AGI deletes all the pictures on this device, creates a new base picture and sends the following message to the user.

```
Display has changed size.
Deleting previous database entries.
```

This allows any application that can continue from such a situation to do so. For instance it is sensible to allow a display application that uses the current picture for output to continue in this situation. In this case, after the existing database entries have been deleted, the current picture will be the base picture.

4.2 Device names

For applications linked with the GKS based **AGI** library the user should supply a valid GNS device name (SUN/57) which AGI converts into an internal name that locates the workstation structure in the database. For devices that have more than one plotting plane, it is desirable that

AGI treat pictures plotted on the different planes as being on the same device, so that plots on the overlay plane can be lined up with plots below them. Database entries for pictures on these different planes, which are different devices in GKS and SGS, will therefore appear in the same AGI workstation structure.

The **AGP** library is based on native-PGPLOT style device names of the form `<devname>/<devtype>`. The graphics device to be used is selected by the `<devtype>` component. The `<devname>` component before the / will be some form of name used by the device driver (typically a physical device on the system or an output file name) but is often omitted as most drivers contain a sensible default value. For example a graphics device of just `/gwm` will open a Starlink GWM X11 window with a default window name of `xwindows` while a complete specification of `mywindow/gwm` will open a GWM window with the name `mywindow`.

For native-PGPLOT based applications to co-exist with GKS-based ones it is important that users can continue to supply "old_style" GKS device names where these have a sensible translation. Where there **IS** such a translation the AGI database records must also be made to match. Logic has been incorporated into the AGP library to perform this name translation process. Also in AGP typing a single ? for the device name will list all the possible translations. At the time of writing this document these were (but see also section Section 7):

```
gif_l     (/GIF)           GIF-format file, landscape
gif_p     (/VGIF)          GIF-format file, portrait
hpgl_l    (/HPGL)          Hewlett-Packard HP-GL plotters, landscape
hpgl_p    (/VHPGL)         Hewlett-Packard HP-GL plotters, portrait
hpgl2     (/HPGL2)         Hewlett-Packard graphics language
xterm     (/XTERM)         Tektronix terminal emulator
wd_l      (/WD)            X Window dump file, landscape
wd_p      (/VWD)           X Window dump file, portrait
xserve    (/XSERVE)        PGPLOT X window server
tek_4010  (/TEK4010)       Tektronix 4006-4010 storage-tube terminal
tek_4107  (/TEK4100)       Tektronix 4100-series terminals
ps_p      (/VPS)           Postscript printers, monochrome, portrait
ps_l      (/PS)            Postscript printers, monochrome, landscape
epsf_p    (/VPS)           Encapsulated postscript, portrait
epsf_l    (/PS)            Encapsulated postscript, landscape
pscol_p   (/VCPS)          PostScript printers, color, portrait
pscol_l   (/CPS)           PostScript printers, color, landscape
epsfcol_p (/VCPS)          Colour encapsulated postscript, portrait
epsfcol_l (/CPS)           Colour encapsulated postscript, landscape
xwindows  (xwindows/GWM)   Starlink GWM xwindow
x2windows (xwindows2/GWM)  Starlink GWM xwindow
x3windows (xwindows3/GWM)  Starlink GWM xwindow
x4windows (xwindows4/GWM)  Starlink GWM xwindow
```

As an example of the overall effect native-PGPLOT graphics specifications of `xwindows/gwm`, `/gwm` or `xwindows` are all valid and equivalent in the **AGP** version of the AGI library. They will select a GWM window with name `xwindows` which will have AGI database records identical to those created by the **AGI** library routines for the graphics device with the GNS name of `xwindows`.

The user will still see some slight differences from previous GKS behaviour when using GKS style device names via this scheme. For example:

- Native-PGPLOT, unlike the GKS-based version, always produces "encapsulated" postscript files.
- The default output file for postscript files will be **pgplot.ps** with (unlike GKS) no version numbers. Thus care must be taken to ensure that consecutive graphics applications do not over-write this file. An alternative postscript file name can be specified either by the GKS syntax ps_l;fred.ps or the native-PGPLOT syntax fred.ps/PS

4.3 Picture identifiers

Pictures are flagged by means of an integer identifier (the picture identifier) which combines knowledge of a device name and a picture number in the same way that an SGS zone number does. Both **AGI_ASSOC** and **AGI_OPEN** return a picture identifier to the current picture on the given device. Pictures in the database can only be referenced by means of these identifiers which are returned to the application from the appropriate AGI routines. These identifiers should only be passed on to other AGI routines and should not be altered in any way by an application.

Once a picture is no longer required it is good practice to release it using **AGI_ANNUL** which liberates internal workspace. Picture identifiers can be automatically released by using a begin-end block (see Section 4.5).

4.4 Closing the Database

It may seem a bit premature to close the database having only just described how to open it, but some things are best explained at this stage. In an ADAM environment the routine **AGI_CANCL** cancels the association of the specified parameter and performs specific tidying up operations concerning that parameter. If the parameter association has to be retained outside the program then **AGI_CANCL** should not be called.

If the workstation structure is opened by **AGI_OPEN**, a call to **AGI_CLOSE** should be used to close it.

General tidying up operations (e.g. closing the database) are performed when the last active identifier has been annulled, either with **AGI_ANNUL** or **AGI_END**. It is important that tasks clean up all their identifiers otherwise unusual side effects may occur if the executable image is permanently resident in memory (such as an ADAM task).

Once the database has been closed, **AGI_TRUNC** can be called to truncate the database file to remove any unused space, thus keeping the file size at a minimum.

4.5 Begin-End blocks

As the database is not closed until all identifiers have been annulled it is important that each task keeps its allocation of identifiers under control. For a complicated application annulling individual identifiers may be tiresome and so a general tidying up scheme is available using a begin-end block. The routines **AGI_BEGIN** and **AGI_END** are used to bracket a block of code and all the identifiers allocated within the block are automatically annulled by **AGI_END**. These routines may be nested, (up to a depth of eight) but each **AGI_BEGIN** must have its corresponding **AGI_END**.

As well as annulling identifiers the routine **AGI_END** performs another important function. The routine takes a picture identifier as its input and this picture is made current. If the identifier is valid then this is no different to calling **AGI_SELP**, but if the argument passed to **AGI_END** is a negative number then the picture that was current when the corresponding **AGI_BEGIN** was called is made current again. This is useful for applications that need to restore the current picture to that which was current when the application began.

The following two examples show different uses of a begin-end block. In the first example the call to **AGI_END** annuls all the identifiers, which results in the database being closed, since there are no more active identifiers. As there was no current picture when the corresponding **AGI_BEGIN** was called (since the database had not been opened) the negative argument in **AGI_END** has no effect in this case, and the picture that was current just before the call to **AGI_END** remains current.

```
*   Begin an AGI scope
      CALL AGI_BEGIN

*   Open AGI on a device obtained from the parameter system
      CALL AGI_ASSOC( 'DEVICE', 'WRITE', ID, STATUS )

*   Main body of program
      <main body of program>

*   Annul identifiers and close the database
      CALL AGI_END( -1, STATUS )
```

In the second example **AGI_BEGIN** is put after **AGI_ASSOC**. When **AGI_END** is called the identifier returned from **AGI_ASSOC** is not annulled, since it is outside the scope of the begin-end block. The negative argument in **AGI_END** makes the picture that was current when the corresponding **AGI_BEGIN** was called current again (in this example the picture that was current when the application began).

```
*   Open AGI on a device obtained from the parameter system
      CALL AGI_ASSOC( 'DEVICE', 'WRITE', ID, STATUS )

*   Begin an AGI scope
      CALL AGI_BEGIN

*   Main body of program
      <main body of program>

*   Annul identifiers from the main body of the program
*   and reinstate the current picture
      CALL AGI_END( -1, STATUS )

*   Annul the initial identifier and close the database
      CALL AGI_ANNUL( ID, STATUS )
```

4.6 The Database file

The database is stored in an HDS container file named agi_<node>.sdf in the directory defined by the environment variable AGI_USER or in your home directory if AGI_USER hasn't been

defined (HDS is the Hierarchical Data System described in SUN/92). The name of the node is included in the file name to prevent two or more devices which have the same name but are on different nodes causing a conflict. The contents of the file can be examined using the ADAM application TRACE. In certain circumstances the database file may become corrupted. If this is suspected it is best to delete the appropriate file and start again.

The name of the container file can be optionally changed by defining the environment variable AGI_NODE to be equivalent to some user defined string. If such a environment variable exists then the container file is constructed by replacing the default node with the new string. Thus if the environment variable AGI_NODE is defined to be 'nonode' then the container file will have the name agi_nonode.sdf in the directory AGI_USER. This mechanism can be used to pick up a database created on a different node by defining the logical name to be equivalent to the required node name.

5 Components

5.1 Pictures

Pictures represent rectangular areas on a display. The pictures have default coordinate systems that are linear and increase left to right and bottom to top, corresponding to GKS rules. Each picture can optionally have an associated transformation to allow for other coordinate systems. Pictures are stored in the database in the order they were created, so when recalling the last picture of a given name, for example, the most recently created picture of that name will be recovered.

The AGI interface to each graphics package will interpret the meaning of a picture to suit the design of that particular package. When used in conjunction with SGS a picture is made equivalent to an SGS zone. For example to save an SGS zone as a picture use

```
CALL AGS_SZONE( PNAME, COMENT, PICID, STATUS )
```

PNAME and COMENT are used to identify the picture in the database as described in the sections below. PICID returns the identifier for the new picture in the database.

There is one routine, **AGI_NUPIC**, that allows new pictures to be created in the database without necessarily opening a graphics package first.

```
 CALL AGI_NUPIC( WX1, WX2, WY1, WY2, PNAME, COMENT,
:                NEWX1, NEWX2, NEWY1, NEWY2, PICID, STATUS )
```

WX1, WX2, WY1, WY2 define the rectangular extent of the new picture in terms of the world coordinates of the current picture. PNAME and COMENT describe the picture as discussed in the following sections. NEWX1, NEWX2, NEWY1, NEWY2 specify the world coordinate system of the new picture. PICID returns the identifier for the new picture in the database.

AGI works with a current picture in an analogous manner to SGS and its current zone. A new picture has to be created within the bounds of the current one, and this then becomes the current picture. Pictures are recalled from the database only if they lie within the physical bounds of the

current one; the recalled picture then becomes the current one. A picture can be made current by calling the routine **AGI_SELP**.

The picture recall routines, **AGI_RC***, provide the means of traversing the database. The key to the search is the picture name, and the search ends when a picture of the given name is found that lies within completely the bounds of the current picture. This picture then becomes the current picture. If no suitable picture is found then an error status is returned. The search can be made more general by giving an empty name string. This results in the first picture on the search path, that lies within the current one, being recalled. **AGI_RCL** searches backwards from the last (most recent) picture in the database. **AGI_RCF** searches forwards from the first (most ancient) picture in the database. **AGI_RCP** searches backwards starting at the picture given by the PSTART identifier, and **AGI_RCS** does the same searching forwards.

A further restriction on the search can be introduced by using the **AGI_RC*P** routines. With these, a picture is only recalled if it corresponds to the given name, lies within the bounds of the current picture, *and* encompasses a given point in the world coordinate space of the current picture. This type of recall is useful when a cursor has been put up on the display, and the user asked to select a picture with the cursor.

A potential problem arises when searching backwards through the database. The hierarchy of pictures in the database implies that previous pictures will usually be larger than the current one, but the search strategy requires the current picture to be larger than the picture being sought. This problem can be overcome by selecting a picture known to be larger than any picture being sought as the current one. The base picture is the most convenient as this is larger than any picture in the database. The following example shows how to search backwards through the database for a picture with a particular label (labels are discussed in Section 5.6).

```
*   Get an identifier for the base picture and select it as current.
      CALL AGI_IBASE( BASEID, STATUS )
      CALL AGI_SELP( BASEID, STATUS )

*   Recall the last picture in the database of any name and get its label.
      CALL AGI_RCL( ' ', PICID, STATUS )
      CALL AGI_ILAB( PICID, PLABEL, STATUS )

*   Loop through the database until the a match with LABEL is found.
      DO WHILE ( ( PLABEL .NE. LABEL ) .AND. ( STATUS .EQ. SAI__OK ) )

*   Reselect the base picture as the current one.
*   This ensures the search does not fail because a picture is not
*   within the current one.
         CALL AGI_SELP( BASEID, STATUS )

*   Search backwards starting at the previously recalled picture
         PICIDS = PICID
         CALL AGI_RCP( ' ', PICIDS, PICID, STATUS )

*   Inquire the label of this picture
         CALL AGI_ILAB( PICID, PLABEL, STATUS )
      ENDDO
```

5.2 The root picture

For devices that allow the memories to be scrolled independently there is a potential conflict when searching for a picture. This occurs because under normal conditions a picture can only be recalled from the database if it lies within the current picture. If the pictures are located on different memory planes then this can be negated by scrolling one of the memories. To overcome this a more general search is allowed by selecting the root picture which contains all other pictures (including the base picture). A call to the routine **AGI_SROOT** selects the root picture. A subsequent call to any of the recall routines **AGI_RC*** will obtain the relevant picture without it having to lie within the current picture. The root picture is deselected in the recall routine, and so **AGI_SROOT** has to be called on every occasion that an unlimited search is to be made. Recall is the only operation allowed with the root picture. Any other operation called while the root picture is selected will result in the previously current picture being used.

5.3 World coordinates

When AGI needs to access information in the database by position it uses the world coordinate system of the current picture. World coordinates are those set up by the user to represent a useful linear range of coordinates which fill the picture space. These world coordinates must be linear and increase from left to right and from bottom to top. If the required coordinate system does not correspond to these rules then the user should choose a basic world coordinate system that does obey them, and then define a transformation to go to and from the basic system and the required one (see the section on transformations 5.8).

When a picture is created with one of the package interface routines the current world coordinates of the graphics zone are stored in the database. If the user subsequently changes the world coordinate system of the graphics space on the physical device, errors may result if information is requested from the database by position. The routine **AGI_IWOCO** can be used to inquire the world coordinates of the current picture.

5.4 Name

The name is a character string indicating the type of picture created. It should reflect the general intention of the plotting space, and not describe specifically what the picture contains. For example, the name 'FRAME' is used to indicate that the space will be used to group together a collection of other plots. The name 'DATA' is used to indicate that the space will contain some sort of representation of data in graphical form, e.g. a grey-scale image, or a scatter plot. The name 'BASE' is used by AGI to signify the base picture, and this name should not be used for other purposes. The length of the name string is limited to the number of characters defined by the AGI__SZNAM constant from the AGI_PAR FORTRAN include file (currently 15 characters). When stored in the database the character string will have leading blanks removed and converted into upper case.

There are at present no compulsory names, but clearly consistent usage is necessary if packages written by different programmers are to integrate successfully.

5.5 Comment

The comment is a character string containing a description of the picture. The application can use this component to store any text that will help identify the picture in the database. It is restricted

to one line of text at present, with a maximum length given by the parameter AGI__CMAX defined in the AGI_PAR include file. The comment string is stored with the picture at the time of picture creation.

5.6 Labels

There is another character identifier that can be stored in the database and used to distinguish a picture more precisely. This is the label which is an optional component of a picture. It differs from other structures in AGI in that its contents can be changed at any time (by overwriting the existing contents), whereas other picture elements can only be defined when they are first created. Because the label contents can be changed at any time the responsibility for guarding against misuse is left up to the application/user.

The labels differ in another way from the name component in that each label within a workstation structure must be unique. If a new label clashes with an exiting one within a workstation structure then the existing label will be deleted and replaced by a blank string. Although the labels are stored with mixed case the comparison is done with leading blanks removed and in uppercase only.

The routine **AGI_SLAB** will associate a label with an existing picture, indicated by the picture identifier. If the picture identifier is negative then the current picture is used to store the label. If a label already exists for a picture then the old one will be overwritten. The routine **AGI_ILAB** will return the label of the requested picture. If the picture identifier is negative then the current picture is searched. If no label is associated with this picture then a blank string is returned.

The length of the label string is limited to the number of characters defined by the AGI__SZLAB constant from the AGI_PAR include file (currently 15 characters).

5.7 Reference to data

It may often be desirable to access the data used by one application in a second one. To store the actual data in the database would be wasteful of disk space since this data would be replicated in the original data file and in the database. Even more space would be wasted if the same data was associated with more than one picture in the database. Instead of saving the data in the database AGI stores a reference to it (a string describing it's location) that enables other applications to recover the data.

A reference to a data object is not a compulsory component of a database picture, and as such will not be created when a new picture is inserted in the database. An explicit call to one of the appropriate routines is used to create or access a reference object in the database. The subroutine argument that defines the reference object can be either an HDS locator or any character string reference. If the string is a valid HDS locator then a reference is constructed to point to the relevant object, otherwise the string is assumed to be a reference itself and is stored as supplied. This dual definition of what defines a reference is intended to smooth the transition from applications using raw HDS calls to using NDF routines to access the data. When the NDF reference mechanism is well established the HDS interface will be removed.

A reference is saved in the database by calling the routine **AGI_PTREF**. The reference structure is stored within the picture indicated by the picture identifier. If the picture identifier is negative then the current picture is used to store the reference. If a reference already exists in the picture then an error status will be returned.

A reference is obtained from the database by calling the routine **AGI_GTREF**. The reference is obtained from the picture indicated by the picture identifier. If the picture identifier is negative then the current picture is used to obtain the reference. If the reference was created from an HDS locator then an HDS locator is returned and this should be annulled in the application using **REF_ANNUL** rather than **DAT_ANNUL**. This will ensure that the file containing the reference is properly shut down.

One important point to note is that AGI does not check if the data being referenced is valid or not.

5.8 Coordinate transformations

The default coordinate system used by AGI is an orthogonal linear system of world coordinates that increase from left to right and bottom to top. If the data coordinate system used by an application complies with these rules then a basic picture saving operation will store sufficient information to recreate these coordinates at a later stage.

If the transformation between data coordinates and the world coordinates of a rectangular piece of the display screen is more complicated than this simple system then a separate transformation has to be stored in the database to define the relationships. AGI uses the TRANSFORM facility described in SUN/61 to define the transformations. AGI does not offer the full flexibility of the TRANSFORM package as it is only interested in the limited case of transforming from the data coordinates into the flat, world coordinates of the display device.

There are two types of transformation to be defined, the first is the transformation from data to world coordinates, and the second is the transformation from world to data coordinates. Usually both will be defined, although this is not a necessity. The present implementation only allows for 2-dimensional data coordinates, and an error will be returned if the number of data variables differs from this.

The transformations are defined as character strings which describe the mathematical formulas as if they appeared in a piece of FORTRAN code. Therefore the description of the formulas should follow all the FORTRAN rules for operators and functions. As an example the case of data coordinates in polar coordinates will be used. The transformation from polar coordinates (r, θ) to Cartesian world coordinates (x,y) is defined by the equations $x = r\cos(\theta)$ and $y = r\sin(\theta)$. The inverse transformation is defined by the equations $r = \sqrt{x^2 + y^2}$ and $\theta = \tan^{-1}(y/x)$. These are formulated in a program in the following way.

```
*   Define the number of input ( data ) and output ( world ) variables
*   Note: This should be 2 for each transformation.
      INTEGER NCD, NCW
      PARAMETER ( NCD = 2, NCW = 2 )

*   Declare arrays for the two sets of transformations
      CHARACTER * 32 DTOW( NCW ), WTOD( NCD )

*   Assign the data to world transformation functions
      DTOW( 1 ) = 'X = R * COS( THETA )'
      DTOW( 2 ) = 'Y = R * SIN( THETA )'

*   Assign the world to data transformation functions
      WTOD( 1 ) = 'R = SQRT( X * X + Y * Y )'
      WTOD( 2 ) = 'THETA = ATAN2( Y, X )'
```

The transformation is then stored in the database using the routine **AGI_TNEW**.

If a transformation already exists in an HDS structure which converts the data coordinates into the two dimensional world coordinates of the display screen then the transformation can be copied into the database using the routine **AGI_TCOPY**.

The AGI interface has four routines which will perform transformations that have been stored in the database. The routine **AGI_TDTOW** will transform data coordinates into world coordinates, and the routine **AGI_TWTOD** will perform the inverse. The argument list for these routines includes a picture identifier which indicates the picture containing the transformation. If the identifier is negative then the current picture is searched. The routines take two arrays containing the x and y coordinates of the points to be transformed and returns two arrays containing the new x and y coordinates. In this context x and y need not mean a Cartesian system when used for the data coordinates, they are simply names for the first and second coordinate of the data system. In the above example r and θ correspond to the x and y coordinates of the data. If no transformation is found in the indicated picture then the identity transformation (output = input) is used. The transformations can be executed with double precision by calling the equivalent routines **AGI_TDDTW** and **AGI_TWTDD**.

5.9 Inquiries

There are a number of inquiry routines available to investigate the properties of the current picture. **AGI_INAME** and **AGI_ICOM** inquire the name and description of the current picture. **AGI_ISAMD** inquires if the given picture is on the same physical device as the current picture. **AGI_ISAMP** inquires if the given picture identifier points to the same picture in the database as the current picture identifier. **AGI_IPOBS** tests if the current picture is obscured either partially or totally by a specified picture in the database. Obscuration means here that the given picture intersects the current picture and that it overlays the current picture (was created more recently). The obscuration of the current picture can be tested against all subsequent pictures in the database by specifying a negative number in place of the picture identifier in the argument list. **AGI_ITOBS** tests the obscuration of a number of test points. The test points are specified in the coordinate system of the current picture, and each point is tested in turn for obscuration by any picture overlying the current one.

5.10 More

It may be desirable for an application to store information in the database that is outside the scope of the routines provided. For this reason an application can access a MORE structure using the routine **AGI_MORE** and save arbitrary HDS objects within it. A MORE structure can be created for each of the pictures in the database. It must be stressed that if an application uses this facility then it is responsible for the contents and whatever any subsequent application may make of them. Also an application may need to cope with the situation of a corrupted structure.

The MORE structure is accessed for reading and writing using the routine **AGI_MORE**

```
CALL AGI_MORE( PICID, ACMODE, MORLOC, STATUS )
```

The access mode (ACMODE) controls the action of this routine. If the mode is 'WRITE' then an HDS locator to an empty MORE structure is returned. If there was no previous MORE structure

then one is created, and if there was a structure present then the previous contents are erased. At present there is no lock to prevent one application erasing a MORE structure being accessed by another. If the access mode is 'READ' or 'UPDATE' then the routine returns a locator to the top of the MORE structure, unless there was no structure present in which case an error is returned. The HDS locator returned by this routine has to be annulled by the calling application.

The presence of a MORE structure can be tested using the routine **AGI_IMORE**. The return argument is true if a MORE structure exists for the given picture, otherwise the argument is false.

6 Interface to Graphics Systems

The AGI database is useful in two basic contexts. The first is to pass information between separate applications that may or may not be plotting on the device with the same graphics package. An example of two independent PGPLOT based applications has been given in Section 3.11 while the use of the database file to allow native-PGPLOT based applications to interact with GKS-based applications is described in Section 4.2. For applications using the **AGI** GKS-based library a second use is to allow different graphics packages to interact within a single application, for example by using SGS to draw on top of an image displayed with IDI. These two packages have completely different models of the display device and AGI is used to mediate between them. The major restriction with this second mode of use is that the two graphics packages cannot both be open at the same time. There will be other restrictions depending on how the physical device characteristics are utilized by each graphics package; for example the size of the display area may be different in the different packages and this will cause problems if a picture defined by one package is beyond the limits of another.

The following simplified block diagram illustrates the recommended operation of AGI and two graphics packages within one application.

```
OPEN AGI ( e.g. AGI_ASSOC )

   OPEN the first graphics package ( e.g. AGD_ACTIV )
   OPEN the device ( e.g. AGD_NWIND )
   plot with the first graphics package
   CLOSE the first graphics package ( e.g. AGD_DEACT )

   OPEN the second graphics package ( e.g. AGS_ACTIV )
   OPEN the device ( e.g. AGS_NZONE )
   plot with the second graphics package
   CLOSE the second graphics package ( e.g. AGS_DEACT )

CLOSE AGI ( e.g. AGI_CANCL )
```

6.1 Interface to PGPLOT

As described in Section 2 versions of the PGPLOT interface routines with prefix **AGP_** exist in two libraries. The **AGI** library includes these and SGS routines based on GKS graphics together with IDI routines. The **AGP** library is designed to only interface to native-PGPLOT and has

its own link commands described in Section 8. The **AGP_** routines have indentical names and argument lists for both libraries. Note, however, that the **AGP** library does **not** support the concept of PGPLOT plotting into an SGS zone.

A set of routines are supplied to interface the database to the PGPLOT graphics package, these are prefixed by AGP_ instead of AGI_. A picture in the database is equated with a PGPLOT viewport. Previous versions of PGPLOT only allowed one device to be open at one time and this restriction still exists in the **AGP_** routines. The PGPLOT interface to AGI signifies this by returning an error status if another device is requested while a current PGPLOT device is open.

PGPLOT is activated with a call to **AGP_ACTIV**, but it does not actually open a device for plotting. A device is opened when the first call to **AGP_NVIEW** is made. The actual device that is opened will be the one associated with the current picture in the database. **AGP_NVIEW** creates a new PGPLOT viewport from the current AGI picture. If the border argument in **AGP_NVIEW** is false then the new viewport fills the area specified by the picture and the world coordinate system of the database picture is recreated in the viewport. If however the border argument in **AGP_NVIEW** is true then the viewport is created smaller than the area defined by the picture so that a standard width border surrounds the viewport. The world coordinates of this viewport are given the default values of 0 to 1 in each axis.

The current viewport can be saved as a picture in the database using a call to **AGP_SVIEW**. The size of the new picture must be less than or equal to the size of the current picture in the database, otherwise an error status is returned. Although PGPLOT uses the concept of a border around a plot in which to put annotations, it does not let an application inquire the size of this border and so it is not possible to save the border area as a 'FRAME' picture in the database from within the associated viewport. If such a 'FRAME' picture is required in the database then it must be saved before the viewport is created, or must be recreated from an existing picture.

PGPLOT is shut down using **AGP_DEACT**.

The following code segment gives an example of an application that creates a viewport that matches the size of the current picture if the overlay flag is true, or creates a viewport with a border for annotation if the overlay flag is false.

```
*   Open AGI on a device obtained from the parameter system.
*   Do not clear the viewport if overlay mode is requested.
      IF ( OVER ) THEN
         CALL AGI_ASSOC( 'DEVICE', 'UPDATE', ID1, STATUS )
      ELSE
         CALL AGI_ASSOC( 'DEVICE', 'WRITE', ID1, STATUS )
      ENDIF

*   Activate the PGPLOT interface to AGI
      CALL AGP_ACTIV( STATUS )

*   If plotting over previous plot find the last data picture and
*   match the PGPLOT viewport to it
      IF ( OVER ) THEN

*   Use the current picture if it is a data picture otherwise
*   find the last data picture that fits within the current one
         CALL AGI_INAME( CNAME, STATUS )
         IF ( CNAME .NE. 'DATA' ) THEN
```

```
            CALL AGI_RCL( 'DATA', ID2, STATUS )
          ENDIF

*   If a suitable data picture was not found then abort
          IF ( STATUS .NE. SAI__OK ) <goto closedown>

*   Create the viewport to match the data picture
          CALL AGP_NVIEW( .FALSE., STATUS )

*   If not using an overlay then create a viewport with a border
        ELSE
          CALL AGP_NVIEW( .TRUE., STATUS )
        ENDIF
```

The next piece of code shows how the viewport and frame from the previous example can be stored in the database as pictures. If the overlay mode has been used then the viewport and associated frame will be the same size.

```
*   Select the picture within which the viewport was created
*   and inquire its world coordinates
      CALL AGI_SELP( ID1, STATUS )
      CALL AGI_IWOCO( WX1, WX2, WY1, WY2, STATUS )

*   Save this in the database as the frame picture using the
*   full extent of the picture.
*   Define a default world coordinate system for the new picture.
      CALL AGI_NUPIC( WX1, WX2, WY1, WY2, 'FRAME', 'Frame picture',
     :                0.0, 1.0, 0.0, 1.0, FID, STATUS )

*   Save the current viewport as the data picture
      CALL AGP_SVIEW( 'DATA', 'Data picture', VID, STATUS )
```

There are wrap-up routines to open and close AGI and PGPLOT within the ADAM environment using single calls. The sequence of calls **AGI_ASSOC**, **AGI_BEGIN**, **AGP_ACTIV**, **AGP_NVIEW** can be replaced by a single call to **AGP_ASSOC**. As with **AGS_ASSOC** an optional call to **AGI_RCL** can be made by passing a non-blank name argument. The sequence of calls **AGP_DEACT**, **AGI_END**, **AGI_CANCL** or **AGI_ANNUL** can be replaced by a single call to **AGP_DEASS**.

PGPLOT can be used as a stand-alone package. When GKS-based PGPLOT is used from the **AGI** library PGPLOT can also be started from within an SGS zone. In this case PGPLOT is initialised while SGS is still open. This is the only situation where AGI allows more than one graphics package to be open at one time. The following simplified program section illustrates the opening of PGPLOT from within an SGS zone.

```
*   Open AGI and SGS on a device obtained from the parameter system
      CALL AGS_ASSOC( 'DEVICE', 'WRITE', ID, IZONE, STATUS )

*   Create a new SGS zone of the required size and shape
      <Create a new zone with SGS calls>

*   Open up PGPLOT and create a new viewport matching the current zone
```

```
      CALL AGP_ACTIV( STATUS )
      CALL AGP_NVIEW( .FALSE., STATUS )

*   Create a new PGPLOT environment and draw the picture
      <Use PGPLOT to draw the picture>

*   Close down PGPLOT, SGS and AGI
      CALL AGP_DEACT( STATUS )
      CALL AGS_DEASS( 'DEVICE', .TRUE., STATUS )
```

Note that when using the AGP routines to open PGPLOT in the current SGS zone it is essential that SGS has been opened using the AGS routines. Normally **AGP_NVIEW** creates a viewport that matches the size of the current picture in the database, but in this instance the size of the viewport matches the current SGS zone. The new viewport inherits the world coordinate system of the current picture in the usual way.

The job of opening and closing the device is handled by the AGP_ routines and therefore PGBEGIN and PGEND should not be called in an application using AGI. Similarly the routines PGENV, PGPAPER and PGVSTAND should not be called as these subvert the underlying coordinate system.

6.2 Interface to SGS – (AGI library only)

A set of routines are supplied to interface the database to the SGS graphics package, these are prefixed by AGS_ instead of AGI_. A picture in the database is equated with an SGS zone. SGS is activated with a call to **AGS_ACTIV**. This does not open a device for plotting. A device is opened when the first call to **AGS_NZONE** is made. The actual device that is opened will be the one associated with the current picture in the database. This means that AGI has to be open before **AGS_NZONE** can be called. **AGS_NZONE** creates an SGS zone from the current AGI picture, and returns the zone identifier. An SGS zone is saved as a picture in the database using a call to **AGS_SZONE**. The size of the new picture must be less than or equal to the size of the current picture in the database otherwise and error status is returned. SGS is shut down using **AGS_DEACT**.

The job of opening and closing the device is handled by the AGS_ routines and therefore the routines **SGS_ASSOC** and **SGS_CANCL** or **SGS_OPEN** and **SGS_CLOSE** should not be called when using the AGS_ routines.

The first example shows how an application might save an SGS zone in the database.

```
*   Obtain the device name from the ADAM parameter system
      CALL AGI_ASSOC( 'DEVICE', 'WRITE', ID1, STATUS )
      CALL AGI_BEGIN

*   Open the given device for plotting
      CALL AGS_ACTIV( STATUS )
      CALL AGS_NZONE( IZONE, STATUS )

*   Plot the data in an SGS zone
      <plot the data with SGS>

*   Save the current zone as a picture in the database
```

```
      CALL AGS_SZONE( 'DATA', 'Description', ID2, STATUS )

*   Close down
      CALL AGS_DEACT( STATUS )
      CALL AGI_END( -1, STATUS )
      CALL AGI_CANCL( 'DEVICE', STATUS )
```

The second example shows how another application could recreate a zone, perhaps to overlay one plot with another, or to obtain coordinates with a cursor. It assumes that the current picture in the database is the one that is wanted. If this is not the case the recall routines can be used to search the database for the correct picture.

```
*   Obtain the device name from the ADAM parameter system
*   Use update mode to ensure the original plot is not cleared
      CALL AGI_ASSOC( 'DEVICE', 'UPDATE', ID1, STATUS )
      CALL AGI_BEGIN

*   Open the given device for plotting recreating the current
*   picture as an SGS zone
      CALL AGS_ACTIV( STATUS )
      CALL AGS_NZONE( IZONE, STATUS )

*   Plot with SGS
      <plot with SGS>

*   Close down
      CALL AGS_DEACT( STATUS )
      CALL AGI_END( -1, STATUS )
      CALL AGI_CANCL( 'DEVICE', STATUS )
```

These examples show that the AGI operations of accessing the database and opening the graphics package are common to both applications. These common opening and closing operations have been packaged up into single routines. Thus the sequence of calls **AGI_ASSOC**, **AGI_BEGIN**, **AGS_ACTIV**, **AGS_NZONE** can be replaced by a single call to **AGS_ASSOC**, and the sequence of calls **AGS_DEACT**, **AGI_END**, **AGI_CANCL** can be replaced by a single call to **AGS_DEASS**. Note that these wrap-up routines only exist for the ADAM interface. If the application is using the non-ADAM routines **AGI_OPEN** and **AGI_CLOSE** then the sequence of calls has to be made explicitly. Using these wrap-up routines then second application above would become

```
*   Obtain the device name from the ADAM parameter system and open
*   SGS on the given device
      CALL AGS_ASSOC( 'DEVICE', 'UPDATE', ' ', ID1, IZONE, STATUS )

*   Plot with SGS
      <plot with SGS>

*   Close down
      CALL AGS_DEASS( 'DEVICE', .TRUE., STATUS )
```

The third (picture name) argument in **AGS_ASSOC** can optionally be used to recall the last picture of the given name using **AGI_RCL**. If the name string is blank (as in the example above)

then this recall routine is not called and the current database picture is returned. If the second argument in **AGS_DEASS** is false then the ADAM parameter is not cancelled and instead the picture identifier returned from **AGS_ASSOC** is annulled.

If GKS calls are mixed in with the SGS calls to produce the plot then some GKS operations may result in the plot being cleared when the device is shut down. This can happen if the device is opened with 'WRITE' mode, which results in the zone being initially cleared, and then some aspect of the device, such as a pen colour, is changed using GKS calls. GKS believes that the original plot is now wrong because the colours are wrong and requests that the plot be redrawn in the new colours. By default this request does not happen immediately but is deferred until closedown. An application can force the update to happen immediately by setting the deferral mode:

```
*   Set the GKS deferral mode to 'ASAP'
      CALL SGS_ICURW( IWKID )
      CALL GSDS( IWKID, 0, 1 )
```

6.3 Interface to IDI – (AGI library only)

A set of routines are supplied to interface the database to the IDI graphics package, these are prefixed by AGD_ instead of AGI_. IDI is a low-level graphics package that does not have the kind of abstraction that packages like SGS and PGPLOT have. The coordinate system is not device independent and there is no natural structure in the specification that corresponds to a picture in the way that an SGS zone does. The AGI interface to IDI is therefore more contrived than for the other packages.

IDI instructions are directed to a specific device by means of a display identifier that has to be supplied as an argument to the IDI subroutines. The routine **AGD_NWIND** returns a display identifier that corresponds to the device associated with the current picture. This identifier can then be used as input to the IDI routines. If the device is not already open then **AGD_NWIND** will open the relevant device. Subsequent calls to **AGD_NWIND** will return the same identifier if the current picture is associated with the same device.

The routines **AGD_ACTIV** and **AGD_DEACT** should be called before and after any other AGD_ routines.

IDI has no structure that matches the concept of a picture in the database, and so an abstract space has been defined, here called simply a window, which represents a rectangular space in the memory of the device. The definition of such a window is similar to a the definition of a transfer window in IDI, which is used to load images into the display memory. IDI works in pixel coordinates and a window is defined by its size in pixels along each axis (XSIZE and YSIZE), and by an offset, in pixels, of its bottom left hand corner from the memory origin (XOFF and YOFF). The routine **AGD_SWIND** will save such a window as a picture in the database. The following code segment shows how a transfer window which has been set up to load an image into a memory can be saved as a picture in the database.

```
*   Set up the transfer window defined by XSIZE, YSIZE, XOFF, YOFF
*   to match the size of the image NX * NY
      XSIZE = NX
```

```
      YSIZE = NY
      CALL IIMSTW( DISPID, MEMID, 0, XSIZE, YSIZE, 8, XOFF, YOFF, STATUS )

*   Load the image held in the array ADATA
      NPIX = NX * NY
      CALL IIMWMY( DISPID, MEMID, ADATA, NPIX, 8, 1, 0, 0, STATUS )

*   Store the transfer window as a picture in the database
      CALL AGD_SWIND( DISPID, MEMID, XSIZE, YSIZE, XOFF, YOFF,
     :                'DATA', 'IDI transfer window', 0.0, REAL( NX ),
     :                 0.0, REAL( NY ), ID, STATUS )
```

As usual when creating a picture in the database if this new picture lies outside the current picture an error status is returned.

The routine **AGD_NWIND** will return the size and offsets of a window that matches the current picture in the database. The returned arguments should not be used to define a transfer window directly unless it is known that the picture exactly matches the size and shape of the image to be loaded into the transfer window. This is because IDI uses the transfer window to define where the end of one row occurs in the input stream of pixel values. If the transfer window is the wrong shape for the image then the line breaks will not occur in the correct place and the image will appear garbled. When using **AGD_NWIND** to set up a transfer window it is best to compare the returned arguments to the size of the image and set up the transfer window accordingly. The following code segment shows how a transfer window is set up so that the image loads from the bottom left of the picture. The transfer window is set up so that the bottom left corner of the image is aligned with the bottom left corner of the picture, but the size of the transfer window is defined by the size of the image and not the size of the picture.

```
*   Get the size of the current picture from the database
      CALL AGD_NWIND( DISPID, MEMID, XSIZE, YSIZE, XOFF, YOFF, STATUS )

*   Check that the image does not overflow the picture
      IF ( ( NX .LE. XSIZE ) .AND. ( NY .LE. YSIZE ) ) THEN

*   Set up the transfer window to match the size of the image NX * NY
*   and to begin at the bottom left of the picture
         CALL IIMSTW( DISPID, MEMID, 0, NX, NY, 8, XOFF, YOFF, STATUS )

*   Load the image held in the array ADATA
         NPIX = NX * NY
         CALL IIMWMY( DISPID, MEMID, ADATA, NPIX, 8, 1, 0, 0, STATUS )
      ENDIF
```

There is a slight difference between the definition of an IDI window and a picture in the database due to the former being stored as integers and the latter being stored as floating point numbers. This does not matter when saving a window in the database because the integer coordinates are simply expressed as reals in the database, but a picture already defined in the database may have limits that correspond to partial pixels on the screen. In the latter case the pixel coordinates returned by the routine **AGD_NWIND** are rounded up from the picture coordinates so that the window contains the whole picture. This means that the resulting window may be slightly larger (by fractions of a pixel) than the original picture.

There are wrap-up routines to open and close AGI and IDI within the ADAM environment using single calls. The sequence of calls **AGI_ASSOC**, **AGI_BEGIN**, **AGD_ACTIV**, **AGD_NWIND** can be replaced by a single call to **AGD_ASSOC**. As with **AGS_ASSOC** an optional call to **AGI_RCL** can be made by passing a non-blank name argument. The sequence of calls **AGD_DEACT**, **AGI_END**, **AGI_CANCL** or **AGI_ANNUL** can be replaced by a single call to **AGD_DEASS**.

The job of opening and closing the device is handled by the AGD_ routines and therefore the routines **IDI_ASSOC** and **IDI_CANCL** or **IIDOPN** and **IIDCLO** should not be called when using the AGD_ routines.

7 Additional Postscript Features for Native PGPLOT

When AGI is used to open the native PGPLOT graphics system for output to a Postscript file, several extra "pseudo-devices" are recognised that provide additional features over and above the corresponding standard PGPLOT devices. These device names are:

GNS name	PGPLOT name	Description
aps_p	/AVPS	Accumulating EPS, monochrome, portrait
aps_l	/APS	Accumulating EPS, monochrome, landscape
apscol_p	/AVCPS	Accumulating EPS, color, portrait
apscol_l	/ACPS	Accumulating EPS, color, landscape

AGI strips the leading "A" from the above PGPLOT device names to create the device names that are actually used by PGPLOT. The name of the file to receive the Postscript output can be included in the device name as normal (*e.g.* "`fred.ps/ACPS`" or "`apscol_l;fred.ps`").

If one of these devices is used, then the AGI library will automatically concatenate the new Postscript output created by PGPLOT, with any old Postscript in the same file after PGPLOT is closed down. It does this by temporarily changing the name of the output file by adding the string "AGIPS_" to the start of the file name before opening PGPLOT. PGPLOT then writes the new Postscript to this temporary file. When PGPLOT is closed down, AGI appends the contents of this temporary file to the end of any pre-existing file with the specified name[2].

Thus, using these devices allows complex composite Postscript pictures to be created without the need to use external tools such as psmerge (see SUN/164) to merge individual Postscript files.

Some care is taken to keep the size of the merged Postscript file to a minimum. For instance, if a new picture obscures an old picture (and the new picture has an opaque background) then the old picture is not included in the merged Postscript file. Additionally, Postscript files that do not generate any visible output are excluded from the merged file. Such empty files can be created

[2]If no file with the given name already exists, then the temporary file is just renamed by the removal of the "AGIPS_" prefix.

for instance (when using the standard Postscript devices), by the KAPPA:PICSEL command that simply selects a different AGI picture but does not draw anything.

An additional minor feature of this merging process - the `BoundingBox` comment that PGPLOT places at the end of the Postscript output is moved to the start by this process. This allows a wider range of applications to read the resulting Postscript file.

Some Postscript viewing tools such as Okular (see `http://okular.kde.org`) will automatically re-draw the display if the contents of the displayed file changes on disk. Combined with these "accumulating" Postscript devices, this provides a scheme that is similar to the use of a traditional persistent X-window device. A typical scenario could be:

```
% rm pgplot.ps
% kappa
% gdset /acps
% gdclear
% okular pgplot.ps &
% picdef mode=a xpic=2 ypic=2 prefix=a outline=no
% display $KAPPA_DIR/m31 accept
% picsel a2
% linplot $KAPPA_DIR/m31'(,150)' style=def
% linplot clear=no $KAPPA_DIR/m31'(,140)' style='+colour=red'
% picsel a3
% display $KAPPA_DIR/m57 accept
```

The `okular` display will update as each KAPPA command adds additional pictures into the `pgplot.ps` file. Some notes:

(1) The `okular` command was put into the background by the trailing "&" character in order to allow subsequent commands to execute.

(2) The LINPLOT style setting begins with a "+" to indicate that it should be used only for one invocation of LINPLOT.

8 Linking

To include either of the AGI include files, create links in the directory containing the program source code with the command:

```
agi_dev
```

and use and include statement such as:

```
INCLUDE 'AGI_ERR'
INCLUDE 'AGI_PAR'
```

For programs in the ADAM environment one of the the shell scripts `agi_link_adam` or `agp_link_adam` should be used. For example to compile and link an ADAM task called 'task.f' which is designed to use PGPLOT only the following is used:

```
% alink task.f `agp_link_adam`
```

(note the use of the backward quotes).

A standalone program is linked by specifying one of `agi_link` or `agp_link` on the compiler command line. Thus to compile and link a standalone application called 'prog.f' which, this time, uses GKS based graphics the following is used:

```
% f77 prog.f -o prog `agi_link`
```

9 Routine Specifications

AGD_ACTIV
Initialise IDI

Description:

Initialise IDI. This has to be called before any other AGD or IDI routines. An error is returned if this or any other graphics interface is already active.

Invocation:

```
CALL AGD_ACTIV( STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status

AGD_ASSOC
Associate a device with AGI and IDI

Description:

This is a wrap-up routine to associate a device with the AGI database via the ADAM parameter system and open IDI on it. The size and position of an IDI window corresponding to the current picture in the database is returned. This routine calls AGI_ASSOC, AGI_BEGIN , AGD_ACTIV and AGD_NWIND. Also if the name string is not blank then AGI_RCL is called to recall the last picture of that name. This routine should be matched by a closing call to AGD_DEASS.

Invocation:

```
CALL AGD_ASSOC ( PARAM, ACMODE, PNAME, MEMID, PICID, DISPID,
XSIZE, YSIZE, XOFF, YOFF, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)
The name of the ADAM parameter for accessing device names

ACMODE = CHARACTER * (*) (Given)
Access mode for pictures. 'READ', 'WRITE' or 'UPDATE'.

PNAME = CHARACTER * (*) (Given)
Recall last picture of this name if not blank.

MEMID = INTEGER (Given)
IDI Memory identifier.

PICID = INTEGER (Returned)
Picture identifier for current picture on given device.

DISPID = INTEGER (Returned)
IDI Display identifier.

XSIZE = INTEGER (Returned)
X size of window

YSIZE = INTEGER (Returned)
Y size of window

XOFF = INTEGER (Returned)
X offset of window from memory origin

YOFF = INTEGER (Returned)
Y offset of window from memory origin

STATUS = INTEGER (Given and Returned)
The global status.

AGD_DEACT
Close down IDI

Description:

Close down IDI whatever the value of status. This should be called after all AGD and IDI routines.

Invocation:

```
CALL AGD_DEACT( STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status

AGD_DEASS
Deassociate a device from AGI and IDI

Description:

This is a wrap-up routine to deassociate a device from the AGI database and to close down IDI. The picture current when AGD_ASSOC was called is reinstated. This routine calls AGD_DEACT, AGI_END and either AGI_CANCL or AGI_ANNUL. This routine is executed regardless of the given value of status.

Invocation:

```
CALL AGD_DEASS( PARAM, PARCAN, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)
The name of the ADAM parameter associated with the device.

PARCAN = LOGICAL (Given)
If true the parameter given by PARAM is cancelled, otherwise it is annulled.

STATUS = INTEGER (Given and Returned)
The global status.

AGD_NWIND
Define an IDI window from the current picture

Description:

Define an IDI window in the given memory from the current picture. The window coordinates define the size of the rectangular area in pixels and the offset of its bottom left corner from the memory origin. The window is defined as the smallest possible area, made up of whole pixels, that completely contains the picture. If the device associated with the current picture is not already open then this routine will open the device and return the display identifier. Furthermore if the device was opened with 'WRITE' access (in AGI_ASSOC or AGI_OPEN) then the window will be cleared. If the device is already open then the display identifier will be the same as previously and the device will not be cleared.

Invocation:

```
CALL AGD_NWIND( MEMID, DISPID, XSIZE, YSIZE, XOFF, YOFF, STATUS )
```

Arguments:

MEMID = INTEGER (Given)
Memory identifier

DISPID = INTEGER (Returned)
Display identifier

XSIZE = INTEGER (Returned)
X size of window

YSIZE = INTEGER (Returned)
Y size of window

XOFF = INTEGER (Returned)
X offset of window from origin

YOFF = INTEGER (Returned)
Y offset of window from origin

STATUS = INTEGER (Given and Returned)
The global status

AGD_SWIND
Save an IDI window in the database

Description:

Save an IDI window as a picture in the database. The new picture must be equal in size or smaller than the current picture in the database. The window coordinates define the size of the rectangular area in pixels and the offset of its bottom left corner from the memory origin. The name of the picture and a comment are used to identify the picture in the database. The name string has leading blanks removed and is converted to upper case. The world coordinates define the user coordinate system and are saved in the database as given. They should be linear and increasing left to right and bottom to top. If the picture was successfully created then a valid picture identifier is returned and the new picture becomes the current picture.

Invocation:

```
CALL AGD_SWIND( DISPID, MEMID, XSIZE, YSIZE, XOFF, YOFF, PNAME, COMENT,
WX1, WX2, WY1, WY2, PICID, STATUS )
```

Arguments:

DISPID = INTEGER (Given)
Display identifier

MEMID = INTEGER (Given)
Memory identifier

XSIZE = INTEGER (Given)
X size of window (pixels)

YSIZE = INTEGER (Given)
Y size of window (pixels)

XOFF = INTEGER (Given)
X offset of window from origin (pixels)

YOFF = INTEGER (Given)
Y offset of window from origin (pixels)

PNAME = CHARACTER * (*) (Given)
Name of picture

COMENT = CHARACTER * (*) (Given)
Description of picture

WX1 = REAL (Given)
World coordinate of left edge of new picture

WX2 = REAL (Given)
World coordinate of right edge of new picture

WY1 = REAL (Given)
World coordinate of bottom edge of new picture

WY2 = REAL (Given)
World coordinate of top edge of new picture

PICID = INTEGER (Returned)
Picture identifier

STATUS = INTEGER (Given and Returned)
The global status

AGI_ANNUL
Annul a picture identifier

Description:

Annul the picture identifier. If this is the last active identifier then the database is closed. This routine is executed regardless of the given value of status.

Invocation:

```
CALL AGI_ANNUL( PICID, STATUS )
```

Arguments:

PICID = INTEGER (Given)
Picture identifier

STATUS = INTEGER (Given and Returned)
The global status

AGI_ASSOC
Associate an AGI device with an ADAM parameter

Description:

Associate an AGI device with a parameter in the ADAM environment and return an identifier to the current picture. If there are no pictures on the device then a base picture is created and made current. If the size of the display window has changed since a previous database operation the database is cleared and a message sent to the user. The access mode does not affect the database operation, but it is used by the graphics system to determine if the display should be cleared the first time the device is opened; 'READ' and 'UPDATE' access do not clear the display, but 'WRITE' access does.

Invocation:

```
CALL AGI_ASSOC( PARAM, ACMODE, PICID, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)
Name of the parameter used for accessing the device

ACMODE = CHARACTER * (*) (Given)
Access mode: 'READ', 'WRITE' or 'UPDATE'

PICID = INTEGER (Returned)
Identifier for current picture on the given device

STATUS = INTEGER (Given and Returned)
The global status

AGI_BEGIN
Mark the beginning of a new AGI scope

Description:

Mark the beginning of a new AGI scope. This should be matched with a call to AGI_END. Up to eight levels of nested begin-end blocks are allowed.

Invocation:

```
CALL AGI_BEGIN
```

AGI_CANCL
Cancel the ADAM device parameter

Description:

Cancel the association of the ADAM device parameter with AGI. Any picture identifiers associated with this parameter are annulled. This routine is executed regardless of the given value of status.

Invocation:

```
CALL AGI_CANCL( PARAM, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)

Name of the parameter used for accessing the device

STATUS = INTEGER (Given and Returned)

The global status

AGI_CLOSE
Close AGI in non-ADAM environments

Description:

Close AGI in non-ADAM environments. The database file is closed.

Invocation:

```
CALL AGI_CLOSE( STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status

AGI_END
Mark the end of an AGI scope

Description:

Mark the end of an AGI scope. The given picture is made the current one. If the argument is a negative number then the picture current when the matching AGI_BEGIN was called is made current. All identifiers allocated within this begin-end block are annulled. If the last active identifier is annulled the database is closed.

Invocation:

```
CALL AGI_END( PICID, STATUS )
```

Arguments:

PICID = INTEGER (Given)

Picture identifier

STATUS = INTEGER (Given and Returned)

The global status

AGI_GTREF
Get a reference object from a picture

Description:

This returns a reference to a data object which has been stored in the database. The picture identifier signifies which picture to obtain the reference from. If this identifier is negative then the current picture is used. If no reference object is found for the given picture an error is returned. If the reference was created from an HDS locator then an HDS locator is returned, and this should be annulled in the application using REF_ANNUL to ensure the file is properly closed.

Invocation:

```
CALL AGI_GTREF ( PICID, MODE, DATREF, STATUS )
```

Arguments:

PICID = INTEGER (Given)
Picture identifier

MODE = CHARACTER * (*) (Given)
Access mode for object, 'READ', 'WRITE', or 'UPDATE'

DATREF = CHARACTER * (*) (Returned)
String containing reference object

STATUS = INTEGER (Given and Returned)
The global status

AGI_IBASE
Inquire base picture for current device

Description:

A picture identifier for the base picture on the current device is returned. The picture is not selected as the current picture.

Invocation:

```
CALL AGI_IBASE( PICID, STATUS )
```

Arguments:

PICID = INTEGER (Returned)
Identifier for base picture

STATUS = INTEGER (Given and Returned)
The global status

AGI_ICOM
Inquire comment for the current picture

Description:

The comment string for the current picture is returned.

Invocation:

```
CALL AGI_ICOM( COMENT, STATUS )
```

Arguments:

COMENT = CHARACTER * (*) (Returned)
Comment for current picture

STATUS = INTEGER (Given and Returned)
The global status

AGI_ICURP
Inquire the current picture

Description:

The picture identifier of the current picture is returned.

Invocation:

```
CALL AGI_ICURP( PICID, STATUS )
```

Arguments:

PICID = INTEGER (Returned)

Picture identifier

STATUS = INTEGER (Given and Returned)

The global status

AGI_ILAB
Inquire label of a picture

Description:

Inquire the label of a picture referenced by the identifier. If the picture identifier is negative then the current picture is searched. If no label is associated with this picture then a blank string is returned.

Invocation:

```
CALL AGI_ILAB ( PICID, LABEL, STATUS )
```

Arguments:

PICID = INTEGER (Given)
Picture identifier

LABEL = CHARACTER * (AGI__SZLAB) (Returned)
Label string

STATUS = INTEGER (Given and Returned)
The global status

AGI_IMORE
Inquire if a MORE structure exists

Description:

Inquire if a MORE structure exists for the given picture. If the given value for PICID is negative then the current picture is used. The return argument is true if a MORE structure exists for the picture otherwise it is false.

Invocation:

```
CALL AGI_IMORE( PICID, LMORE, STATUS )
```

Arguments:

PICID = INTEGER (Given)
Picture identifier

LMORE = LOGICAL (Returned)
Locator to the transformation structure

STATUS = INTEGER (Given and Returned)
The global status

AGI_INAME
Inquire name of the current picture

Description:

The name of the current picture is returned.

Invocation:

```
CALL AGI_INAME ( PNAME, STATUS )
```

Arguments:

PNAME = CHARACTER * (*) (Returned)
Name of current picture

STATUS = INTEGER (Given and Returned)
The global status

AGI_IPOBS
Is current picture obscured by another?

Description:

Inquire if the current picture is obscured, either totally or partially by another picture. Obscured means that a picture intersects the current picture and was created more recently. If the input value for the picture identifier is negative the current picture is tested against all other overlying pictures; i.e. those created more recently than the current picture. If the picture identifier corresponds to a valid picture then the current one is only tested against the given one.

Invocation:

```
CALL AGI_IPOBS( PICID, LOBS, STATUS )
```

Arguments:

PICID = INTEGER (Given)
Picture identifier.

LOBS = LOGICAL (Returned)
True if picture is obscured, otherwise false.

STATUS = INTEGER (Given and Returned)
The global status

AGI_ISAMD
Inquire if pictures are on same device

Description:

Inquire if the given picture is on the same device as the current picture.

Invocation:

```
CALL AGI_ISAMD( PICID, LSAME, STATUS )
```

Arguments:

PICID = INTEGER (Given)
Picture identifier

LSAME = LOGICAL (Returned)
True if pictures on same device, otherwise false.

STATUS = INTEGER (Given and Returned)
The global status

AGI_ISAMP
Inquire if two pictures are the same

Description:

Inquire if a picture identifier references the same picture as the current picture. The picture referenced by the given picture identifier is compared with the current picture to see if they point to the same picture in the database.

Invocation:

```
CALL AGI_ISAMP( PICID, LSAME, STATUS )
```

Arguments:

PICID = INTEGER (Given)
Picture identifier.

LSAME = LOGICAL (Returned)
True if the pictures are the same, otherwise false.

STATUS = INTEGER (Given and Returned)
The global status.

AGI_ITOBS
Inquire if test points are obscured

Description:

Inquire if the members of the array of test points are obscured by any picture overlying (create more recently than) the current picture. The points are defined in the world coordinate system of the current picture. An array of logical values is returned containing true if the corresponding point is obscured, otherwise false.

Invocation:

```
CALL AGI_ITOBS( NXY, X, Y, LTOBS, STATUS )
```

Arguments:

NXY = INTEGER (Given)
Number of test points

X = REAL(NXY) (Given)
Array of x coordinates of test points

Y = REAL(NXY) (Given)
Array of y coordinates of test points

LTOBS = LOGICAL(NXY) (Returned)
Array of results. True if point is obscured, otherwise false.

STATUS = INTEGER (Given and Returned)
The global status

AGI_IWOCO
Inquire world coordinates of current picture

Description:

Return the world coordinate limits of the current picture.

Invocation:

```
CALL AGI_IWOCO( WX1, WX2, WY1, WY2, STATUS )
```

Arguments:

WX1 = REAL (Returned)
World coordinate of left edge of picture

WX2 = REAL (Returned)
World coordinate of right edge of picture

WY1 = REAL (Returned)
World coordinate of bottom edge of picture

WY2 = REAL (Returned)
World coordinate of top edge of picture

STATUS = INTEGER (Given and Returned)
The global status

AGI_MORE
Return an HDS locator to a MORE structure

Description:

An HDS locator that points to a MORE structure in the database is returned. A MORE structure can be associated with any picture and it can be used to store any application specific information. If the given value for PICID is negative then the current picture is used. If the access mode is 'WRITE' then an empty MORE structure is created if none existed. If there is an existing structure then 'WRITE' mode will erase the existing contents and return a locator to an empty structure. If the access mode is 'READ' or 'UPDATE' then a locator to an existing structure is returned. In this case an error is returned if there is not a MORE structure for the given picture. The database is not responsible for what goes in the MORE structure or how the information is used. The application is also responsible for annulling the returned locator.

Invocation:

```
CALL AGI_MORE( PICID, ACMODE, MORLOC, STATUS )
```

Arguments:

PICID = INTEGER (Given)
Picture identifier

ACMODE = CHARACTER * (*) (Given)
Access mode for MORE structure. 'READ', 'WRITE' or 'UPDATE'.

MORLOC = CHARACTER * (DAT__SZLOC) (Returned)
Locator to the transformation structure

STATUS = INTEGER (Given and Returned)
The global status

AGI_NUPIC
Create a new picture in the database

Description:

Create a new picture in the database. The extent of the new picture is defined in the world coordinate system of the current picture. The world coordinates of this new picture are set to the values passed. The name string has leading blanks removed and is converted to upper case. The new picture is selected as the current one and a picture identifier returned.

Invocation:

```
CALL AGI_NUPIC( WX1, WX2, WY1, WY2, PNAME, COMENT,
NEWX1, NEWX2, NEWY1, NEWY2, PICID, STATUS )
```

Arguments:

WX1 = REAL (Given)
Current world coordinate of left edge of picture

WX2 = REAL (Given)
Current world coordinate of right edge of picture

WY1 = REAL (Given)
Current world coordinate of bottom edge of picture

WY2 = REAL (Given)
Current world coordinate of top edge of picture

PNAME = CHARACTER * (*) (Given)
Name of new picture

COMENT = CHARACTER * (*) (Given)
Comment for new picture

NEWX1 = REAL (Given)
World coordinate of left edge of new picture

NEWX2 = REAL (Given)
World coordinate of right edge of new picture

NEWY1 = REAL (Given)
World coordinate of bottom edge of new picture

NEWY2 = REAL (Given)
World coordinate of top edge of new picture

PICID = INTEGER (Returned)
Picture identifier

STATUS = INTEGER (Given and Returned)
The global status

AGI_OPEN
Open an AGI device in a non-ADAM environment

Description:

Open an AGI device and return an identifier to the current picture. If there are no pictures on the device then a base picture is created and made current. If the size of the display window has changed since a previous database operation the database is cleared and a message sent to the user. The access mode does not affect the database operation, but it is used by the graphics system to determine if the display should be cleared the first time a zone is created; 'READ' and 'UPDATE' access do not clear the display, but 'WRITE' access does.

Invocation:

```
CALL AGI_OPEN( WKNAME, ACMODE, PICID, STATUS )
```

Arguments:

WKNAME = CHARACTER * (*) (Given)
Name of the device to open

ACMODE = CHARACTER * (*) (Given)
Access mode: 'READ', 'WRITE' or 'UPDATE'

PICID = INTEGER (Returned)
Identifier for current picture on the given device

STATUS = INTEGER (Given and Returned)
The global status

AGI_PDEL
Delete all the pictures on the current device

Description:

Delete all the pictures (except the base picture) on the current device. This routine will only execute if the current picture is the base picture, otherwise no action is taken. All picture identifiers associated with this device are released except for the current one.

Invocation:

```
CALL AGI_PDEL( STATUS )
```

Arguments:

STATUS = INTEGER (Given and returned)

The global status.

AGI_PTREF
Store a reference object in a picture

Description:

This creates a reference to a data object in the database. The argument can be either an HDS locator or any character string reference. If the string is a valid HDS locator then a reference is constructed to point to the relevant object, otherwise the string is assumed to be a reference itself and is stored as supplied. The picture identifier signifies which picture to put the reference into. If this identifier is negative then the current picture is used. If a reference already exists for the given picture then an error is returned.

Invocation:

```
CALL AGI_PTREF( DATREF, PICID, STATUS )
```

Arguments:

DATREF = CHARACTER * (*) (Given)
String containing reference object

PICID = INTEGER (Given)
Picture identifier

STATUS = INTEGER (Given and Returned)
The global status

AGI_RCF
Recall first picture of specified name

Description:

Recall the first picture on the current device that has the specified name and lies within the bounds of the current picture. The name string has leading blanks removed and is converted to upper case before being compared. An empty name string (just spaces) results in a search for a picture of any name. This picture becomes the current picture. If no picture fulfills the conditions an error is returned.

Invocation:

```
CALL AGI_RCF( PNAME, PICID, STATUS )
```

Arguments:

PNAME = CHARACTER * (*) (Given)
Name of picture

PICID = INTEGER (Returned)
Picture identifier

STATUS = INTEGER (Given and Returned)
The global status

AGI_RCFP
Recall first picture embracing a position

Description:

Recall the first picture on the current device that has the specified name, embraces the given position and lies within the bounds of the current picture. The name string has leading blanks removed and is converted to upper case before being compared. An empty name string (just spaces) results in a search for a picture of any name. The position has to be given in the world coordinates of the current picture. This picture becomes the current picture. If no picture fulfills the conditions an error is returned.

Invocation:

```
CALL AGI_RCFP( PNAME, X, Y, PICID, STATUS )
```

Arguments:

PNAME = CHARACTER * (*) (Given)
Name of picture

X = REAL (Given)
X position of test point

Y = REAL (Given)
Y position of test point

PICID = INTEGER (Returned)
Picture identifier

STATUS = INTEGER (Given and Returned)
The global status

AGI_RCL
Recall last picture of specified name

Description:

Recall the last picture on the current device that has the specified name and lies within the bounds of the current picture. The name string has leading blanks removed and is converted to upper case before being compared. An empty name string (just spaces) results in a search for a picture of any name. This picture becomes the current picture. If no picture fulfills the conditions an error is returned.

Invocation:

```
CALL AGI_RCL( PNAME, PICID, STATUS )
```

Arguments:

PNAME = CHARACTER * (*) (Given)
Name of picture

PICID = INTEGER (Returned)
Picture identifier

STATUS = INTEGER (Given and Returned)
The global status

AGI_RCLP
Recall last picture embracing a position

Description:

Recall the last picture on the current device that has the specified name, embraces the given position and lies within the bounds of the current picture. The name string has leading blanks removed and is converted to upper case before being compared. An empty name string (just spaces) results in a search for a picture of any name. The position has to be given in the world coordinates of the current picture. This picture becomes the current picture. If no picture fulfills the conditions an error is returned.

Invocation:

```
CALL AGI_RCLP( PNAME, X, Y, PICID, STATUS )
```

Arguments:

PNAME = CHARACTER * (*) (Given)
Name of picture

X = REAL (Given)
X position of test point

Y = REAL (Given)
Y position of test point

PICID = INTEGER (Returned)
Picture identifier

STATUS = INTEGER (Given and Returned)
The global status

AGI_RCP
Recall preceding picture of specified name

Description:

Recall the picture preceding the given one on the current device that has the specified name and lies within the bounds of the current picture. The search is started at the picture identified by the PSTART argument. The name string has leading blanks removed and is converted to upper case before being compared. An empty name string (just spaces) results in a search for a picture of any name. This picture becomes the current picture. If no picture fulfills the conditions an error is returned.

Invocation:

```
CALL AGI_RCP( PNAME, PSTART, PICID, STATUS )
```

Arguments:

PNAME = CHARACTER * (*) (Given)
Name of picture

PSTART = INTEGER (Given)
Identifier of picture starting the search

PICID = INTEGER (Returned)
Picture identifier

STATUS = INTEGER (Given and Returned)
The global status

AGI_RCPP
Recall preceding picture embracing a position

Description:

Recall the picture preceding the given one on the current device that has the specified name, embraces the given position and lies within the bounds of the current picture. The search is started at the picture identified by the PSTART argument. The name string has leading blanks removed and is converted to upper case before being compared. An empty name string (just spaces) results in a search for a picture of any name. This picture becomes the current picture. If no picture in the workstation structure fulfills the conditions an error is returned.

Invocation:

```
CALL AGI_RCPP( PNAME, PSTART, X, Y, PICID, STATUS )
```

Arguments:

PNAME = CHARACTER * (*) (Given)
Name of picture

PSTART = INTEGER (Given)
Identifier of picture starting the search

X = REAL (Given)
X position of test point

Y = REAL (Given)
Y position of test point

PICID = INTEGER (Returned)
Picture identifier

STATUS = INTEGER (Given and Returned)
The global status

AGI_RCS
Recall succeeding picture of specified name

Description:

Recall the picture succeeding the given one on the current device that has the specified name and lies within the bounds of the current picture. The search is started at the picture identified by the PSTART argument. The name string has leading blanks removed and is converted to upper case before being compared. An empty name string (just spaces) results in a search for a picture of any name. This picture becomes the current picture. If no picture fulfills the conditions an error is returned.

Invocation:

```
CALL AGI_RCS( PNAME, PSTART, PICID, STATUS )
```

Arguments:

PNAME = CHARACTER * (*) (Given)
Name of picture

PSTART = INTEGER (Given)
Identifier of picture starting the search

PICID = INTEGER (Returned)
Picture identifier

STATUS = INTEGER (Given and Returned)
The global status

AGI_RCSP
Recall succeeding picture embracing a position

Description:

Recall the picture succeeding the given one on the current device that has the specified name, embraces the given position and lies within the bounds of the current picture. The search is started at the picture identified by the PSTART argument. The name string has leading blanks removed and is converted to upper case before being compared. An empty name string (just spaces) results in a search for a picture of any name. This picture becomes the current picture. If no picture in the workstation structure fulfills the conditions an error is returned.

Invocation:

```
CALL AGI_RCSP( PNAME, PSTART, X, Y, PICID, STATUS )
```

Arguments:

PNAME = CHARACTER * (*) (Given)
Name of picture

PSTART = INTEGER (Given)
Identifier of picture starting the search

X = REAL (Given)
X position of test point

Y = REAL (Given)
Y position of test point

PICID = INTEGER (Returned)
Picture identifier

STATUS = INTEGER (Given and Returned)
The global status

AGI_SELP
Select the given picture as the current one

Description:

Select the given picture as the current one.

Invocation:

```
CALL AGI_SELP( PICID, STATUS )
```

Arguments:

PICID = INTEGER (Given)
Picture identifier

STATUS = INTEGER (Given and Returned)
The global status

AGI_SLAB
Store label in picture

Description:

Store a label in the picture referenced by the identifier. If the picture identifier is negative then the current picture is used to store the label. If a label already exists for the picture then the old one is overwritten. If this label clashes with another on the same device then the existing label will be replaced with a blank string. For comparison purposes the label string has leading blanks removed and is converted to upper case before being processed, although it is stored as supplied. An empty label string will delete any label stored for that picture.

Invocation:

```
CALL AGI_SLAB ( PICID, LABEL, STATUS )
```

Arguments:

PICID = INTEGER (Given)
Picture identifier

LABEL = CHARACTER * (AGI__SZLAB) (Given)
Label string

STATUS = INTEGER (Given and Returned)
The global status

AGI_SROOT
Select the root picture for searching

Description:

The root picture is selected for searching operations. The root picture contains all other pictures (including the base picture) and can be used to recall any picture whether it lies within the current picture or not. This is used to override the usual restriction that a recalled picture must lie within the bounds of the current picture. The root picture is automatically deselected after a call to any of the recall routines AGI_RC ∗ . Recall is the only operation allowed with the root picture, any other operation called while the root picture is selected will use the current picture.

Invocation:

```
CALL AGI_SROOT( STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status

AGI_TCOPY
Copy a transformation structure to the database

Description:

The transformation pointed to by the HDS locator is stored in the database. The picture identifier signifies which picture is to receive the transformation structure. If this identifier is negative then the current picture will be used. If a transformation already exists for this picture then an error will be returned. The supplied transformation should convert data coordinates into the world coordinates of the database picture, as if it had been created with a call to AGI_TNEW.

Invocation:

```
CALL AGI_TCOPY( TRNLOC, PICID, STATUS )
```

Arguments:

TRNLOC = CHARACTER * (DAT__SZLOC) (Given)
Locator to the transformation structure

PICID = INTEGER (Given)
Picture identifier

STATUS = INTEGER (Given and Returned)
The global status

AGI_TDDTW
Transform double precision data to world coordinates

Description:

Transform a set of double precision data coordinates into world coordinates using a transformation stored in the database. The picture identifier signifies which picture contains the required transformation structure. If this identifier is negative then the current picture is used. If no transformation structure is found then the identity transformation is used, i.e. the world coordinates equal the data coordinates.

Invocation:

```
CALL AGI_TDDTW( PICID, NXY, DX, DY, WX, WY, STATUS )
```

Arguments:

PICID = INTEGER (Given)
Picture identifier

NXY = INTEGER (Given)
Number of data points to transform

DX = DBLE(NXY) (Given)
Array of x data coordinates

DY = DBLE(NXY) (Given)
Array of y data coordinates

WX = DBLE(NXY) (Returned)
Array of x world coordinates

WY = DBLE(NXY) (Returned)
Array of y world coordinates

STATUS = INTEGER (Given and Returned)
The global status

AGI_TDTOW
Transform data to world coordinates

Description:

Transform a set of data coordinates into world coordinates using a transformation stored in the database. The picture identifier signifies which picture contains the required transformation structure. If this identifier is negative then the current picture is used. If no transformation structure is found then the identity transformation is used, i.e. the world coordinates equal the data coordinates.

Invocation:

```
CALL AGI_TDTOW( PICID, NXY, DX, DY, WX, WY, STATUS )
```

Arguments:

PICID = INTEGER (Given)
Picture identifier

NXY = INTEGER (Given)
Number of data points to transform

DX = REAL(NXY) (Given)
Array of x data coordinates

DY = REAL(NXY) (Given)
Array of y data coordinates

WX = REAL(NXY) (Returned)
Array of x world coordinates

WY = REAL(NXY) (Returned)
Array of y world coordinates

STATUS = INTEGER (Given and Returned)
The global status

AGI_TNEW
Store a transformation in the database

Description:

The transformation defined by the pseudo-code FORTRAN statements DTOW and WTOD is stored in the database. The picture identifier signifies which picture is to receive the transformation structure. If this identifier is negative then the current picture will be used. If a transformation already exists for this picture then an error will be returned. The number of world variables NCW has to be equal to 2 otherwise an error will be returned. The number of data variables NCD also has to equal 2 in the present implementation.

Invocation:

```
CALL AGI_TNEW ( NCD, NCW, DTOW, WTOD, PICID, STATUS )
```

Arguments:

NCD = INTEGER (Given)
Number of data variables

NCW = INTEGER (Given)
Number of world variables

DTOW = CHARACTER * (*)(NCW) (Given)
Array of forward transformation functions

WTOD = CHARACTER * (*)(NCD) (Given)
Array of inverse transformation functions

PICID = INTEGER (Given)
Picture identifier

STATUS = INTEGER (Given and Returned)
The global status

AGI_TRUNC
Truncate the AGI database file by removing unused space

Description:

This routine attempts to reduce the size of the AGI database file by removing any unused space from the end. The database must be closed before calling this routine (an error is reported otherwise).

Invocation:

```
CALL AGI_TRUNC( STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)
The global status

AGI_TWTDD
Transform double precision world to data coordinates

Description:

Transform a set of double precision world coordinates into data coordinates using a transformation stored in the database. The picture identifier signifies which picture contains the required transformation structure. If this identifier is negative then the current picture is used. If no transformation structure is found then the identity transformation is used, i.e. the data coordinates equal the world coordinates.

Invocation:

```
CALL AGI_TWTDD( PICID, NXY, WX, WY, DX, DY, STATUS )
```

Arguments:

PICID = INTEGER (Given)
Picture identifier

NXY = INTEGER (Given)
Number of data points to transform

WX = DBLE(NXY) (Given)
Array of x world coordinates

WY = DBLE(NXY) (Given)
Array of y world coordinates

DX = DBLE(NXY) (Returned)
Array of x data coordinates

DY = DBLE(NXY) (Returned)
Array of y data coordinates

STATUS = INTEGER (Given and Returned)
The global status

AGI_TWTOD
Transform world to data coordinates

Description:

Transform a set of world coordinates into data coordinates using a transformation stored in the database. The picture identifier signifies which picture contains the required transformation structure. If this identifier is negative then the current picture is used. If no transformation structure is found then the identity transformation is used, i.e. the data coordinates equal the world coordinates.

Invocation:

```
CALL AGI_TWTOD( PICID, NXY, WX, WY, DX, DY, STATUS )
```

Arguments:

PICID = INTEGER (Given)
Picture identifier

NXY = INTEGER (Given)
Number of data points to transform

WX = REAL(NXY) (Given)
Array of x world coordinates

WY = REAL(NXY) (Given)
Array of y world coordinates

DX = REAL(NXY) (Returned)
Array of x data coordinates

DY = REAL(NXY) (Returned)
Array of y data coordinates

STATUS = INTEGER (Given and Returned)
The global status

AGP_ACTIV
Initialise PGPLOT

Description:

Initialise PGPLOT. This has to be called before any other AGP or PGPLOT routines. An error is returned if this or any other graphics interface, other than AGS_, is active.

Invocation:

```
CALL AGP_ACTIV( STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status

AGP_ASSOC
Associate a device with AGI and PGPLOT

Description:

This is a wrap-up routine to associate a device with the AGI database via the ADAM parameter system and open PGPLOT on it. A PGPLOT viewport corresponding to the current picture in the database is created. This routine calls AGI_ASSOC, AGI_BEGIN AGP_ACTIV and AGP_NVIEW. Also if the name string is not blank then AGI_RCL is called to recall the last picture of that name. This routine should be matched by a closing call to AGP_DEASS.

Invocation:

```
CALL AGP_ASSOC( PARAM, ACMODE, PNAME, BORDER, PICID, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)
The name of the ADAM parameter for accessing device names

ACMODE = CHARACTER * (*) (Given)
Access mode for pictures. 'READ', 'WRITE' or 'UPDATE'.

PNAME = CHARACTER * (*) (Given)
Recall last picture of this name if not blank.

BORDER = LOGICAL (Given)
Flag to indicate if a border is to be left around the viewport.

PICID = INTEGER (Returned)
Picture identifier for current picture on given device.

STATUS = INTEGER (Given and Returned)
The global status.

AGP_DEACT
Close down PGPLOT

Description:

Close down PGPLOT whatever the value of status. This should be called after all AGP and PGPLOT routines.

Invocation:

```
CALL AGP_DEACT( STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status

AGP_DEASS
Deassociate a device from AGI and PGPLOT

Description:

This is a wrap-up routine to deassociate a device from the AGI database and to close down PGPLOT. The picture current when AGP_ASSOC was called is reinstated. This routine calls AGP_DEACT, AGI_END and either AGI_CANCL or AGI_ANNUL. This routine is executed regardless of the given value of status.

Invocation:

```
CALL AGP_DEASS( PARAM, PARCAN, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)
The name of the ADAM parameter associated with the device.

PARCAN = LOGICAL (Given)
If true the parameter given by PARAM is cancelled, otherwise it is annulled.

STATUS = INTEGER (Given and Returned)
The global status.

AGP_NVIEW
Create a new PGPLOT viewport from the current picture

Description:

Create a new PGPLOT viewport from the current picture. The viewport will be created with the coordinate system of the current picture. The border flag allocates space around the plot for annotation if required. If true the viewport is made approximately 10% smaller than the picture to allow space for annotation. If false the viewport matches the picture exactly. If the device associated with the current picture is not open then this routine will open it, and additionally the viewport will be cleared if the access mode is 'WRITE' (in AGI_ASSOC or AGI_OPEN). If the SGS interface is already active then the first call to this routine will open PGPLOT in the current SGS zone. In this case the viewport normalised device coordinates will not match the coordinate system of the current picture, but will have the default range of 0 to 1.

Invocation:

```
CALL AGP_NVIEW ( BORDER, STATUS )
```

Arguments:

BORDER = LOGICAL (Given)
Flag to indicate if a border is to be left around the viewport

STATUS = INTEGER (Given and Returned)
The global status

AGP_SVIEW
Save the current PGPLOT viewport in the database

Description:

Save the current PGPLOT viewport as a picture in the database. The new picture must be equal in size or smaller than the current picture in the database. The name of the picture and a comment are used to identify the picture in the database. The name string has leading blanks removed and is converted to upper case. If the picture was successfully created then a valid picture identifier is returned and the new picture becomes the current picture.

Invocation:

```
CALL AGP_SVIEW( PICNAM, COMENT, PICID, STATUS )
```

Arguments:

PICNAM = CHARACTER * (*) (Given)
Name of picture

COMENT = CHARACTER * (*) (Given)
Description of picture

PICID = INTEGER (Returned)
Picture identifier

STATUS = INTEGER (Given and Returned)
The global status

AGS_ACTIV
Initialise SGS

Description:

Initialise SGS. This has to be called before any other AGS or SGS routines. An error is returned if this or any other graphics interface is already active.

Invocation:

```
CALL AGS_ACTIV( STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status

AGS_ASSOC
Associate a device with AGI and SGS

Description:

This is a wrap-up routine to associate a device with the AGI database via the ADAM parameter system and open SGS on it. An SGS zone corresponding to the current picture in the database is created. This routine calls AGI_ASSOC , AGI_BEGIN, AGS_ACTIV and AGS_NZONE. Also if the name string is not blank then AGI_RCL is called to recall the last picture of that name. This routine should be matched by a closing call to AGS_DEASS.

Invocation:

```
CALL AGS_ASSOC( PARAM, ACMODE, PNAME, PICID, NEWZON, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)
The name of the ADAM parameter for accessing device names

ACMODE = CHARACTER * (*) (Given)
Access mode for pictures. 'READ', 'WRITE' or 'UPDATE'.

PNAME = CHARACTER * (*) (Given)
Recall last picture of this name if not blank.

PICID = INTEGER (Returned)
Picture identifier for current picture on given device.

NEWZON = INTEGER (Returned)
The new SGS zone that matches the current picture.

STATUS = INTEGER (Given and Returned)
The global status.

AGS_DEACT
Close down SGS

Description:

Close down SGS whatever the value of status. This should be called after all AGS and SGS routines.

Invocation:

```
CALL AGS_DEACT( STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status

AGS_DEASS
Deassociate a device from AGI and SGS

Description:

This is a wrap-up routine to deassociate a device from the AGI database and to close down SGS. The picture current when AGS_ASSOC was called is reinstated. This routine calls AGS_DEACT, AGI_END and either AGI_CANCL or AGI_ANNUL. This routine is executed regardless of the given value of status.

Invocation:

```
CALL AGS_DEASS( PARAM, PARCAN, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)
The name of the ADAM parameter associated with the device.

PARCAN = LOGICAL (Given)
If true the parameter given by PARAM is cancelled, otherwise it is annulled.

STATUS = INTEGER (Given and Returned)
The global status.

AGS_NZONE
Create a new SGS zone from the current picture

Description:

Create a new SGS zone from the current picture. The zone will be created with the coordinate system of the current picture. If the device associated with the current picture is not already open then this routine will open it, and additionally the zone will be cleared if the access mode is 'WRITE' (in AGI_ASSOC or AGI_OPEN).

Invocation:

```
CALL AGS_NZONE ( NEWZON, STATUS )
```

Arguments:

NEWZON = INTEGER (Returned)
SGS zone identifier of new zone

STATUS = INTEGER (Given and Returned)
The global status

AGS_SZONE
Save the current SGS zone in the database

Description:

Save the current SGS zone as a picture in the database. The new picture must be equal in size or smaller than the current picture in the database. The name of the picture and a comment are used to identify the picture in the database. The name string has leading blanks removed and is converted to upper case. If the picture was successfully created then a valid picture identifier is returned and the new picture becomes the current picture.

Invocation:

```
CALL AGS_SZONE( PNAME, COMENT, PICID, STATUS )
```

Arguments:

PNAME = CHARACTER * (*) (Given)
Name of picture

COMENT = CHARACTER * (*) (Given)
Description of picture

PICID = INTEGER (Returned)
Picture identifier

STATUS = INTEGER (Given and Returned)
The global status