SUN/263.1

Starlink Project
Starlink User Note 263.1

Malcolm J. Currie
D.S. Berry

2009 August 16

LPG — Loop processing of groups Version 3.0 Programmer's Manual

Abstract

This document describes the routines provided within the LPG subroutine library for looping of monolith tasks to process a group of catalogues or NDFs in sequence.

Contents

List of Figures

1 Introduction

When an application prompts the user for a catalogue or NDF using the facilities of the CAT (see SUN/181) or NDF (see SUN/33) libraries, the user may only reply with the name of a single catalogue or NDF respectively. If the user has many files to process in the same fashion, it can prove tedious to repeat the commands for each input dataset. Now one solution is to write a script that loops, executing the various applications for each input file. More elegant and convenient to users would be to allow a *group* of files to be processed in a single command. This is what LPG offers. The group or list is supplied to the relevant ADAM parameter, possibly defined using wildcards. Using LPG enhances your application package to users.

Note the rôle of LPG is different from NDG or CTG libraries. While these supply groups of NDFs and catalogues, again with wildcards (via the underlying GRP library) these are processed in the *same invocation* of an application. Examples of this method include forming a flat field from a series of CCD image NDFs, and merging catalogues.

2 Interaction Between LPG and GRP

LPG uses the facilities of the GRP package and programmers incorporating LPG should be familiar with the content of SUN/150 which describes the GRP package. Examples of the GRP wildcards and indirection through text files are presented in SUN/95, and in addition advice for users.

3 Using LPG

To introduce the looping facility into an applications package a number of steps are required. These affect the monolith routine, individual applications and possibly their interface files, and documentation.

3.1 Monolith

You must modify both the monolith to loop. The basic arrangement is shown below.

```
*  External References:
      LOGICAL LPG_AGAIN          ! Invoke the application again?

           :           :           :           :           :

*  Obtain the command from the environment.  This returns uppercase
*  names.
      CALL TASK_GET_NAME( ACTION, STATUS )

*  Initialise the common blocks used to control multiple invocation of
*  applications to process lists of NDFs or catalogues.
      CALL LPG_START( VERB, DELAY, DISAB, STATUS )

*  Loop round invoking the task for each set of NDFS or catalogues
*  specified by the user.
      DO WHILE ( LPG_AGAIN( STATUS ) )

         IF ( ACTION .EQ. 'ADD' ) THEN
            CALL ADD( STATUS )

         ELSE IF ( ACTION .EQ. 'BIND' ) THEN
            CALL BIND( STATUS )

         ELSE IF...
            ...
         END IF
      END DO
```

LPG_AGAIN returns `.TRUE.` value until the list of data files is exhausted.

The additional code is the LPG_START call, the testing of LPG_AGAIN for any further files to process in a DO WHILE . . . END DO loop (or use IF .. END IF with a GOTO if you prefer), and the declaration of LPG_AGAIN.

3.2 Tuning

LPG_START has three tuning arguments.

- VERB set to `.TRUE.` causes multi-valued parameters (*i.e.* ones for accessing data files) to report their value at each invocation; in essence this presents the names of the data file at each invocation. Single-valued parameters are not shown. Set

- A pause of DELAY seconds occurs betwen invocations for each NDF or catalogue.
- DISAB set to `.TRUE.` disables the looping. Thus LPG_AGAIN would only return `.TRUE.` at the first invocation. Thus the package behaves as if LPG looping was not present. The application corresponding to the required action will always be invoked at least once.

  These tuning options are best controlled through environment variables accessed in the monolith. For example, KAPPA invokes KAPLIBS calls

```
*  See if NDF names should be reported when looping.
      CALL KPG1_ENVDF( 'KAPPA_REPORT_NAMES', VERB, STATUS )

*  See if looping should be disabled.
      CALL KPG1_ENVDF( 'KAPPA_LOOP_DISABLE', DISAB, STATUS )

*  See if a delay should be included between invocations.
      DELAY = 0.0
      CALL KPG1_ENV0R( 'KAPPA_LOOP_DELAY', DELAY, STATUS )
```

  where KPG1_ENVDF inquires whether an environment variable is defined or not, and KPG1_ENV0R obtains a floating-point value, but using the default of 0.0 seconds should `KAPPA_LOOP_DELAY` be undefined.

There is a further tuning possibility. Some users like to be able supply the same data for output as input, although this is potentially hazardous. Here is another extract from KAPPA showing how this is switched using LPG_REPLA.

```
*  See if input NDFs are allowed to be overwritten by output NDFs.
      CALL KPG1_ENVDF( 'KAPPA_REPLACE', REPL, STATUS )
      CALL LPG_REPLA( REPL, STATUS )
```

Variable REPL is boolean. The environment variable need just have a value, any value for this switch to be enabled.

3.3 Applications

The applications should use the routines LPG_ASSOC, LPG_PROP, LPG_CREAT, and LPG_CREP to get identifier for NDFs, in place of the corresponding routines (*i.e.* replace LPG with NDF in the names) from the NDF library.

For catalogues, routines LPG_CATASSOC and LPG_CATCREAT should be used in place of CAT_ASSOC and CAT_CREAT.

On the first invocation of the application, groups of data files are obtained whenever one of the above LPG routines is used to get an NDF or CAT identifier, and an identifier corresponding to the first name in each group is returned to the application. On subsequent invocations, the names in the groups obtained during the first invocation are used without obtaining new parameter values from the environment. The index of the returned data file within each group is incremented by 1 each time the application is invoked.

3.4 Other Parameters

If an application is invoked more than once, all other parameters retain the values they had at the end of the first invocation. Applications that use this scheme should avoid having parameters with `VPATH=DYNAMIC` in the interface file (described in SUN/115), since the dynamic default calculated on the first invocation will then be re-used for all subsequent invocations; that may be inappropriate. A better scheme is to have `VPATH=DEFAULT, PPATH=DYNAMIC` and `DEFAULT=!`. The code should then annul any `PAR__NULL` status after accessing the parameter, and use the previously calculated dynamic default value for the parameter. In this scheme, the parameter value is `!` at the end of the first invocation, and so retains this value for all subsequent invocations, resulting in appropriate dynamic defaults being used.

A situation in which the above suggestion does not work is if an application sometimes sets a dynamic default, and sometimes does not. In this case, you do not want to have `VPATH=DEFAULT, DEFAULT=!` because this would require the application to abort in the cases where there is no dynamic default available. It is probably better in these cases to have `VPATH=PROMPT, PPATH=DYNAMIC` and accept the fact that the user will be prompted for a parameter that was previously defaulted.

Some applications test to see if a parameter was specified on the command line, and vary their behaviour accordingly. This is achieved by checking the state of the parameter before accessing it, a state of `PAR__ACTIVE` (or `SUBPAR__ACTIVE`) indicating that the parameter already has a value. This is correct on the first invocation, but not on subsequent invocations because the first invocation may have set a parameter value, resulting in subsequent invocations thinking that the parameter was given on the command line. To avoid this, applications should call LPG_STATE in place of PAR_STATE. LPG_STATE remembers the state of the parameter on the first invocation, and returns that state, rather than the current parameter state, on subsequent invocations. The arguments are the same.

3.5 Output Parameters

One disadvantage of LPG is that any parameters written by the application, such the results of some analysis or statistics, will only record the values for the *last* data file processed.

4 Compiling and Linking with LPG

This section describes how to compile and link applications that use LPG subroutines, on UNIX systems. It is assumed that the LPG library is installed as part of the Starlink Software Collection.

The library only has an ADAM interface to obtain the groups of catalogues.

4.1 ADAM Applications

Users of the ADAM programming environment (SG/4) should use the **alink** command (SUN/144) to compile and link applications, and can access the LPG_ library by including execution of the command `lpg_link_adam` on the command line, as follows:

```
% alink prog.f ‘lpg_link_adam‘
```

where `prog.f` is the Fortran source file for the A-TASK. Again note the use of opening apostrophies (‘) instead of the more usual closing apostrophy (’) in the above **alink** command.

To build a program written in C (instead of Fortran), simply name the source file `prog.c`, instead of `prog.f`.

A List of Routines

RESULT = LPG_AGAIN(STATUS)
Decide if the application should be executed again.

CALL LPG_ASSOC(PARAM, MODE, INDF, STATUS)
Obtain an identifier for an existing NDF via the parameter system.

CALL LPG_CATASSOC(PARAM, MODE, CI, STATUS)
Obtain an identifier for an existing catalogue via the parameter system.

CALL LPG_CATCREAT(PARAM, CI, STATUS)
Creat a new catalogue via the parameter system.

CALL LPG_CREA1(PARAM, FTYPE, NDIM, LBND, UBND, INDF, NAME, STATUS)
Create a single new simple NDF using a specified parameter.

CALL LPG_CREAT(PARAM, FTYPE, NDIM, LBND, UBND, INDF, STATUS)
Create a new simple NDF via the parameter system.

CALL LPG_CREP1(PARAM, FTYPE, NDIM, UBND, INDF, NAME, STATUS)
Create a single new primitive NDF using a specified parameter.

CALL LPG_CREP(PARAM, FTYPE, NDIM, UBND, INDF, STATUS)
Create a new primitive NDF via the parameter system.

CALL LPG_CRPL1(PARAM, PLACE, NAME, STATUS)
Create a single new NDF placeholder using a specified parameter

CALL LPG_PROP1(INDF1, CLIST, PARAM, INDF2, NAME, STATUS)
Create a single new NDF by propagation using a specified parameter.

CALL LPG_PROP(INDF1, CLIST, PARAM, INDF2, STATUS)
Propagate NDF information to create a new NDF via the parameter system.

CALL LPG_REPLA(REPLAC, STATUS)
Indicate if input NDFs can be replaced.

CALL LPG_START(VERBO, DELAYO, DISABO, STATUS)
Initialise the contents of the LPG common blocks.

CALL LPG_STATE(PARAM, STATE, STATUS)
Return the original PAR state of a parameter.

B Full Fortran Routine Specifications

LPG_AGAIN
Decide if the application should be executed again

Description:

This routine is used to allow multiple invocations of an application within an Starlink monolith to process a group of data files. The initialization routine LPG_START should be called prior to this routine. This routine returns a logical flag indicating if the application should be invoked again. A typical way to use this routine within a monolith is as follows:

```
CALL LPG_START( VERB, DELAY, DISAB, STATUS )
DO WHILE ( LPG_AGAIN( STATUS ) )
   IF ( ACTION .EQ. 'ADD' ) THEN
      CALL ADD( STATUS )
   ELSE IF ( ACTION .EQ. 'SUB' ) THEN
      CALL SUB( STATUS )
   ELSE IF...
      ...
   END IF
END DO
```

The application corresponding to the required action will always be invoked once. The applications should use the routines LPG_ASSOC, LPG_PROP, LPG_CREAT and LPG_CREP to get identifiers for NDFs, in place of their equivalent routines from the NDF library.

For catalogues, routines LPG_CATASSOC and LPG_CATCREAT should be used in place of CAT_ASSOC and CAT_CREAT.

LPG_AGAIN returns a `.TRUE.` value until a group of data files is exhausted, whereupon it deletes all its groups and returns a `.FALSE.` value.

On the first invocation of the application, groups of data files are obtained whenever one of the above LPG routines is used to get an NDF or CAT identifier, and an identifier corresponding to the first name in each group is returned to the application. On subsequent invocations, the names in the groups obtained during the first invocation are used without obtaining new parameter values from the environment. The index of the returned data file within each group is increment by 1 each time the application is invoked.

If an application is invoked more than once, all other parameters retain the values they had at the end of the first invocation. Applications that use this scheme should avoid having parameters with `"VPATH=DYNAMIC"` in the interace file, since the dynamic default calculated on the first invocation will then be re-used for all subsequent invocations, which may be inappropriate. A better scheme is to have `"VPATH=DEFAULT"`, `"PPATH=DYNAMIC"` and `"DEFAULT=!"`. The code should then annul any PAR__NULL status after accessing the parameter, and use the previously calculated dynamic default value for the parameter. With this scheme, the parameter value is `"!"` at the end of the first invocation, and so retains this value for all subsequent invocations, resulting in appropriate dynamic defaults being used.

A situation in which the above suggestion does not work is if an application sometimes sets a dynamic default, and sometimes does not. In this case, you do not want to have `VPATH=DEFAULT,DEFAULT=!` because this would require the application to abort in the cases where there is no dynamic default available. It is probably better in these cases to have `VPATH=PROMPT,PPATH=DYNAMIC` and accept the fact that the user will be prompted for a parameter that was previously defaulted.

Some applications test to see if a parameter was specified on the command line, and vary their behaviour accordingly. This is done by checking the state of the parameter before accessing it, a state of PAR__ACTIVE (or SUBPAR__ACTIVE) indicating that the parameter already has a value. This is correct on the first invocation, but not on subsequent invocations because the first invocation may have set a parameter value, resulting in subsequent invocations thinking that the parameter was given on the command line. To avoid this, applications should use LPG_STATE in place of PAR_STATE. LPG_STATE remembers the state of the parameter on the first invocation, and returns that state, rather than the current parameter state, on subsequent invocations.

Invocation:

```
RESULT = LPG_AGAIN( STATUS )
```

Arguments:

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

Returned Value:

LPG_AGAIN = LOGICAL
This is `.TRUE.` if the application should be executed again.

LPG_ASSOC
Obtain an identifier for an existing NDF via the parameter system

Description:

This routine should be called in place of NDF_ASSOC within applications that process groups of NDFs.

On the first invocation of the application, a group of names of existing NDFs will be obtained from the environment using the specified parameter, and an NDF identifier for the first one will be returned. If more than one NDF was supplied for the parameter then the application may be invoked again (see LPG_AGAIN), in which case this routine will return an identifier for the next NDF in the group supplied on the first invocation.

If an application attempts to get a new NDF by cancelling the parameter (PAR_CANCL), the returned NDF is *NOT* the next one in the group, but is obtained by prompting the user for a single NDF.

The monolith routine should arrange to invoke the application repeatedly until one or more of its NDF parameters have been exhausted (*i.e.* all its values used). See LPG_AGAIN.

Invocation:

```
CALL LPG_ASSOC( PARAM, MODE, INDF, STATUS )
```

Arguments:

PARAM = CHARACTER $*$ ($*$) (Given)
Name of the parameter.

MODE = CHARACTER $*$ ($*$) (Given)
Type of NDF access required: `'READ'`, `'UPDATE'` or `'WRITE'`.

INDF = INTEGER (Returned)
NDF identifier.

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

LPG_CATASSOC
Obtain an identifier for an existing catalogue via the parameter system

Description:

This routine should be called in place of CAT_ASSOC within applications that process groups of catalogues.

On the first invocation of the application, a group of names of existing catalogues will be obtained from the environment using the specified parameter, and a CAT identifier for the first one will be returned. If more than one catalogue was supplied for the parameter then the application may be invoked again (see LPG_AGAIN), in which case this routine will return an identifier for the next catalogue in the group supplied on the first invocation.

If an application attempts to get a new catalogue by cancelling the parameter (PAR_CANCL), the returned catalogue is `NOT` the next one in the group, but is obtained by prompting the user for a single catalogue.

The monolith routine should arrange to invoke the application repeatedly until one or more of its catalogue parameters have been exhausted (*i.e.* all its values used). See LPG_AGAIN.

Invocation:

```
CALL LPG_CATASSOC( PARAM, MODE, CI, STATUS )
```

Arguments:

PARAM = CHARACTER $*$ ($*$) (Given)
Name of the parameter.

MODE = CHARACTER $*$ ($*$) (Given)
Type of catalogue access required: `'READ'`, or `'WRITE'`.

CI = INTEGER (Returned)
The catalogue identifier.

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

LPG_CATCREAT
Create a new catalogue via the parameter system

Description:

This routine should be called in place of CAT_CREAT within applications that process lists of catalogues.

On the first invocation of the application, a group of names for some new catalogues will be obtained from the environment using the specified parameter. The first name will be used to create an catalogue with the requested attributes, and an identifier for the new catalogue will be returned. If more than one name was supplied for the parameter then the application may be invoked again (see LPG_AGAIN), in which case this routine will return an identifier for a new catalogue with the next name in the group supplied on the first invocation.

If a modification element is included in the group expression supplied for the parameter on the first invocation of the application, the new catalogue names are based on the names of the first group of existing data files (catalogues or NDFs) to be accessed by the application.

If an application attempts to get a new catalogue by cancelling the parameter (PAR_CANCL), the name used to create the returned catalogue is *NOT* the next one in the group, but is obtained by prompting the user for a single new catalogue.

The monolith routine should arrange to invoke the application repeatedly until one or more of its catalogue parameters have been exhausted (*i.e.* all its values used). See LPG_AGAIN.

Invocation:

```
CALL LPG_CATCREAT( PARAM, CI, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)
Name of the parameter.

CI = INTEGER (Returned)
The catalogue identifier.

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

LPG_CREA1
Create a single new simple NDF using a specified parameter

Description:

This routine is equivalent to NDF_CREAT except that it allows the NDF to be specified using a GRP group expression (for instance, its name may be given within a text file, *etc.*). The first NDF in the group expression is returned. Any other names in the group expression are ignored. Any modification elements in the supplied group expression will be treated literally.

Invocation:

```
CALL LPG_CREA1( PARAM, FTYPE, NDIM, LBND, UBND, INDF, NAME, STATUS )
```

Arguments:

PARAM = CHARACTER $*$ ($*$) (Given)
Name of the ADAM parameter.

FTYPE = CHARACTER $*$ ($*$) (Given)
Full data type of the NDF's DATA component (e.g. `'_DOUBLE'` or `'COMPLEX_REAL'`).

NDIM = INTEGER (Given)
Number of NDF dimensions.

LBND(NDIM) = INTEGER (Given)
Lower pixel-index bounds of the NDF.

UBND(NDIM) = INTEGER (Given)
Upper pixel-index bounds of the NDF.

INDF = INTEGER (Returned)
NDF identifier.

NAME = CHARACTER $*$ ($*$) (Returned)
The full file specification for the NDF.

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

LPG_CREAT
Create a new simple NDF via the parameter system

Description:

This routine should be called in place of NDF_CREAT within applications which process lists of NDFs.

On the first invocation of the application, a group of names for some new NDFs will be obtained from the environment using the specified parameter. The first name will be used to create an NDF with the requested attributes, and an identifier for the new NDF will be returned. If more than one name was supplied for the parameter then the application may be invoked again (see LPG_AGAIN), in which case this routine will return an identifier for a new NDF with the next name in the group supplied on the first invocation.

If a modification element is included in the group expression supplied for the parameter on the first invocation of the application, the new NDF names are based on the names of the first group of existing data files (NDFs or catalogues) to be accessed by the application.

If an application attempts to get a new NDF by cancelling the parameter (PAR_CANCL), the name used to create the returned NDF is*NOT* the next one in the group, but is obtained by prompting the user for a single new NDF.

The monolith routine should arrange to invoke the application repeatedly until one or more of its NDF parameters have been exhausted (*i.e.* all its values used). See LPG_AGAIN.

Invocation:

```
CALL LPG_CREAT( PARAM, FTYPE, NDIM, LBND, UBND, INDF, STATUS )
```

Arguments:

PARAM = CHARACTER ∗ (∗) (Given)
Name of the parameter.

FTYPE = CHARACTER ∗ (∗) (Given)
Full data type of the NDF's DATA component (e.g. `'_DOUBLE'` or `'COMPLEX_REAL'`).

NDIM = INTEGER (Given)
Number of NDF dimensions.

LBND(NDIM) = INTEGER (Given)
Lower pixel-index bounds of the NDF.

UBND(NDIM) = INTEGER (Given)
Upper pixel-index bounds of the NDF.

INDF = INTEGER (Returned)
NDF identifier.

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

LPG_CREP1
Create a single new primitive NDF using a specified parameter

Description:

This routine is equivalent to NDF_CREP except that it allows the NDF to be specified using a GRP group expression (for instance, its name may be given within a text file, *etc.*). The first NDF in the group expression is returned. Any other names in the group expression are ignored. Any modification elements in the supplied group expression will be treated literally.

Invocation:

```
CALL LPG_CREP1( PARAM, FTYPE, NDIM, UBND, INDF, NAME, STATUS )
```

Arguments:

PARAM = CHARACTER ∗ (∗) (Given)
Name of the ADAM parameter.

FTYPE = CHARACTER ∗ (∗) (Given)
Type of the NDF's DATA component (e.g. `'_REAL'`). Note that complex types are not permitted when creating a primitive NDF.

NDIM = INTEGER (Given)
Number of NDF dimensions.

UBND(NDIM) = INTEGER (Given)
Upper pixel-index bounds of the NDF (the lower bound of each dimension is taken to be 1).

INDF = INTEGER (Returned)
NDF identifier.

NAME = CHARACTER ∗ (∗) (Returned)
The full file specification for the NDF.

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

LPG_CREP
Create a new primitive NDF via the parameter system

Description:

This routine should be called in place of NDF_CREP within applications that process lists of NDFs.

On the first invocation of the applicaton, a group of names for some new NDFs will be obtained from the environment using the specified parameter. The first name will be used to create an NDF with the requested attributes, and an identifier for the new NDF will be returned. If more than one name was supplied for the parameter then the application may be invoked again (see LPG_AGAIN), in which case this routine will return an identifier for a new NDF with the next name in the group supplied on the first invocation.

If a modification element is included in the group expression supplied for the parameter on the first invocation of the application, the new NDF names are based on the names of the first group of existing data files (NDFs or catalogues) to be accessed by the application.

If an application attempts to get a new NDF by cancelling the parameter (PAR_CANCL), the name used to create the returned NDF is NOT the next one in the group, but is obtained by prompting the user for a single new NDF.

The monolith routine should arrange to invoke the application repeatedly until one or more of its NDF parameters have been exhausted (i.e. all its values used). See NDF_AGAIN.

Invocation:

```
CALL LPG_CREP( PARAM, FTYPE, NDIM, UBND, INDF, STATUS )
```

Arguments:

PARAM = CHARACTER $*$ ($*$) (Given)
Name of the parameter.

FTYPE = CHARACTER $*$ ($*$) (Given)
Type of the NDF's DATA component (e.g. '_REAL'). Note that complex types are not permitted when creating a primitive NDF.

NDIM = INTEGER (Given)
Number of NDF dimensions.

UBND(NDIM) = INTEGER (Given)
Upper pixel-index bounds of the NDF (the lower bound of each dimension is taken to be 1).

INDF = INTEGER (Returned)
NDF identifier.

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

LPG_CREPL
Create a new NDF placeholder via the parameter system

Description:

This routine should be called in place of NDF_CREPL within applications that process lists of NDFs.

On the first invocation of the application, a group of names for some new NDFs will be obtained from the environment using the specified parameter. The first name will be used to create an NDF placeholder with the requested attributes, and an identifier for the placeholder will be returned. If more than one name was supplied for the parameter then the application may be invoked again (see LPG_AGAIN), in which case this routine will return an identifier for another placeholder with the next name in the group supplied on the first invocation.

If a modification element is included in the group expression supplied for the parameter on the first invocation of the application, the placeholder are based on the names of the first group of existing data files (NDFs or catalogues) to be accessed by the application.

If an application attempts to get a new NDF by cancelling the parameter (PAR_CANCL), the name used to create the returned NDF is NOT the next one in the group, but is obtained by prompting the user for a single new placeholder.

The monolith routine should arrange to invoke the application repeatedly until one or more of its NDF parameters have been exhausted (i.e. all its values used). See LPG_AGAIN.

Invocation:

```
CALL LPG_CREPL( PARAM, PLACE, STATUS )
```

Arguments:

PARAM = CHARACTER ∗ (∗) (Given)
Name of the parameter.

PLACE = INTEGER (Returned)
NDF placeholder identifying the nominated position in the data system.

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

LPG_PROP1
Create a single new NDF by propagation using a specified parameter

Description:

This routine is equivalent to NDF_PROP except that it allows the NDF to be specified using a GRP group expression (for instance, its name may be given within a text file, *etc.*). The first NDF in the group expression is returned. Any other names in the group expression are ignored. Modification elements use the name of the supplied NDF as the basis name.

Invocation:

```
CALL LPG_PROP1( INDF1, CLIST, PARAM, INDF2, NAME, STATUS )
```

Arguments:

INDF1 = INTEGER (Given)
Identifier for an existing NDF (or NDF section) to act as a template.

CLIST = CHARACTER * (*) (Given)
A comma-separated list of the NDF components which are to be propagated to the new data structure. By default, the HISTORY, LABEL and TITLE components and all extensions are propagated. See the "Component Propagation" section in the documentation for routine NDF_PROP within SUN/33 for further details.

PARAM = CHARACTER * (*) (Given)
Name of the ADAM parameter for the new NDF.

INDF2 = INTEGER (Returned)
Identifier for the new NDF.

NAME = CHARACTER * (*) (Returned)
The full file specification for the NDF.

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

LPG_PROP
Propagate NDF information to create a new NDF via the parameter system

Description:

This routine should be called in place of NDF_PROP within applications that process groups of NDFs.

On the first invocation of the application, a group of names for some new NDFs will be obtained from the environment using the specified parameter. The first name will be used to create an NDF by propagation from INDF1, and an identifier for the new NDF will be returned. If more than one name was supplied for the parameter then the application may be invoked again (see LPG_AGAIN), in which case this routine will return an identifier for a new NDF with the next name in the group supplied on the first invocation.

If a modification element is included in the group expression supplied for the parameter on the first invocation of the application, the new NDF names are based on the names of the first group of existing data files (NDFs or catalogues) to be accessed by the application.

If an application attempts to get a new NDF by cancelling the parameter (PAR_CANCL), the name used to create the returned NDF is *NOT* the next one in the group, but is obtained by prompting the user for a single new NDF.

The monolith routine should arrange to invoke the application repeatedly until one or more of its NDF parameters have been exhausted (*i.e.* all its values used). See LPG_AGAIN.

Invocation:

```
CALL LPG_PROP( INDF1, CLIST, PARAM, INDF2, STATUS )
```

Arguments:

INDF1 = INTEGER (Given)
Identifier for an existing NDF (or NDF section) to act as a template.

CLIST = CHARACTER ∗ (∗) (Given)
A comma-separated list of the NDF components which are to be propagated to the new data structure. By default, the HISTORY, LABEL and TITLE components and all extensions are propagated. See the "Component Propagation" section in the documentation for routine NDF_PROP within SUN/33 for further details.

PARAM = CHARACTER ∗ (∗) (Given)
Name of the parameter for the new NDF.

INDF2 = INTEGER (Returned)
Identifier for the new NDF.

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

LPG_REPLA
Indicate if input NDFs can be replaced

Description:

Sets a flag indicating if LPG applications can use a single NDF as both input and output. If so, a temporary NDF is used to store the output. This NDF is then used to replace the existing input NDF once the application has completed. If REPLAC is `.FALSE.` (the default), an error is reported if an attempt is made to use a single NDF as both input and output.

Invocation:

```
CALL LPG_REPLA( REPLAC, STATUS )
```

Arguments:

REPLAC = LOGICAL (Given)

If `.TRUE.`, a single NDF can be used as both input and output from an application. If `.FALSE.`, an error will be reported if this is attempted.

STATUS = INTEGER (Given and Returned)

The global status.

LPG_START
Initialise the contents of the LPG common blocks

Description:

Initialises the global variables used by LPG. See LPG_AGAIN.

Invocation:

```
CALL LPG_START( VERBO, DELAYO, DISABO, STATUS )
```

Arguments:

VERBO = LOGICAL (Given)

If `.TRUE.` then the name of the data file being used for each parameter will be displayed on each invocation of the application at the point where the parameter is accessed. Parameters which are not multi-valued (*i.e.* that are associated with the same data file on all invocations) are not displayed. In addition, a blank line will be displayed on the screen between each invocation of the application. No text is displayed if VERB is `.FALSE.`.

DELAYO = REAL (Given)

Put a delay of DELAY seconds between invocations.

DISABO = LOGICAL (Given)

If `.TRUE.`, the looping facilities are disabled. LPG_AGAIN returns `.TRUE.` only on the first invocation, and LPG_ASSOC, LPG_CREAT, LPG_PROP, LPG_CREP, LPG_CATASSOC and LPG_CATCREAT make simple calls to the corresponding NDF or CAT routine.

STATUS = INTEGER (Given and Returned)

The global status.

LPG_CRPL1
Create a single new NDF placeholder using a specified parameter

Description:

This routine is equivalent to NDF_CREPL except that it allows the NDF to be specified using a GRP group expression (for instance, its name may be given within a text file, etc.). The first NDF in the group expression is returned. Any other names in the group expression are ignored. Any modification elements in the supplied group expression will be treated literally.

Invocation:

```
CALL LPG_CRPL1( PARAM, PLACE, NAME, STATUS )
```

Arguments:

PARAM = CHARACTER $*$ ($*$) (Given)
Name of the ADAM parameter.

PLACE = INTEGER (Returned)
NDF placeholder.

NAME = CHARACTER $*$ ($*$) (Returned)
The full file specification for the NDF.

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

LPG_STATE
Return the original PAR state of a parameter

Description:

On the first invocation of the application, this routine returns the current PAR state of specified parameter and stores it in common. On subsequent invocations, the stored state is returned rather than the current state.

Invocation:

```
CALL LPG_STATE( PARAM, STATE, STATUS )
```

Arguments:

PARAM = CHARACTER ∗ (∗) (Given)
The parameter name.

STATE = INTEGER (Returned)
The original PAR state of the parameter.

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

C Changes Introduced in LPG Version 3.0

- It has a separate identity. It was previously bundled into KAPLIBS.
- There is preliminary documentation.