Other Stuff for Overture
User Guide, Version 1.0

William D. Henshaw
Department of Mathematical Sciences,
Rensselaer Polytechnic Institute,
Troy, NY, USA, 12180.
June 6, 2021

Abstract: We describe miscellaneous Overture stuff:

Overture start and finish functions, Overture global variables

sPrintF,sScanF,... : miscellaneous utility routines.

getIndex : the `getIndex` utility routines for generating A++ Index's for operating on sub-domains of `MappedGrid`'s.

Twilight-zone functions : `OGFunction`, `OGPolyFunction`, `OGTrigFunction` and `OGPulseFunction`.

Database access functions : functions for reading various objects (such as a `CompositeGrid`) from a database file, including the function `getFromADataBase`.

display functions : formatted display functions for A++ arrays and for writing A++ arrays to files.

Integrate : This class provides methods for easily integrating grid functions on domains or boundaries of domains.

TridiagonalSystem solver : solve tridiagonal (or pentadiagonal) and block tridiagonal systems.

FortranIO : write unformatted Fortran files.

Reference Counting : A description of how reference counting works.

Contents

1 Overture start and finish functions, Overture global variables

1.1 Overture global variables.

The Overture class contains global variables that can be used as default arguments to functions. For example, `Overture::nullRealArray()` , can be used as a default argument for a `RealArray`. These instances of classes are accessed as function calls as opposed to building static global variables. Initially the later approach was taken but his caused difficulties since the loader would build the classes in some unknown order.

floatSerialArray & nullFloatArray():

doubleSerialArray & nullDoubleArray():

RealArray & nullRealArray():

intSerialArray & nullIntArray():

floatDistributedArray & nullFloatDistributedArray():

doubleDistributedArray & nullDoubleDistributedArray():

RealDistributedArray & nullRealDistributedArray():

IntegerDistributedArray & nullIntegerDistributedArray():

MappingParameters & nullMappingParameters():

floatMappedGridFunction & nullFloatMappedGridFunction():

doubleMappedGridFunction & nullDoubleMappedGridFunction():

realMappedGridFunction & nullRealMappedGridFunction():

intMappedGridFunction & nullIntMappedGridFunction():

floatGridCollectionFunction & nullFloatGridCollectionFunction():

realGridCollectionFunction & nullRealGridCollectionFunction():

doubleGridCollectionFunction & nullDoubleGridCollectionFunction():

intGridCollectionFunction & nullIntGridCollectionFunction():

BoundaryConditionParameters & defaultBoundaryConditionParameters():

GraphicsParameters & defaultGraphicsParameters():

1.2 start

int
start(int & argc, char **&argv)

Description: Overture initialization function. Call this routine before calling any Overture functions.

NOTES: In parallel the value of argc and the values in argv will be sent to all processors. Thus argc and argv will be changed on all processors except processor 0.

In parallel call ParallelUtility::broadCastArgsCleanup(argc,argv); to delete the argv arrays created by the parallel broad-cast

1.3 finish

int
finish()

Description: Overture cleanup function. Call this routine when you are done using Overture.

1.4 getAbortOption

AbortEnum
getAbortOption()

Description: Return the current abort option.

```
enum AbortEnum
{
  abortOnAbort,
  throwErrorOnAbort
};
```

1.5 setAbortOption

void
setAbortOption(AbortEnum action)

Description: Specify the action to take when the Overture::abort() function is called.

1.6 abort

void
abort()

Description: Abort the program or throw an error depending on the value of getAbortOption

1.7 abort

void
abort(const aString & message)

Description: Abort the program or throw an error depending on the value of getAbortOption

message (input) : print this message.

1.8 getGraphicsInterface

GenericGraphicsInterface*
getGraphicsInterface(const aString & windowTitle/*="Your Slogan Here"*/, const bool initialize/*=true*/,
int argc/* = 0*/, char *argv[] /*=NULL*/)

Description: Return a pointer to the one and only graphics interface. If the pointer is null, a new graphics interface will be built.

1.9 setGraphicsInterface

void
setGraphicsInterface(GenericGraphicsInterface *ps)

Description: Set the default graphics interface. This is normally called by the GenericGraphicsInterface constructor, there is no need for a typical user to call this function.

1.10 getMappingList

ListOfMappingRC*
getMappingList()

Description: Return a pointer to the default list of mappings. This pointer may be NULL.

1.11 setMappingList

void
setMappingList(ListOfMappingRC *list)

Description: Set the default list of mappings.

1.12 setDefaultGraphicsParameters

void
setDefaultGraphicsParameters(GraphicsParameters *gp =NULL)

Description: Set the default graphics parameters. By default reset the graphics parameters to the standard one.

1.13 openDebugFile

void
openDebugFile()

Description: Open the file Overture::debugFile for writing debugging info to. On a serial machine the file is named "overture.debug" On a parallel machine the file on processor 0 is named "overture.debug" while the file on processor X is named "overtureX.debug"

1.14 checkMemoryUsage

real
checkMemoryUsage(const aString & label, FILE *file =stdout)

Description: Check the current memory usage in Mega-bytes and print a message if the memory use has increased by 10 percent. You must first call turnOnMemoryChecking(true) for this function to be turned on.

label (input) : use this label on message

file (input): output messages to this file. If NULL, output no message.

Return value: current memory use (Mb).

1.15 getMaximumMemoryUsage

real
getMaximumMemoryUsage()

Description: Return the maximum memory use recorded by calls to checkMemoryUsage.

Return value: maximum memory use detected (Mb).

1.16 checkMemoryUsage

real
printMemoryUsage(const aString & label, FILE *file =stdout)

Description: Display the current memory usage.

label (input) : use this label on message

file (input): output messages to this file. If NULL, output no message.

1.17 incrementReferenceCountForPETSc

int
incrementReferenceCountForPETSc()

Description: Increment the reference count for objects that use PETSc.

1.18 decrementReferenceCountForPETSc

int
decrementReferenceCountForPETSc()

Description: Decrement the reference count for objects that use PETSc.

2 Miscellaneous Stuff

2.1 sPrintF

aString
sPrintF(const char *format, ...)

Description: Implementation of an "sprintf" like function that returns the formatted string NOT the number of chars assigned.

NOTE: this function assumes a maximum of 300 chars in the format string.

format (input) : use this printf style format.

argument '...' (input): variable length argument list.

Return value: formatted string

author: wdh

2.2 sScanF

int
sScanF(const aString & s, const char *format,
void *p0,
void *p1 =NULL,
void *p2 =NULL,
void *p3 =NULL,
void *p4 =NULL,
void *p5 =NULL,
void *p6 =NULL,
void *p7 =NULL,
void *p8 =NULL,
void *p9 =NULL,
void *p10 =NULL,
void *p11 =NULL,
void *p12 =NULL,
void *p13 =NULL,
void *p14 =NULL,
void *p15 =NULL,
void *p16 =NULL,
void *p17 =NULL,
void *p18 =NULL,
void *p19 =NULL,
void *p20 =NULL,
void *p21 =NULL,
void *p22 =NULL,
void *p23 =NULL,
void *p24 =NULL,
void *p25 =NULL,
void *p26 =NULL,
void *p27 =NULL,
void *p28 =NULL,
void *p29 =NULL
)

Description: A special version of sscanf that strips out any ',' characters and replaces them with ' ' and converts the format string with ftor so that %e and %f formats are converted properly for double precision.

p0,p1,... (input) : supply addresses of variables to save the results in.

2.3 sScanF

int
fScanF(FILE *file, const char *format,
void *p0,
void *p1 =NULL,
void *p2 =NULL,
void *p3 =NULL,
void *p4 =NULL,
void *p5 =NULL,
void *p6 =NULL,
void *p7 =NULL,
void *p8 =NULL,
void *p9 =NULL,
void *p10 =NULL,
void *p11 =NULL,
void *p12 =NULL,
void *p13 =NULL,
void *p14 =NULL,
void *p15 =NULL,
void *p16 =NULL,
void *p17 =NULL,
void *p18 =NULL,
void *p19 =NULL,
void *p20 =NULL,
void *p21 =NULL,
void *p22 =NULL,
void *p23 =NULL,
void *p24 =NULL,
void *p25 =NULL,
void *p26 =NULL,
void *p27 =NULL,
void *p28 =NULL,
void *p29 =NULL
)

Description: A special version of fscanf that strips out any ',' characters and replaces them with ' ' and converts the format string with ftor so that %e and %f formats are converted properly for double precision.

file (input) : scan this file.

format (input) : use this printf style format.

p0,p1,... (input) : supply addresses of variables to save the results in.

2.4 getLine

int
getLine(char s[], int lim)

Description: Read a line from standard input.

s (input) : char array in which to store the line

lim (input) : maximum number of chars that can be saved in s

2.5 getLine

int
getLine(aString &answer)

Description: Read a line from standard input.

s (input) : char array in which to store the line

lim (input) : maximum number of chars that can be saved in s

2.6 ftor

aString
ftor(const char *ss)

Description: "float to real aString conversion" This function is used to convert arguments to sscanf and fscanf so they work when OV_USE_DOUBLE is or is not defined. It will convert %e to %le and %f to %lf when OV_USE_DOUBLE is defined. Usually one should use sScanF and fScanF to have this done automatically so there is no need to call ftor directly.

ss (input) : convert this string.

author: wdh

2.7 printF

void
printF(const char *format, ...)

Description: Implementation of an "printf" like function that only prints on processor 0

s (input) : fill in this string.

format (input) : use this printf style format.

argument '...' (input): variable length argument list.

author: wdh

2.8 fPrintF

void
fPrintF(FILE *file, const char *format, ...)

Description: Implementation of an "fprintf" like function that only prints on processor 0

s (input) : fill in this string.

format (input) : use this printf style format.

argument '...' (input): variable length argument list.

author: wdh

3 Constructing A++ Index Objects for Grid Functions: getIndex, getBoundaryIndex, getGhostIndex

A number of functions are provided to construct A++ Index objects for grid functions. The `getIndex` function constructs Index objects for the interior points of a grid function. The function `getBoundaryIndex` constructs the Index objects corresponding to a given boundary face. The `getGhostIndex` function constructs Index objects for a ghost line on a given face of the grid.

Recall that a gridFunction can be cell-centred or vertex-centred in any of the coordinate directions. Typically grid functions are either all vertex-centred, all cell-centred or else face-centred (a face-centred grid function is vertex-centred along one axis and cell-centred along the others). By passing a grid function to `getIndex`, `getBoundaryIndex` or `getGhostIndex` one can be sure to get the Index objects corresponding to the "centred-ness" of the grid function. This is important because the "interior" points of a vertex-centred grid function are different from the interior points of a cell-centred grid function or the interior points of a face-centred grid function. For grid functions that are `faceCenteredAll` (i.e. they have components that are faceCentered along each axis) there is a `getIndex` function with an extra argument that species which face centering axis to use.

3.1 Index functions

Here are the getIndex functions (see also file `/users/henshaw/res/gf/OGgetIndex.h`). There are at least three flavours of each function. The first takes an `indexArray`, the second takes a grid function and component number and the third takes a grid function (in which case `component=0` is used to determine the Index's). The function `getIndex` has an additional 2 flavours.
A summary of the arguments is as follows:

- `indexArray` : an intArray with dimensions (0:1,0:2), such as `indexRange`, `gridIndexRange` or `dimension`, that defines a region on the grid.
- `extra` : Increase the size of the domain by this many lines.
- `side, axis` : these arguments define which face to use
- `floatMappedGridFunction` : Use the cell-centeredness properties of this grid function to determine the index range. Versions of the function also exist for `doubleMappedGridFunction` and `intMappedGridFunction`.
- `component` : base the Index objects on this component of the grid function (required since different components of a grid function may be centred in different ways). The functions where the `component` argument is missing use `component=0`. The exception to the above rule is when the grid function is `faceCenteredAll` in which case `component` =0,1 or 2 will indicate whether to return Index's for the faceCenteredAxis1, faceCenteredAxis2 or the faceCenteredAxis3 components.
- `ghostLine` : return Index objects for this ghost line. Choose `ghostline=1` for the first ghost line, `ghostline=2` for the second ghost line. (`ghostline=0` will give the boundary).

3.2 getIndex from a {float,double,int}MappedGridFunction

void
getIndex(const floatMappedGridFunction & u,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component. To get Index's for a `faceCenteredAll` grid function use the `getIndex` function described next.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getIndex(const floatMappedGridFunction & u,
int component,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.3 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const floatMappedGridFunction & u,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.4 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const intMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1,
int extra2,
int extra3
)

Description: return Index objects for a side of the region defined by indexArray

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function.

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ib1,Ib2,Ib3 (output): Index values for the given boundary of the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

3.5 getGhostIndex from a {float,double,int}MappedGridFunction

void
getGhostIndex(const floatMappedGridFunction & u,
int side,
int axis0,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getGhostIndex(const floatMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

extendedGridIndexRange

IntegerArray
extendedGridIndexRange(const MappedGrid & mg)

Description: Return the extendedGridIndexRange which is equal to mg.gridIndexRange except on interpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points). NOTE: Does not include ghost points on mixed physical/interpolation bounadries

Author: WDH

extendedGridRange

IntegerArray
extendedGridRange(const MappedGrid & mg)

Description: Return the extendedGridRange which is equal to mg.gridIndexRange except on interpolation boundaries AND mixedPhyscialInterpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points).

Author: WDH

3.6 getIndex from a {float,double,int}MappedGridFunction

void
getIndex(const floatMappedGridFunction & u,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component. To get Index's for a `faceCenteredAll` grid function use the `getIndex` function described next.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getIndex(const floatMappedGridFunction & u,
int component,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.7 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const floatMappedGridFunction & u,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.8 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const intMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1,
int extra2,
int extra3
)

Description: return Index objects for a side of the region defined by indexArray

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function.

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ib1,Ib2,Ib3 (output): Index values for the given boundary of the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

3.9 getGhostIndex from a {float,double,int}MappedGridFunction

void
getGhostIndex(const floatMappedGridFunction & u,
int side,
int axis0,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getGhostIndex(const floatMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

extendedGridIndexRange

IntegerArray
extendedGridIndexRange(const MappedGrid & mg)

Description: Return the extendedGridIndexRange which is equal to mg.gridIndexRange except on interpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points). NOTE: Does not include ghost points on mixed physical/interpolation bounadries

Author: WDH

extendedGridRange

IntegerArray
extendedGridRange(const MappedGrid & mg)

Description: Return the extendedGridRange which is equal to mg.gridIndexRange except on interpolation boundaries AND mixedPhyscialInterpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points).

Author: WDH

3.10 getIndex from a {float,double,int}MappedGridFunction

void
getIndex(const floatMappedGridFunction & u,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component. To get Index's for a `faceCenteredAll` grid function use the `getIndex` function described next.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getIndex(const floatMappedGridFunction & u,
int component,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.11 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const floatMappedGridFunction & u,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.12 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const intMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1,
int extra2,
int extra3
)

Description: return Index objects for a side of the region defined by indexArray

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function.

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ib1,Ib2,Ib3 (output): Index values for the given boundary of the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

3.13 getGhostIndex from a {float,double,int}MappedGridFunction

void
getGhostIndex(const floatMappedGridFunction & u,
int side,
int axis0,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getGhostIndex(const floatMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

extendedGridIndexRange

IntegerArray
extendedGridIndexRange(const MappedGrid & mg)

Description: Return the extendedGridIndexRange which is equal to mg.gridIndexRange except on interpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points). NOTE: Does not include ghost points on mixed physical/interpolation bounadries

Author: WDH

extendedGridRange

IntegerArray
extendedGridRange(const MappedGrid & mg)

Description: Return the extendedGridRange which is equal to mg.gridIndexRange except on interpolation boundaries AND mixedPhyscialInterpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points).

Author: WDH

3.14 getIndex from a {float,double,int}MappedGridFunction

void
getIndex(const floatMappedGridFunction & u,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component. To get Index's for a `faceCenteredAll` grid function use the `getIndex` function described next.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getIndex(const floatMappedGridFunction & u,
int component,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.15 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const floatMappedGridFunction & u,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.16 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const intMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1,
int extra2,
int extra3
)

Description: return Index objects for a side of the region defined by indexArray

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function.

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ib1,Ib2,Ib3 (output): Index values for the given boundary of the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

3.17 getGhostIndex from a {float,double,int}MappedGridFunction

void
getGhostIndex(const floatMappedGridFunction & u,
int side,
int axis0,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getGhostIndex(const floatMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

extendedGridIndexRange

IntegerArray
extendedGridIndexRange(const MappedGrid & mg)

Description: Return the extendedGridIndexRange which is equal to mg.gridIndexRange except on interpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points). NOTE: Does not include ghost points on mixed physical/interpolation bounadries

Author: WDH

extendedGridRange

IntegerArray
extendedGridRange(const MappedGrid & mg)

Description: Return the extendedGridRange which is equal to mg.gridIndexRange except on interpolation boundaries AND mixedPhyscialInterpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points).

Author: WDH

3.18 getIndex from a {float,double,int}MappedGridFunction

void
getIndex(const floatMappedGridFunction & u,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component. To get Index's for a `faceCenteredAll` grid function use the `getIndex` function described next.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getIndex(const floatMappedGridFunction & u,
int component,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.19 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const floatMappedGridFunction & u,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.20 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const intMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1,
int extra2,
int extra3
)

Description: return Index objects for a side of the region defined by indexArray

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function.

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ib1,Ib2,Ib3 (output): Index values for the given boundary of the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

3.21 getGhostIndex from a {float,double,int}MappedGridFunction

void
getGhostIndex(const floatMappedGridFunction & u,
int side,
int axis0,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getGhostIndex(const floatMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

extendedGridIndexRange

IntegerArray
extendedGridIndexRange(const MappedGrid & mg)

Description: Return the extendedGridIndexRange which is equal to mg.gridIndexRange except on interpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points). NOTE: Does not include ghost points on mixed physical/interpolation bounadries

Author: WDH

extendedGridRange

IntegerArray
extendedGridRange(const MappedGrid & mg)

Description: Return the extendedGridRange which is equal to mg.gridIndexRange except on interpolation boundaries AND mixedPhyscialInterpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points).

Author: WDH

3.22 getIndex from a {float,double,int}MappedGridFunction

void
getIndex(const floatMappedGridFunction & u,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component. To get Index's for a `faceCenteredAll` grid function use the `getIndex` function described next.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getIndex(const floatMappedGridFunction & u,
int component,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.23 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const floatMappedGridFunction & u,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.24 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const intMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1,
int extra2,
int extra3
)

Description: return Index objects for a side of the region defined by indexArray

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function.

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ib1,Ib2,Ib3 (output): Index values for the given boundary of the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

3.25 getGhostIndex from a {float,double,int}MappedGridFunction

void
getGhostIndex(const floatMappedGridFunction & u,
int side,
int axis0,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getGhostIndex(const floatMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

extendedGridIndexRange

IntegerArray
extendedGridIndexRange(const MappedGrid & mg)

Description: Return the extendedGridIndexRange which is equal to mg.gridIndexRange except on interpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points). NOTE: Does not include ghost points on mixed physical/interpolation bounadries

Author: WDH

extendedGridRange

IntegerArray
extendedGridRange(const MappedGrid & mg)

Description: Return the extendedGridRange which is equal to mg.gridIndexRange except on interpolation boundaries AND mixedPhyscialInterpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points).

Author: WDH

3.26 getIndex from a {float,double,int}MappedGridFunction

void
getIndex(const floatMappedGridFunction & u,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component. To get Index's for a `faceCenteredAll` grid function use the `getIndex` function described next.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getIndex(const floatMappedGridFunction & u,
int component,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.27 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const floatMappedGridFunction & u,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.28 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const intMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1,
int extra2,
int extra3
)

Description: return Index objects for a side of the region defined by indexArray

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function.

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ib1,Ib2,Ib3 (output): Index values for the given boundary of the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

3.29 getGhostIndex from a {float,double,int}MappedGridFunction

void
getGhostIndex(const floatMappedGridFunction & u,
int side,
int axis0,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getGhostIndex(const floatMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

extendedGridIndexRange

IntegerArray
extendedGridIndexRange(const MappedGrid & mg)

Description: Return the extendedGridIndexRange which is equal to mg.gridIndexRange except on interpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points). NOTE: Does not include ghost points on mixed physical/interpolation bounadries

Author: WDH

extendedGridRange

IntegerArray
extendedGridRange(const MappedGrid & mg)

Description: Return the extendedGridRange which is equal to mg.gridIndexRange except on interpolation boundaries AND mixedPhyscialInterpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points).

Author: WDH

3.30 getIndex from a {float,double,int}MappedGridFunction

void
getIndex(const floatMappedGridFunction & u,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component. To get Index's for a `faceCenteredAll` grid function use the `getIndex` function described next.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getIndex(const floatMappedGridFunction & u,
int component,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.31 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const floatMappedGridFunction & u,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.32 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const intMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1,
int extra2,
int extra3
)

Description: return Index objects for a side of the region defined by indexArray

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function.

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ib1,Ib2,Ib3 (output): Index values for the given boundary of the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

3.33 getGhostIndex from a {float,double,int}MappedGridFunction

void
getGhostIndex(const floatMappedGridFunction & u,
int side,
int axis0,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getGhostIndex(const floatMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

extendedGridIndexRange

IntegerArray
extendedGridIndexRange(const MappedGrid & mg)

Description: Return the extendedGridIndexRange which is equal to mg.gridIndexRange except on interpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points). NOTE: Does not include ghost points on mixed physical/interpolation bounadries

Author: WDH

extendedGridRange

IntegerArray
extendedGridRange(const MappedGrid & mg)

Description: Return the extendedGridRange which is equal to mg.gridIndexRange except on interpolation boundaries AND mixedPhyscialInterpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points).

Author: WDH

3.34 getIndex from a {float,double,int}MappedGridFunction

void
getIndex(const floatMappedGridFunction & u,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component. To get Index's for a `faceCenteredAll` grid function use the `getIndex` function described next.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getIndex(const floatMappedGridFunction & u,
int component,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.35 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const floatMappedGridFunction & u,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.36 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const intMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1,
int extra2,
int extra3
)

Description: return Index objects for a side of the region defined by indexArray

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function.

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ib1,Ib2,Ib3 (output): Index values for the given boundary of the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

3.37 getGhostIndex from a {float,double,int}MappedGridFunction

void
getGhostIndex(const floatMappedGridFunction & u,
int side,
int axis0,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getGhostIndex(const floatMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

extendedGridIndexRange

IntegerArray
extendedGridIndexRange(const MappedGrid & mg)

Description: Return the extendedGridIndexRange which is equal to mg.gridIndexRange except on interpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points). NOTE: Does not include ghost points on mixed physical/interpolation bounadries

Author: WDH

extendedGridRange

IntegerArray
extendedGridRange(const MappedGrid & mg)

Description: Return the extendedGridRange which is equal to mg.gridIndexRange except on interpolation boundaries AND mixedPhyscialInterpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points).

Author: WDH

3.38 getIndex from a {float,double,int}MappedGridFunction

void
getIndex(const floatMappedGridFunction & u,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component. To get Index's for a `faceCenteredAll` grid function use the `getIndex` function described next.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getIndex(const floatMappedGridFunction & u,
int component,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.39 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const floatMappedGridFunction & u,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.40 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const intMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1,
int extra2,
int extra3
)

Description: return Index objects for a side of the region defined by indexArray

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function.

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ib1,Ib2,Ib3 (output): Index values for the given boundary of the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

3.41 getGhostIndex from a {float,double,int}MappedGridFunction

void
getGhostIndex(const floatMappedGridFunction & u,
int side,
int axis0,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getGhostIndex(const floatMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

extendedGridIndexRange

IntegerArray
extendedGridIndexRange(const MappedGrid & mg)

Description: Return the extendedGridIndexRange which is equal to mg.gridIndexRange except on interpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points). NOTE: Does not include ghost points on mixed physical/interpolation bounadries

Author: WDH

extendedGridRange

IntegerArray
extendedGridRange(const MappedGrid & mg)

Description: Return the extendedGridRange which is equal to mg.gridIndexRange except on interpolation boundaries AND mixedPhyscialInterpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points).

Author: WDH

3.42 getIndex from a {float,double,int}MappedGridFunction

void
getIndex(const floatMappedGridFunction & u,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component. To get Index's for a `faceCenteredAll` grid function use the `getIndex` function described next.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getIndex(const floatMappedGridFunction & u,
int component,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.43 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const floatMappedGridFunction & u,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.44 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const intMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1,
int extra2,
int extra3
)

Description: return Index objects for a side of the region defined by indexArray

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function.

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ib1,Ib2,Ib3 (output): Index values for the given boundary of the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

3.45 getGhostIndex from a {float,double,int}MappedGridFunction

void
getGhostIndex(const floatMappedGridFunction & u,
int side,
int axis0,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getGhostIndex(const floatMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

extendedGridIndexRange

IntegerArray
extendedGridIndexRange(const MappedGrid & mg)

Description: Return the extendedGridIndexRange which is equal to mg.gridIndexRange except on interpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points). NOTE: Does not include ghost points on mixed physical/interpolation bounadries

Author: WDH

extendedGridRange

IntegerArray
extendedGridRange(const MappedGrid & mg)

Description: Return the extendedGridRange which is equal to mg.gridIndexRange except on interpolation boundaries AND mixedPhyscialInterpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points).

Author: WDH

3.46 getIndex from a {float,double,int}MappedGridFunction

void
getIndex(const floatMappedGridFunction & u,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component. To get Index's for a `faceCenteredAll` grid function use the `getIndex` function described next.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getIndex(const floatMappedGridFunction & u,
int component,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.47 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const floatMappedGridFunction & u,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.48 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const intMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1,
int extra2,
int extra3
)

Description: return Index objects for a side of the region defined by indexArray

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function.

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ib1,Ib2,Ib3 (output): Index values for the given boundary of the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

3.49 getGhostIndex from a {float,double,int}MappedGridFunction

void
getGhostIndex(const floatMappedGridFunction & u,
int side,
int axis0,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getGhostIndex(const floatMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

extendedGridIndexRange

IntegerArray
extendedGridIndexRange(const MappedGrid & mg)

Description: Return the extendedGridIndexRange which is equal to mg.gridIndexRange except on interpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points). NOTE: Does not include ghost points on mixed physical/interpolation bounadries

Author: WDH

extendedGridRange

IntegerArray
extendedGridRange(const MappedGrid & mg)

Description: Return the extendedGridRange which is equal to mg.gridIndexRange except on interpolation boundaries AND mixedPhyscialInterpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points).

Author: WDH

3.50 getIndex from a {float,double,int}MappedGridFunction

void
getIndex(const floatMappedGridFunction & u,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component. To get Index's for a `faceCenteredAll` grid function use the `getIndex` function described next.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getIndex(const floatMappedGridFunction & u,
int component,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.51 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const floatMappedGridFunction & u,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.52 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const intMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1,
int extra2,
int extra3
)

Description: return Index objects for a side of the region defined by indexArray

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function.

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ib1,Ib2,Ib3 (output): Index values for the given boundary of the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

3.53 getGhostIndex from a {float,double,int}MappedGridFunction

void
getGhostIndex(const floatMappedGridFunction & u,
int side,
int axis0,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getGhostIndex(const floatMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

extendedGridIndexRange

IntegerArray
extendedGridIndexRange(const MappedGrid & mg)

Description: Return the extendedGridIndexRange which is equal to mg.gridIndexRange except on interpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points). NOTE: Does not include ghost points on mixed physical/interpolation bounadries

Author: WDH

extendedGridRange

IntegerArray
extendedGridRange(const MappedGrid & mg)

Description: Return the extendedGridRange which is equal to mg.gridIndexRange except on interpolation boundaries AND mixedPhyscialInterpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points).

Author: WDH

3.54 getIndex from a {float,double,int}MappedGridFunction

void
getIndex(const floatMappedGridFunction & u,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component. To get Index's for a `faceCenteredAll` grid function use the `getIndex` function described next.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getIndex(const floatMappedGridFunction & u,
int component,
Index & I1,
Index & I2,
Index & I3,
int extra1,
int extra2,
int extra3)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.55 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const floatMappedGridFunction & u,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description:

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

I1,I2,I3 (output): Index values for the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

3.56 getBoundaryIndex from a {float,double,int}MappedGridFunction

void
getBoundaryIndex(const intMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ib1,
Index & Ib2,
Index & Ib3,
int extra1,
int extra2,
int extra3
)

Description: return Index objects for a side of the region defined by indexArray

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function.

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ib1,Ib2,Ib3 (output): Index values for the given boundary of the region

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

3.57 getGhostIndex from a {float,double,int}MappedGridFunction

void
getGhostIndex(const floatMappedGridFunction & u,
int side,
int axis0,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function. Use the "first" component of the grid function, that is use the base value for each component.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

void
getGhostIndex(const floatMappedGridFunction & u,
int component,
int side,
int axis,
Index & Ig1,
Index & Ig2,
Index & Ig3,
int ghostLine =1,
int extra1 =0,
int extra2 =OGgetIndexDefaultValue,
int extra3 =OGgetIndexDefaultValue)

Description: Get Index's corresponding to a given ghost-line of region defined by a grid function.

u (input): Base the Index's on the `indexRange` and cell-centredness associated with this grid function

component (input): use this component of the grid function, UNLESS the grid function is `faceCenteredAll` in which case component =0,1 or 2 will indicate whether to return Index's for the `faceCenteredAxis1`, `faceCenteredAxis2` or the `faceCenteredAxis3` components.

side,axis (input): defines which side=0,1 and axis=0,1,2

Ig1,Ig2,Ig3 (output): Index values for the given ghostline on the given side

ghostline (input): get Index's for this ghost line, can be positive, negative or zero. A value of zero would give the boundary, a value of 1 would give the first line outside and a value of -1 would give the first line inside.

extra1,extra2,extra3 (input): increase region by this many lines, by default extra1=0, while extra2 and extra3 default to extra1 (so that if you only set extra1=1 then by default extra2=extra3=1)

Author: WDH

extendedGridIndexRange

IntegerArray
extendedGridIndexRange(const MappedGrid & mg)

Description: Return the extendedGridIndexRange which is equal to mg.gridIndexRange except on interpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points). NOTE: Does not include ghost points on mixed physical/interpolation bounadries

Author: WDH

extendedGridRange

IntegerArray
extendedGridRange(const MappedGrid & mg)

Description: Return the extendedGridRange which is equal to mg.gridIndexRange except on interpolation boundaries AND mixedPhyscialInterpolation boundaries where it is equal to mg.extendedIndexRange (i.e. it includes the ghost points).

Author: WDH

4 OGFunction: A class for defining Twilight-zone flows

The class `OGFunction` and derived classes such as `OGTrigFunction`, `OGPolyfunction` and `OGPulseFunction` can be used to define "exact solutions" for PDE solvers in the manner that has come to be known as "twilight-zone flow" (Brown, 1985).

Basically this class defines an analytic function (such as a polynomial) and provides functions for evaluating this function and derivatives in convenient ways. Note that as of version 14 it is possible to evaluate any partial derivative using the `gd` (general derivative) member function.

It is often difficult to come up with exact solutions to test a PDE code. To over come this difficulty one can add forcing functions to the PDE and boundary conditions that will make any given function an exact solution. For example given the time dependent PDE for vector function **u**

$$\mathbf{u}_t + a\mathbf{u}_x + b\mathbf{u}_y = \mathbf{f}(x, y, t)$$

with boundary conditions

$$B(\mathbf{u}) = \mathbf{g}(x, y, t)$$

one can make any given function $\mathbf{U}(x, y, t)$ an exact solution by defining

$$\mathbf{f}(x, y, t) = \mathbf{U}_t + a\mathbf{U}_x + b\mathbf{U}_y$$

and

$$\mathbf{g}(x, y, t) = B(\mathbf{U}(x, y, t))$$

The class `OGPolyFunction` can be used to define the function $\mathbf{U}(x, y, z, t)$ which is defined to be a polynomial in x, y, $[z]$ and t. The class `OGTrigfunction` can be used to define a function $\mathbf{U}(x, y, z, t)$ which is a trigonometric function such as $\cos(2\pi x)\cos(2\pi y)\cos(2\pi t)$.

The base class `OGFunction` does not define any functions and thus you should never construct one of these. You might, however, keep a pointer to the base class. This pointer can point to an object of a derived class.

4.1 Evaluate the function or a derivative at a point

```
real
operator()(const real x,
           const real y,
           const real z,
           const int n =0,
           const real t =0.)

real
operator()(const real x, const real y, const real z, const int n)

real
operator()(const real x, const real y, const real z)

real
x(const real x,
  const real y,
  const real z,
  const int n =0,
  const real t =0.)
```

Description: Evaluate the function or a derivative of the function at a point The function name **x** can be replaced by any one of **t**, **x**, **y**, **z**, **xx**, **yy**, **zz**, **xy**, **xz**, **yz**, **xxx** or **xxxx**.

x,y,z (input) : coordinates

n (input) : component number (starting from 0)

t (input) : time

real
gd(const int & ntd,
const int & nxd,
const int & nyd,
const int & nzd,
const real x0,
const real y0,
const real z0,
const int n =0,
const real t0 =0.)

Description: Evaluate a general derivative. The arguments are the same as in the corresponding `x` function except that the first 4 arguments specify the derivative to compute.

ntd,nxd,nyd,nzd (input): Specify the derivative to compute by indicating the order of each partial derivative.

ntd : number of time derivatives (order of the time derivative).

nxd : number of x derivatives (order of the x derivative).

nyd : number of y derivatives (order of the y derivative).

nzd : number of z derivatives (order of the z derivative).

4.2 Evaluate the function or a derivative on a MappedGrid

RealDistributedArray
operator()(const MappedGrid & c, const Index & I1,
const Index & I2,const Index & I3, const Index & N)

RealDistributedArray
operator()(const MappedGrid & c,
const Index & I1,
const Index & I2,
const Index & I3,
const Index & N,
const real t =0.,
const GridFunctionParameters::GridFunctionType & centering
=defaultCentering)

RealDistributedArray
x(const MappedGrid & c,
const Index & I1,
const Index & I2,const
Index & I3,
const Index & N,
const real t,
const GridFunctionParameters::GridFunctionType & centering
= defaultCentering)

Description: Evaluate the function or a derivative of the function at points on a MappedGrid The function name **x** can be replaced by any one of **t**, **x**, **y**, **z**, **xx**, **yy**, **zz**, **xy**, **xz**, **yz**, **laplacian**, **xxx** or **xxxx**.

I1,I2,I3 (input) : Ranges that indicate points to use, for example by default the points

$$\texttt{c.center}()(\texttt{I1},\texttt{I2},\texttt{I3},\texttt{0}:\texttt{numberOfDimensions}-\texttt{1})$$

are used.

N (input) : component indicies to assign

t (input) : time

centering (input): This enum is found in `GridFunctionParameters`. It indicates the positions of the coordinates, one of

defaultCentering use the `c.center()` array (vertices for a vertex centered grid and cell centers for a cell-centered grid).

vertexCentered grid vertices, `c.vertex()`.

cellCentered `c.center()` for a cell-centered grid or else `c.corner()` for a vertex centered grid (the cell centers).

faceCenteredAxis1 use the center of the cell face in the axis1 direction (defined by averaging the `c.vertex()` `values` for the y,z coordinates).

faceCenteredAxis2 use the center of the cell face in the axis2 direction (defined by averaging the `c.vertex()` `values` for the x,z coordinates).

faceCenteredAxis3 use the center of the cell face in the axis3 direction (defined by averaging the `c.vertex()` `values` for the x,y coordinates).

RealDistributedArray
gd(const int & ntd,
const int & nxd,
const int & nyd,
const int & nzd,
const MappedGrid & c,
const Index & I1,
const Index & I2,
const Index & I3,
const Index & N,
const real t0 =0.,
const GridFunctionParameters::GridFunctionType & centering = defaultCentering)

Description: Evaluate a general derivative. The arguments are the same as in the corresponding `x` function except that the first 4 arguments specify the derivative to compute.

ntd,nxd,nyd,nzd (input): Specify the derivative to compute by indicating the order of each partial derivative.

ntd : number of time derivatives (order of the time derivative).

nxd : number of x derivatives (order of the x derivative).

nyd : number of y derivatives (order of the y derivative).

nzd : number of z derivatives (order of the z derivative).

4.3 Evaluate the function or a derivative on a CompositeGrid

realCompositeGridFunction
operator()(CompositeGrid & cg,
const Index & N,
const real t,
const GridFunctionParameters::
GridFunctionType & centering = defaultCentering)

realCompositeGridFunction
operator()(CompositeGrid & cg, const Index & N)

realCompositeGridFunction
operator()(CompositeGrid & cg)

realCompositeGridFunction
x(CompositeGrid & cg,
const Index & N,
const real t,
const GridFunctionParameters::GridFunctionType & centering
= defaultCentering)

Description: Evaluate the function or a derivative of the function at points on a CompositeGrid The function name **x** can be replaced by any one of **t**, **x**, **y**, **z**, **xx**, **yy**, **zz**, **xy**, **xz**, **yz**, **laplacian**, **xxx** or **xxxx**.

cg (input) : use this grid. By default the points

$$\texttt{c.center}()(\texttt{i1},\texttt{I2},\texttt{I3},\texttt{0}:\texttt{numberOfDimensions}-\texttt{1})$$

are used.

N (input) : evaluate these components

t (input) : time

centering (input): This enum is found in `GridFunctionParameters`. It indicates the positions of the coordinates, one of

defaultCentering use the `c.center()` array (vertices for a vertex centered grid and cell centers for a cell-centered grid).

vertexCentered grid vertices, `c.vertex()`.

cellCentered `c.center()` for a cell-centered grid or else `c.corner()` for a vertex centered grid (the cell centers).

faceCenteredAxis1 use the center of the cell face in the axis1 direction (defined by averaging the `c.vertex()` `values` for the y,z coordinates).

faceCenteredAxis2 use the center of the cell face in the axis1 direction (defined by averaging the `c.vertex()` `values` for the x,z coordinates).

faceCenteredAxis3 use the center of the cell face in the axis1 direction (defined by averaging the `c.vertex()` `values` for the x,y coordinates).

4.4 OGPolyFunction

This class is derived from the `OGFunction` class and defines a function that is a polynomial in space times a polynomial in time of the form

$$U(x,y,z,n,t) = (\sum_{i,j,k} c(i,j,k,n)x^i y^j z^k) \sum_m a(m,n)t^m$$

Each component is a polynomial in space and time. The coefficient matrices, $c(i,j,k,n)$ and $a(m,n)$ can be specified or else default values can be used.

4.4.1 Constructor

OGPolyFunction(const int & degreeOfSpacePolynomial =2,
const int & numberOfDimensions0 =3,
const int & numberOfComponents0 =10,
const int & degreeOfTimePolynomial =1)

Description: Create a polynomial with the given degree, number Of Space Dimensions and for a maximum number of components. The polynomial created is of the form

$$U = (1 + x + x^2 + ... + x^n)(1 + t + ... + t^m) \quad \text{in 1D}$$
$$U = (1 + x + x^2 + ... + x^n + y + y^2 + ... + y^n)(1 + t + ... + t^m) \quad \text{in 2D}$$
$$U = (1 + x + x^2 + ... + x^n + y + y^2 + ... + y^n + z + z^2 + ... + z^n)(1 + t + ... + t^m) \quad \text{in 3D}$$

degreeOfSpacePolynomial (input): degree of the polynomial in x,y,z (n in the above formula).

numberOfDimensions (input): number of space dimensions, 1,2, or 3.

numberOfComponents0 (input): maximum number of components required.

degreeOfTimePolynomial (input): degree of the polynomial in t (m in the above formula).

Notes: Only polynomials with degreeOfSpacePolynomial < 5 and degreeOfTimePolynomial < 5 are supported

Author: WDH

4.4.2 setCoefficients

void
setCoefficients(const RealArray & c, const RealArray & a0)

Description: Use this member function to set the coefficient matrices c and a of a *general* polynomial up to order 6.

$$U(x,y,z,n,t) = (\sum_{i,j,k} c(i,j,k,n)x^i y^j z^k) \sum_m a(m,n)t^m$$

Note that the values of `numberOfDimensions` and `numberOfComponents` given in the call to the constructor help to determine the polynomial created here.

c (input) : array of dimension $c(0:nx, 0:nx, 0:nx, 0:$ `numberOfComponents` $-1)$ that gives the coefficients of the spatial polynomial of degree nx (`numberOfComponents` is the value given in call to the constructor). Some values in c may be ignored depending on the value for `numberOfDimensions`.

a (input) : array of dimension $a(0:nt, 0:$ `numberOfComponents` $-1)$ that gives the coefficients of the time polynomial (`numberOfComponents` is the value given in call to the constructor).

Author: WDH

4.5 OGTrigFunction

4.5.1 Constructors

OGTrigFunction(const real & fx_ =1.,
const real & fy_ =1.,
const real & fz_ =0.,
const real & ft_ =0.,
const int & maximumNumberOfComponents =10)

Description: This class is derived from the `OGFunction` class and defines a function and defines a function that is a trigonometric polynomial:

$$u_n(x,y,z,t) = a(n)\cos(f_x(n)\pi(x-g_x(n)))\cos(f_y(n)\pi(y-g_y(n)))\cos(f_z(n)\pi(z-g_z(n)))\cos(f_t(n)\pi(t-g_t(n)))+c(n)$$

where $a(n)$, $f_x(n)$, $f_y(n)$ etc. can be given values for each component n.

fx_, fy_, fz_, ft_ (input): give frequencies (constant for all components).

maximumNumberOfComponents (input): maximum number of components.

Notes: By default $a(n) = 1$ and $g_x(n) = g_y(n) = g_z(n) = g_t(n) = 0$.

Author: WDH

OGTrigFunction(const RealArray & fx_,
const RealArray & fy_,
const RealArray & fz_,
const RealArray & ft_)

Description: Use this constructor to supply different frequencies for different components.

fx_, fy_, fz_, ft_ (input): give frequencies for different components. The dimension of fx_ will determine the maximumNumberOfComponents.

4.5.2 setAmplitudes

int
setAmplitudes(const RealArray & a_)

Description: Use this function to supply different amplitudes for different components.

a_ (input): give amplitudes for different components. The dimension of a_ should be equal to the maximumNumberOfComponents as determined by the call to the constructor.

4.5.3 setConstants

int
setConstants(const RealArray & c_)

Description: Use this function to supply different constants for different components.

c_ (input): give constants for different components. The dimension of c_ should be equal to the maximumNumberOfComponents as determined by the call to the constructor.

4.5.4 setFrequencies

int
setFrequencies(const RealArray & fx_,
const RealArray & fy_,
const RealArray & fz_,
const RealArray & ft_)

Description: Use this function to supply different frequencies for different components.

fx_, fy_, fz_, ft_ (input): give frequencies for different components. The dimension of fx_ will determine the maximumNumberOfComponents.

4.5.5 setShifts

int
setShifts(const RealArray & gx_,
const RealArray & gy_,
const RealArray & gz_,
const RealArray & gt_)

Description: Use this function to supply different shifts for different components.

gx_, gy_, gz_, gt_ (input): give shifts for different components. The dimensions of gx_, gy_,... should be equal to the maximumNumberOfComponents as determined by the call to the constructor.

4.6 OGPulseFunction

This class defines a *pulse* like function that can be useful for testing adaptive mesh refinement codes.

The pulse is defined as a generalized Gaussian,

$$u(\mathbf{x},t) = a_0 \exp(-a_1|\mathbf{x}-\mathbf{b}(t)|^{2p})$$
$$\mathbf{b}(t) = \mathbf{c}_0 + \mathbf{v}t$$

where $p > \frac{1}{2}$.

The derivatives of u are determined as follows. Letting

$$r = (x-b_0)^2 + (y-b_1)^2 + (z-b_2)^2$$
$$f = r^p$$
$$u = a_0 \exp(-a_1 f)$$

then

$$u_x = a_0 \exp(-a_1 f)\,[-a_1 f_x]$$
$$u_{xx} = a_0 \exp(-a_1 f)\,\left[(a_1 f_x)^2 - a_1 f_{xx}\right]$$
$$u_{xy} = a_0 \exp(-a_1 f)\,[a_1 f_x a_1 f_y - a_1 f_{xy}]$$
$$u_{xxxx} = a_0 \exp(-a_1 f)\,\left[-(a_1 f_x)^3 + 3 f_x f_{xx} - f_{xxx}\right]$$
$$u_{xxxx} = a_0 \exp(-a_1 f)\,\left[(a_1 f_x)^4 + 3 f_{xx}^2 - 6 f_x^2 f_{xx} + 4 f_x f_{xxx} - f_{xxxx}\right]$$

where

$$f_x = 2pr^{p-1}(x-b_0)$$
$$f_{xx} = 2pr^{p-1}(2(p-1)\frac{(x-b_0)^2}{r} + 1)$$
$$f_{xy} = 4p(p-1)r^{p-2}((x-b_0)(y-b_1))$$
$$f_{xxx} = 4p(p-1)r^{p-2}(x-b_0)\left[(p-2)\frac{(x-b_0)^2}{r} + 3\right]$$
$$f_{xxxx} = 4p(p-1)r^{p-2}\left[4(p-2)\frac{(x-b_0)^2}{r}((p-3)\frac{(x-b_0)^2}{r} + 3) + 3\right]$$

4.6.1 Constructor

OGPulseFunction(int numberOfDimensions_ = 2,
int numberOfComponents_ =1,
real a0_ =1.,
real a1_ =5.,
real c0_ =0.,
real c1_ =0.,
real c2_ =0.,
real v0_ =1.,
real v1_ =1.,
real v2_ =1.,
real p_ =1.)

Description: Define a pulse.

$$U = a_0 \exp(-a_1|\mathbf{x}-\mathbf{b}(t)|^{2p})$$
$$\mathbf{b}(t) = \mathbf{c}_0 + \mathbf{v}t$$

numberOfDimensions_ (input): number of space dimensions, 1,2, or 3.

numberOfComponents_ (input): maximum number of components required.

a0_,a1_,...p_ (input): pulse parameters

4.6.2 setRadius

int
setRadius(real radius)

Description: Set the approximte radius of the pulse. This will set the parameter a_1 according to the formula radius $= 1/\sqrt{a_1}$.

radius (input): approximate radius.

Author: WDH

4.6.3 setRadius

int
setShape(real p_)

Description: Set the *shape* parameter p. $p = 1$ gives a Gaussian pulse, choosing a larger value of p will cause the pulse to flatten on the top and approach a top-hat function as p tends to infinity.

p_ (input): shape parameter, $p > \frac{1}{2}$.

Author: WDH

4.6.4 setCentre

int
setCentre(real c0_ =0.,
real c1_ =0.,
real c2_ =0.)

Description: Set the pulse centre.

c0_,c1_,c2_ (input): centre.

Author: WDH

4.6.5 setVelocity

int
setVelocity(real v0_ =1.,
real v1_ =1.,
real v2_ =1.)

Description: Set the pulse velocity.

v0_,v1_,v2_ (input): velocity.

Author: WDH

4.6.6 setCoefficients

void
setParameters(int numberOfDimensions_ = 2,
int numberOfComponents_ =1,
real a0_ =1.,
real a1_ =5.,
real c0_ =0.,
real c1_ =0.,
real c2_ =0.,
real v0_ =1.,
real v1_ =1.,
real v2_ =1.,
real p_ =1.)

Description: Use this member function to set parameters.

Define a pulse.

$$U = a_0 \exp(-a_1|\mathbf{x} - \mathbf{c}(t)|^p) \qquad p > \frac{1}{2}$$
$$\mathbf{c}(t) = \mathbf{c}_0 + \mathbf{v}t$$

numberOfDimensions_ (input): number of space dimensions, 1,2, or 3.

numberOfComponents_ (input): maximum number of components required.

a0_,a1_,...p_ (input): pulse parameters.

Author: WDH

4.7 Examples

The file `Overture/tests/tz.C` is an example code that uses the `OGPolyFunction`, `OGTrigFunction` and `OGPulseFunction` classes. See also the exmaples in the primer directory.

5 Data-base Access Functions

Here are the functions that can be used to read in various objects from a data base.

5.1 getFromADataBase(CompositeGrid & cg,...,loadBalance,...)

int
getFromADataBase(CompositeGrid & cg,
aString & fileName,
bool loadBalance,
const aString & gridName =nullString,
const bool & checkTheGrid =FALSE,
int printInfo =1)

Description: Read in a CompositeGrid from a data-base file and load-balance with the default LoadBalancer

6 Display functions for arrays (writing arrays to files)

6.1 display: display an A++ array

int
display(const floatArray & x, const char *label, const char *format_, const Index *Iv /* =NULL*/)

Description: Another version of display – pass a format but no FILE

x (input) : array to display. There are also versions of this routine for int and double arrays.

label (input): optional header label

format (input) : an optional format such as "%6.1e " or "%11.4e " (note blank at end) that will be used to display each element in the array. The default is "%11.4e "

Iv[d] : If Iv is not NULL then print the values x(Iv[0],Iv[1],Iv[2],...) You must supply at least nd entries in the array Iv[d] where nd=x.numberOfDimensions();

6.2 display: save an A++ array in a file

int
display(const floatArray & x,
const char *label = NULL,
FILE *file = NULL,
const char *format_ = NULL,
const Index *Iv /* =NULL*/)

Description: Display an A++ array

x (input) : array to display. There are also versions of this routine for int and double arrays.

label (input): optional header label

file (input) : optionally supply a file to print to.

format (input) : an optional format such as "%6.1e " or "%11.4e " (note blank at end) that will be used to display each element in the array. The default is "%11.4e "

Iv[d] : If Iv is not NULL then print the values x(Iv[0],Iv[1],Iv[2],...)

6.3 display an A++ array with DisplayParameters

int
display(const floatArray & x, const char *label, const DisplayParameters & displayParameters,
const Index *Iv /* =NULL*/)

Description: Another version of display – pass a format but no FILE

x (input) : array to display. There are also versions of this routine for int and double arrays.

label (input): optional header label

format (input) : an optional format such as "%6.1e " or "%11.4e " (note blank at end) that will be used to display each element in the array. The default is "%11.4e "

displayParameters (input) : provide parameters for display.

Iv[d] : If Iv is not NULL then print the values x(Iv[0],Iv[1],Iv[2],...)

6.4 displayMask

int
displayMask(const intArray & mask,
const aString & label =nullString,
FILE *file = NULL,
const Index *Iv /* =NULL*/)

Description: Display the mask array in a MappedGrid in a reasonable way The mask array in a MappedGrid is a bit-mapping that is difficult to look at if displayed in the formal way. This routine will display the mask in a more compact form (although some information is not printed) where each entry printed will mean:

1 : ISdiscretizationPoint

2 : ISghostPoint

-1 : ISinterpolationPoint

6.5 displayMask

int
displayMask(const intSerialArray & mask,
const aString & label =nullString,
FILE *file = NULL,
const Index *Iv /* =NULL*/)

Description: Display the mask array in a MappedGrid in a reasonable way The mask array in a MappedGrid is a bit-mapping that is difficult to look at if displayed in the formal way. This routine will display the mask in a more compact form (although some information is not printed) where each entry printed will mean:

1 : ISdiscretizationPoint

2 : ISghostPoint

-1 : ISinterpolationPoint

7 Integrate: integrate grid functions on overlapping grids

The `Integrate` class has functions that can be used to integrate a grid function over a domain or over the boundary (or a subset of the boundary). For example, one may want to compute the total mass found in a domain or compute the force on a body.

Integrating a function on an overlapping grid is non-trivial since care must be taken in the region where grids overlap.

The most important member functions are

volumeIntegral(u) : compute the volume integral of a RealCompositeGridFunction u.

surfaceIntegral(u) : compute the surface integral of a RealCompositeGridFunction u.

defineSurface(s,...) : define a sub-surface 's' as a collection of sides of grids. For example, for the 'sphere in a box' grid you could define a sub-surface that represents the two surface grids on the sphere.

surfaceIntegral(u,s) : compute the surface integral on the surface 's'.

The integral of a function f over a domain Ω in $\mathbb{R}^d$ is

$$\mathcal{I} = \int_\Omega f(\mathbf{x})\ d\mathbf{x}$$

If the domain is parameterized by $\mathbf{x} = \mathbf{x}(\mathbf{r})$ for $\mathbf{r} \in [0,1]^d$ then

$$\begin{aligned}\mathcal{I} &= \int_{[0,1]^d} f(\mathbf{x}(\mathbf{r})) \left|\frac{\partial \mathbf{x}}{\partial \mathbf{r}}\right|\ d\mathbf{r} \\ &= \int_{[0,1]^d} f(\mathbf{x}(\mathbf{r}))J(\mathbf{r})\ d\mathbf{r} \\ &= \int_{[0,1]^d} F(\mathbf{r})\ d\mathbf{r}\end{aligned}$$

Thus the integral on a curvilinear grid can be converted to an integral on a uniform grid by integrating the function $F(\mathbf{r}) = f(\mathbf{x}(\mathbf{r}))J(\mathbf{r})$.

Consider the one-dimensional case

$$\mathcal{I} = \int_0^1 F(r)dr$$

This integral can, for example, be approximated to second-order accuracy by the trapezoidal rule

$$\begin{aligned}\int_0^1 F(r)dr &= \sum_{i=0}^{N-1} \frac{1}{2}(F_{i+1} + F_i)\Delta r + O(\Delta r^2) \\ &= \frac{1}{2}F_0\Delta r + \sum_{i=1}^{N-1} F_i\Delta r + \frac{1}{2}F_N\Delta r + O(\Delta r^2)\end{aligned}$$

where $r_i = i\Delta r$, $i = 0, 1, 2, \ldots, N$ and $\Delta r = 1/N$ and $F_i = F(r_i)$.

If the region Ω is covered by an overlapping grid, the situation is more complicated. One cannot simply add contributions from each component grid independently since where the grids overlap some contributions will be added multiple times.

The Integrate class provides two approaches for integration in the case of an overlapping grid

Laplace approximation : the integral is approximated using the left null vector of a discretization to the Laplace equation with Neumann boundary conditions. The left null vector, appropriately scaled, provides integration weights for each grid point. In this case there is no need to eliminate the region of overlap.

Hybrid approximation : in this method the overlap is removed, leaving a gap, and an unstructured grid is generated to fill the gap. An integration formula is defined on the hybrid grid. This formula is transformed into weights applied to the points on the original overlapping grids. NOTE: Currently this option is only available for surface grids in 3D.

7.1 Surface Integrals

Assume that we are given a mapping for a volume, $\mathbf{x} = \mathbf{G}(\mathbf{r})$, with $\mathbf{r} = (r, s, t)$ being the unit cube coordinates.

We want to integrate a function $f(\mathbf{x})$ on a surface, $\mathcal{S}$

$$\int_{\mathcal{S}} f(\mathbf{x}) d\mathbf{x} = \int_{\mathcal{S}} f(\mathbf{x})(\mathbf{x}_r \times \mathbf{x}_s) \cdot \mathbf{n} \; d\mathbf{r} \equiv \int_{\mathcal{S}} f(\mathbf{x}(\mathbf{r}))\gamma(\mathbf{r}) \; d\mathbf{r}$$

where $\gamma \equiv (\mathbf{x}_r \times \mathbf{x}_s) \cdot \mathbf{n}$. Assume we a given a tesselation for $\mathcal{S} = \bigcup_{k=1}^{N} \sigma_k$ for the surface consisting of triangles and quadrilaterals (σ_k represents either a triangle or quadrilateral on the surface). Then

$$\int_{\mathcal{S}} f(\mathbf{x}) d\mathbf{x} = \sum_k \int_{\sigma_k} f(\mathbf{x})\gamma(\mathbf{r}) \; d\mathbf{r}$$

We must now define approximations to surface integrals over triangles or quaridlaterals.

For triangles we can define the approximation

$$\int_{\sigma_k} f(\mathbf{x})\gamma(\mathbf{r}) \; d\mathbf{r} \approx I_1 \equiv \frac{1}{M} \left\{ \sum_{i=1}^{M} f(\mathbf{x}_i)\gamma(\mathbf{x}_i) \right\} \; \Delta_{\mathbf{r}}(\sigma_k) \tag{1}$$

where $\{\mathbf{x}_i\}_{i=1}^{M}$ are the $M = 3$ vertices of σ_k and where $\Delta_{\mathbf{r}}(\sigma_k)$ is the area of σ_k in the (r, s) plane. Some other possible approximations are

$$\int_{\sigma_k} f(\mathbf{x})\gamma(\mathbf{r}) \; d\mathbf{r} \approx I_2 \equiv f(\mathbf{x}_b)\gamma(\mathbf{x}_b) \; \Delta_{\mathbf{r}}(\sigma_k) \tag{2}$$

$$\approx I_3 \equiv \frac{1}{M} \left\{ \sum_{i=1}^{M} f(\mathbf{x}_i) \right\} \; \Delta_{\mathbf{x}}(\sigma_k) \tag{3}$$

$$\approx I_4 \equiv f(\mathbf{x}_b) \; \Delta_{\mathbf{x}}(\sigma_k) \tag{4}$$

where $\mathbf{x}_b$ is the barycenter (centroid) and $\Delta_{\mathbf{x}}(\sigma_k)$ is the area σ_k in the $\mathbf{x}$ plane. . Apparently all the approximations I_m lead to globaly second-order accurate approximations to the surface integral [**?**]. Note that I_3 and I_4 do not require the mapping derivatives.

7.1.1 Higher-order approximations to integrals

High-order approximations to integrals can be defined by first introducing additional nodes on each element σ. A polynomial interpolant, $I_\sigma(f)$, can be defined in terms of these new degrees of freedom. One can define an interpolant for the metric term as well, $\gamma \approx I_\sigma(\gamma)$. The approximation to the integral on σ can then be defined as the exact integral of the interpolants,

$$\int_{\sigma} f(\mathbf{x})\gamma(\mathbf{r}) \; d\mathbf{r} \approx \int_{\sigma} I_\sigma(f) I_\sigma(\gamma) \; d\mathbf{r} \; . \tag{5}$$

Suppose that we are initially only given the values of $f(\mathbf{x})$ on some set of points (e.g. the nodes of the elements). The values of f can be obtained at the additional nodes introduced above by defining another interpolant that spans multiple elements. Alternatively, if we know the derivatives of f at points on the element then an Hermite interpolant $I_\sigma(f)$ can be defined.

7.1.2 Relating the surface area element to the volume element

We can related the volume Jacobian J to the surface area function, $\gamma = (\mathbf{x}_r \times \mathbf{x}_s) \cdot \mathbf{n}$, as follows. The volume Jacobian is $J = |\partial \mathbf{x}/\partial \mathbf{r}|$, or $J = (\mathbf{x}_r \times \mathbf{x}_s) \cdot \mathbf{x}_t$ (assuming a righ-handed coordinate system). Consider a surface $t = 0$ with normal $\mathbf{n} = \nabla_{\mathbf{x}} t / |\nabla_{\mathbf{x}} t|$. The surface area function is

$$\gamma = (\mathbf{x}_r \times \mathbf{x}_s) \cdot \mathbf{n} = (\mathbf{x}_r \times \mathbf{x}_s) \cdot \nabla_{\mathbf{x}} t / |\nabla_{\mathbf{x}} t|$$

But $\mathbf{x}_t = \alpha \mathbf{n} + \beta \mathbf{x}_r + \gamma \mathbf{x}_s$ where

$$\alpha = \mathbf{x}_t \cdot \mathbf{n} = \mathbf{x}_t \cdot \nabla_{\mathbf{x}} t / |\nabla_{\mathbf{x}}| = 1/|\nabla_{\mathbf{x}} t| \; .$$

We have used $\mathbf{x}_t \cdot \nabla_{\mathbf{x}} t = 1$ (since $\partial t(\mathbf{x})/\partial t = 1$). Thus

$$\begin{aligned} J &= (\mathbf{x}_r \times \mathbf{x}_s) \cdot \mathbf{x}_t = (\mathbf{x}_r \times \mathbf{x}_s) \cdot (\alpha \mathbf{n} + \beta \mathbf{x}_r + \gamma \mathbf{x}_s) \\ &= \alpha (\mathbf{x}_r \times \mathbf{x}_s) \cdot \mathbf{n} \\ &= \alpha \gamma \end{aligned}$$

Therefore $\gamma = J|\nabla_{\mathbf{x}} t|$.

7.1.3 Integration weights for an overlapping surface grid from a stitched hybrid grid

In order to compute integrals on an overlapping surface mesh we can first build a hybrid mesh that fills a gap between the surface grids with an unstructured grid. The function `SurfaceStitcher` will perform this operation.

Given a hybrid mesh for the surface we can define a numerical formula (quadrature) of the form

$$\int_{\mathcal{S}} f(\mathbf{x}) d\mathbf{x} \approx \sum_j f(\mathbf{x}_j) w_j \tag{6}$$

where the function is evaluated at some points $\mathbf{x}_j$ (e.g. cell centroids or cell nodes) and where w_j are some weights.

From this formula, we can define another quadrature that only uses values of f on the grid points of the original overlapping (structured) grid, denoted by $f^k_{\mathbf{i}}$ for a point $\mathbf{i}$ on grid k. To do this we determine how to interpolate $f(\mathbf{x}_j)$ from valid points on the overlapping surface grid,

$$f(\mathbf{x}_j) \approx \sum_{\mathbf{j} \in J_j} \alpha^j_{\mathbf{j}} f^k_{\mathbf{j}} \ . \tag{7}$$

For example if $\mathbf{x}_j$ is the centroid of a quadrilateral on the original structured grid then $f(\mathbf{x}_j)$ could be the average of the 4 neighbours. Substituting the interpolation formula (7) for $f(\mathbf{x}_j)$ into the quadrature formula (6) results in a new quadrature formula for the overlapping grid:

$$\int_{\mathcal{S}} f(\mathbf{x}) d\mathbf{x} \approx \sum_{\mathbf{j}} f^k_{\mathbf{j}} \hat{w}_{\mathbf{j}}$$

7.2 Results

Table 1 shows some results for the circle-in-a-channel grid which is a circle of radius $\frac{1}{2}$ embedded in an square $[-2,2]^2$. Note that the surface area of the embedded circle is computed very accurately since the trapezoidal rule is exponentially accurate on a periodic region.

Table 1 also shows results for a sphere-in-a-box grid.

grid	h_0/h	err-vol	err-surf
cic2	1	$6.71e-2$	$3.6e-15$
cic3	2	$1.59e-2$	$1.1e-14$
cic4	4	$4.24e-3$	$1.8e-14$

grid	h_0/h	err-vol	err-surf
sib1	1	$1.44e-1$	$2.05e-1$
sib2	2	$3.27e-2$	$5.47e-2$
sib2x2	4	$8.33e-3$	$1.40e-2$

Table 1: Left: Errors in computing the volume and surface-area of a circle-in-a-channel grid which is a circle of radius $\frac{1}{2}$ embedded in a square $[-2,2]^2$. The volume is $16-\pi/4 \approx 15.2146$ while the surface-area is $16+\pi \approx 19.14159$. Right: Errors in computing the volume and surface area for a sphere-in-a-box grid which consists of a sphere of radius $\frac{1}{2}$ embedded in a box $[-2,2]^2$. These results were computed with the left-null vector approach.

grid	h_0/h	1	x	y	z	x^2	y^2	z^2
sibe2	1	$9.9e-03$	$2.8e-03$	$1.0e-03$	$1.3e-02$	$4.2e-04$	$1.3e-04$	$3.0e-03$
sibe4	2	$2.3e-03$	$5.4e-04$	$4.2e-04$	$3.5e-03$	$4.3e-05$	$8.5e-06$	$5.5e-04$
sibe8	4	$6.3e-04$	$2.6e-04$	$1.5e-04$	$1.2e-03$	$8.5e-06$	$2.5e-05$	$1.9e-04$
sibe16	4	$1.7e-04$	$8.6e-05$	$4.4e-05$	$2.6e-04$	$1.3e-05$	$1.8e-07$	$3.0e-05$

Table 2: Left: Errors in the numerical integration of the functions $1, x, y, z, x^2, y^2, z^2$ on the surface of a sphere. These results where computed with the hybrid-grid approach.

7.3 Sample usage

To use the `Integrate` class you should follow the example given below. (file `Overture/examples/ti.C`)

7.4 Member Functions

7.5 constructor

Integrate()

Description: Default constructor.

7.6 constructor

Integrate(CompositeGrid & cg_)

Description:

cg_ (input) : supply a grid on which to integrate.

7.7 updateToMatchGrid

int
updateToMatchGrid(CompositeGrid & cg_)

Description: Call this routine to supply a grid or to indicate that the grid has changed.

cg (input) : supply a grid on which to integrate.

7.8 defineSurface

int
defineSurface(const int & surfaceNumber, const int & numberOfFaces_, IntegerArray & boundary)

Description: Specify the sides of grids that define a "surface". A surface represents some subset of the boundary of an entire domain. For example, for the sphere-in-a-box grid a surface could represent the surface of the sphere. To define a surface you must supply:

surfaceNumber (input) : a surface identifier. This value must be bigger than or equal to zero. Normally surfaces should be numbered starting from zero.

numberOfFaces (input) : the number of faces that make up the surface.

boundary (input): boundary(3,numberOfFaces) : (side,axis,grid)=boundary(0:2,i) i=0,1,...numberOfFaces. To define a surface you must supply a list of sides of grids.

7.9 updateForAMR

int
updateForAMR(CompositeGrid & cgu)

Description: Call this function when AMR grids have changed. This will cause the arrays of integration weights for AMR grids to be destroyed. They will be regenerated as needed.

7.10 numberOfFacesOnASurface

int
numberOfFacesOnASurface(const int surfaceNumber) const

Description: Return the number of faces that form a given surface. For AMR grids this will return of the total number of faces including AMR grids (assuming useAdaptiveMeshRefinementGrids(true) has been set).

Return value: number of faces for surface.

7.11 getBoundaryDefinition

const BodyDefinition &
getBodyDefinition() const

Description: Return the BodyDefinition object which defines the relationship between grids and boundaries.

7.12 getFace

int
getFace(const int surfaceNumber, const int face,
int & side, int & axis, int & grid) const

Description: Return the data for a particular face of a surface.

For AMR grids this will return of the data for AMR grids as well (assuming useAdaptiveMeshRefinementGrids(true) has been set).

surface,face (input) : return info for this surface and face.

side,axis,grid (output): this face corresponds to these values.

Return value: 0 for success.

7.13 getSurfaceStitcher

SurfaceStitcher*
getSurfaceStitcher() const

Description: Return a pointer to the surface stitcher (if it exists)

7.14 setInteractiveStitching

void
setInteractiveStitching(bool trueOrFalse)

Description: Turn on interactive stitching (for debugging)

7.15 integrationWeights

RealCompositeGridFunction &
integrationWeights()

Description: Return the integration weights.

Return value: a grid function that holds the integration weights.

7.16 leftNullVector

RealCompositeGridFunction &
leftNullVector()

Description: Return the left null vector of the Neumann problem. This vector is related to the integration weights.

Return value: a grid function that holds the left null vector.

7.17 surfaceIndex

int
surfaceIndex(int surfaceNumber)

Access level: protected.

Description: For a given surfaceNumber determine the surfaceIndex such that surfaceIdentifier(surfaceIndex)==surfaceNumber. Return -1 if no match is found.

surfaceNumber (input) : the surface ID for a user defined surface.

Return value: the index into the surfaceIdentifier array, or -1 if no match exists.

7.18 surfaceIntegral

real
surfaceIntegral(const RealCompositeGridFunction & u, const int & surfaceNumber = -1)

Description: Compute the surface integral of u.

u (input) : function to integrate. This function must be defined at the appropriate points.

surfaceNumber (input) : the surface identifier as defined through a call to `defineSurface`. If no surfaceNumber is specified then the entire surface will be integrated.

Return value: The integral of u.

Author: WDH

7.19 useAdaptiveMeshRefinementGrids

void
useAdaptiveMeshRefinementGrids(bool trueOrFalse = true)

Description: Indicate whether AMR grids should be used when computing integrals on grid functions that have AMR. If false, only the base grids are used in the integration.

trueOrFalse (input) :

Author: WDH

7.20 useHybridGrids

void
useHybridGrids(bool trueOrFalse = true)

Description: Indicate whether hybrid grids should be used to compute the weights for integrals.

trueOrFalse (input) :

Author: WDH

7.21 surfaceIntegral

int
surfaceIntegral(const RealCompositeGridFunction & u,
const Range & C,
RealArray & integral,
const int & surfaceNumber = -1)

Description: Compute the surface integral of u, one or more components.

u (input) : function to integrate. This function must be defined at the appropriate points.

C (input) : integrate these components.

integral (output): array of values, `integral(C)`, the integrals of the components.

surfaceNumber (input) : the surface identifier as defined through a call to `defineSurface`. If no surfaceNumber is specified then the entire surface will be integrated.

Note: For AMR grids one should call updateForAMR(cg) after a AMR regridding step and before calling this function.

Note: Currently the surface integral for AMR grids only works in 2D and when there is a single grid on the surface – i.e. overlapping surface grids are not handled yet.

Author: WDH

7.22 volumeIntegral

real
volumeIntegral(const RealCompositeGridFunction & u)

Description: Compute the volume integral of u.

u (input) : function to integrate. This function must be defined at the appropriate points.

Author: WDH

7.23 computeStitchedSurfaceWeights

int
computeStitchedSurfaceWeights(int surfaceNumber =-1)

Description: This is a protected routine that computes surface weights by building a hybrid grid on the surface using the advacing front algorithm.

Author: Initial version: Kyle Chand. New version by wdh.

8 TridiagonalSolver: Solve sets of tridiagonal (or pentadiagonal) systems or block tridiagonal systems

The `TridiagonalSolver` class can be used to solve tridiagonal (or pentadiagonal) systems and block tridiagonal systems. Sets of tridiagonal (pentadiagonal) systems can be solved such as those that are formed when line smoothing is performed on a 2 or 3 dimensional grid or an ADI type method is used to solve a PDE. Currently only blocks or size 2 or 3 are implemented.

The two basic steps to solve a tridiagonal (pentadiagonal) system are to first factor the system by calling the `factor` member function and then solve the system using `solve`.

There are three types of boundary conditions supported, `normal`, `extended` and `periodic`. A `normal` matrix tridiagonal is of the form

$$\text{normal} = \begin{bmatrix} b_0 & c_0 & & & & \\ a_1 & b_1 & c_1 & & & \\ & a_2 & b_2 & c_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & a_n & b_n \end{bmatrix}$$

The extended matrix allows extra entries on the first and last rows (required by some PDE boundary conditions)

$$\text{extended} = \begin{bmatrix} b_0 & c_0 & a_0 & & & \\ a_1 & b_1 & c_1 & & & \\ & a_2 & b_2 & c_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & c_n & a_n & b_n \end{bmatrix}$$

The periodic case is

$$\text{periodic} = \begin{bmatrix} b_0 & c_0 & & & & a_0 \\ a_1 & b_1 & c_1 & & & \\ & a_2 & b_2 & c_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{n-1} & b_{n-1} & c_{n-1} \\ c_n & & & & a_n & b_n \end{bmatrix}$$

A `normal` pentadiagonal matrix is of the form

$$\text{normal} = \begin{bmatrix} c_0 & d_0 & e_0 & & & & & \\ b_1 & c_1 & d_1 & e_1 & & & & \\ a_2 & b_2 & c_2 & d_2 & e_2 & & & \\ & a_3 & b_3 & c_3 & d_3 & e_3 & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & a_{n-2} & b_{n-2} & c_{n-2} & d_{n-2} & e_{n-2} \\ & & & & a_{n-1} & b_{n-1} & c_{n-1} & d_{n-1} \\ & & & & & a_n & b_n & c_n \end{bmatrix}$$

The `extended` pentadigonal matrix allows extra equations in the first two and last two rows,

$$\text{extended} = \begin{bmatrix} c_0 & d_0 & e_0 & a_0 & b_0 & & & \\ b_1 & c_1 & d_1 & e_1 & a_1 & & & \\ a_2 & b_2 & c_2 & d_2 & e_2 & & & \\ & a_3 & b_3 & c_3 & d_3 & e_3 & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & a_{n-2} & b_{n-2} & c_{n-2} & d_{n-2} & e_{n-2} \\ & & & e_{n-1} & a_{n-1} & b_{n-1} & c_{n-1} & d_{n-1} \\ & & & d_n & e_n & a_n & b_n & c_n \end{bmatrix}$$

The periodic pentadigonal case is

$$\text{periodic} = \begin{bmatrix} c_0 & d_0 & e_0 & & & & a_0 & b_0 \\ b_1 & c_1 & d_1 & e_1 & & & & a_1 \\ a_2 & b_2 & c_2 & d_2 & e_2 & & & \\ & a_3 & b_3 & c_3 & d_3 & e_3 & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & a_{n-2} & b_{n-2} & c_{n-2} & d_{n-2} & e_{n-2} \\ e_{n-1} & & & & a_{n-1} & b_{n-1} & c_{n-1} & d_{n-1} \\ d_n & e_n & & & & a_n & b_n & c_n \end{bmatrix}$$

Here is an example code that shows how to use the TridiagonalSolver (file `Overture/tests/trid.C`)

8.1 Member Functions

8.2 constructor

TridiagonalSolver()

Description: Use this class to solve a tridiagonal system or a pentadiagonal system. The system may be block tridiagonal. There may be multiple independent tridiagonal (pentadiagonal) systems to be solved. The basic tridiagonal system is (`type=normal`)

```
    | b[0] c[0]                      |
    | a[1] b[1] c[1]                 |
A = |      a[2] b[2] c[2]            |
    |            .    .    .        |
    |                a[.] b[.] c[.]  |
    |                     a[n] b[n]  |
```

We can also solve the `type=periodic`

```
    | b[0] c[0]                 a[0] |
    | a[1] b[1] c[1]                 |
A = |      a[2] b[2] c[2]            |
    |            .    .    .        |
    |                a[.] b[.] c[.]  |
    | c[n]                a[n] b[n]  |
```

and the `type=extended`

```
    | b[0] c[0] a[0]                 |
    | a[1] b[1] c[1]                 |
A = |      a[2] b[2] c[2]            |
    |            .    .    .        |
    |                a[.] b[.] c[.]  |
    |                c[n] a[n] b[n]  |
```

which may occur with certain boundary conditions.

This class expects the matrices a,b,c to be passed separately and to be of the form

- a(I1,I2,I3), b(I1,I2,I3), c(I1,I2,I3) : if the block size is 1.
- a(b,b,I1,I2) : if the block size is b> 1.

The '`axis`' argument to the member functions indicates which of I1,I2 or I3 represents the axis along which the tridiagonal matrix extends. The other axes can be used to hold independent tridiagonal systems. Thus if `axis=0` then `a(i1,i2,i3) i1=0,1,2,...,n` are the entries in the tridiagonal matrix for each fixed i2 and i3. If `axis=1` then `a(i1,i2,i3) i2=0,1,2,...,n` are the entries in the tridiagonal matrix for fixed i1 and i3.

8.3 factor

int
factor(RealArray & a_,
RealArray & b_,
RealArray & c_,
const SystemType & type_ =normal,
const int & axis_ =0,
const int & block =1)

Description: Factor the tri-diagonal (block) matrix defined by (a,b,c). NOTE: This routine keeps a reference to (a,b,c) and factors in place.

a,b,c (input/output) : on input the 3 diagonals, on output the LU factorization

type (input) : One of `normal`, `periodic` or `extended`.

axis (input) : 0, 1, or 2. See the comments below.

block (input) : block size. If block=2 or 3 then the matrix is block tridiagonal.

Notes: This class expects the matrices a,b,c to be of the form

- a(I1,I2,I3), b(I1,I2,I3), c(I1,I2,I3) : if the block size is 1.
- a(b,b,I1,I2), b(b,b,I1,I2), c(b,b,I1,I2) : if the block size is b> 1.

The '`axis`' argument to the member functions indicates which of I1,I2 or I3 represents the axis along which the tridiagonal matrix extends. The other axes can be used to hold independent tridiagonal systems. Thus if `axis=0` then `a(i1,i2,i3) i1=0,1,2,...,n` are the entries in the tridiagonal matrix for each fixed i2 and i3. If `axis=1` then `a(i1,i2,i3) i2=0,1,2,...,n` are the entries in the tridiagonal matrix for fixed i1 and i3.

8.4 factor

int
factor(RealArray & a_,
RealArray & b_,
RealArray & c_,
RealArray & d_,
RealArray & e_,
const SystemType & type_ =normal,
const int & axis_ =0,
const int & block =1)

Description: Factor the penta-diagonal (block) matrix defined by (a,b,c,d,e). NOTE: This routine keeps a reference to (a,b,c,d,e) and factors in place.

a,b,c,d,e (input/output) : on input the 5 diagonals, on output the LU factorization

type (input) : One of `normal`, `periodic` or `extended`.

axis (input) : 0, 1, or 2. See the comments below.

block (input) : block size. If block=2 or 3 then the matrix is block tridiagonal.

Notes: This class expects the matrices a,b,c to be of the form

- a(I1,I2,I3), b(I1,I2,I3), c(I1,I2,I3), d(I1,I2,I3), e(I1,I2,I3) : if the block size is 1.
- a(b,b,I1,I2), b(b,b,I1,I2), c(b,b,I1,I2) d(b,b,I1,I2), e(b,b,I1,I2) : if the block size is b> 1.

The '`axis`' argument to the member functions indicates which of I1,I2 or I3 represents the axis along which the tridiagonal matrix extends. The other axes can be used to hold independent tridiagonal systems. Thus if `axis=0` then `a(i1,i2,i3) i1=0,1,2,...,n` are the entries in the tridiagonal matrix for each fixed i2 and i3. If `axis=1` then `a(i1,i2,i3) i2=0,1,2,...,n` are the entries in the tridiagonal matrix for fixed i1 and i3.

8.5 solve

real
sizeOf(FILE *file =NULL) const

Description: Return number of bytes allocated by Oges; optionally print detailed info to a file

file (input) : optinally supply a file to write detailed info to. Choose file=stdout to write to standard output.

Return value: the number of bytes.

8.6 solve

int
solve(const RealArray & r_ this is not really const,
const Range & R1 =nullRange,
const Range & R2 =nullRange,
const Range & R3 =nullRange)

Description: Solve a set of tri-diagonal systems (or penta-diagonal systems).

r_ (input/output) : rhs vector on input, solution on output. This is declared const to avoid compiler warnings.

R1,R2,R3: Specifies which systems to solve. By default all the systems are solved. These Ranges must be a subset of the collection of systems that are found in the matrices passed to the `factor` function. One of these is arguments is ignored, the one corresponding to the axis along which the tridiagonal system extends.

9 FortranIO : Write Fortran formatted or unformated files from C++

Use this class to write fortran files from C++.

9.1 constructor

FortranIO()

Description: Build a FortranIO object.

9.2 open

int
open(const aString & fileName,
const aString & fileForm,
const aString & fileStatus,
const int & fortranUnitNumber =25)

Description: Open a fortran file with a fortran statement of the form:

```
open (unit=io, file=fileName,form=fileForm,status=fileStatus)
```

fileName (input) : name of the file.

fileForm (input) : a valid fortran file format such as "formatted" or "unformatted". *** only "unformatted" is currently supported.

fileStatus (input) : a valid file status such as "old", "new", "unknown"

fortranUnitNumber (input) : a positive integer.

9.3 close

int
close()

Description: Close a fortran file.

9.4 print(int)

int
print(const int & i)

Description: Save an int in the file.

9.5 print(float)

int
print(const float & f)

Description:

9.6 print(double)

int
print(const double & d)

Description:

9.7 print(int*)

int
print(const int *a, const int & count)

Description: Save an array of values.

9.8 print(float*)

int
print(const float *a, const int & count)

Description: Save an array of values.

9.9 print(double*)

int
print(const double *a, const int & count)

Description: Save an array of values.

9.10 print(aString)

int
print(const aString & label)

Description:

9.11 print(intArray)

int
print(const intArray & u)

Description:

9.12 print(floatArray)

int
print(const floatArray & u)

Description:

9.13 print(doubleArray)

int
print(const doubleArray & u)

Description:

9.14 print(intArray,floatArray)

int
print(const intArray & u, const floatArray & v)

Description: Output an int and float array.

9.15 print(intArray,doubleArray)

int
print(const intArray & u, const doubleArray & v)

Description:

9.16 read(int)

int
read(const int & i)

Description: Save an int in the file.

9.17 read(float)

int
read(const float & f)

Description:

9.18 read(double)

int
read(const double & d)

Description:

9.19 read(int*)

int
read(const int *a, const int & count)

Description: Save an array of values.

9.20 read(float*)

int
read(const float *a, const int & count)

Description: Save an array of values.

9.21 read(double*)

int
read(const double *a, const int & count)

Description: Save an array of values.

9.22 read(aString)

int
read(const aString & label)

Description:

9.23 read(intArray)

int
read(const intArray & u)

Description: Read in an array – the array must be dimensioned to the correct size.

9.24 read(floatArray)

int
read(const floatArray & u)

Description: Read in an array – the array must be dimensioned to the correct size.

9.25 read(doubleArray)

int
read(const doubleArray & u)

Description: Read in an array – the array must be dimensioned to the correct size.

10 Reference Counted Objects

10.1 Introduction

This section describes our notion of reference counted objects and describes the class `ListOfReferenceCountedObjects`.

Grids, grid functions and A++ arrays are all reference counted objects.

A reference counted object can be extremely useful for a number of reasons

- a reference counted objects can be easily shared between different classes without the need to use pointers.
- a reference counted object can automatically delete itself when it's reference count goes to zero.
- Someone who makes a reference to a reference counted object does not have to worry that the object will be deleted by someone else.

Any object can be made into a reference counted object by supplying certain member functions and by following certain rules. We will describe how to do this later in this section.

A reference counted object extends the notion of a reference in C++. The statement

```
realArray & u = v;
```

makes `u` become a reference (alias) for v. Similiarly the statements

```
realArray u;
u.reference(v);
```

also make u a reference for v; In both cases changing `u` with a statement like `u=5.;` will cause `v` to also change. However, the latter method for creating a reference is more dynamic. For example, at a later stage `u` can reference a different array, `u.reference(w)` or `u` can break the reference `u.breakReference()`. When a reference is broken `u` will be given it's own copy of the array data.

10.2 How to Write a Reference Counted Class

In this section we describe how to write a reference counted class. Examples of reference counted classes are `MappedGrid`, `GridCollection`, `typeMappedGridFunction` and `typeGridCollectionFunction`.

Here is an example of a header file for a reference counted class (file `Overture/examples/ReferenceCounted-Class.h`)

A reference counted class should

- be derived from the `ReferenceCounting` class.
- contain a copy constructor which can be a deep or shallow copy.
- have an assignment operator which is a deep copy.
- have a `reference` function
- have a `breakReference` function
- create another class to hold all the data associated with the class (class RCData in the example).
- contain a pointer to a class that holds the data for the object
- a private section with functions `reference`, assignment, and `virtualConstructor`, see example. These member functions allow all class's that are derived from the `ReferenceCounting` class to be put in a list.

Here is the implementation of `ReferenceCountedClass` (file `Overture/examples/ReferenceCountedClass.C`)

A reference counted class should be derived from the `ReferenceCounting` class (file `/ReferenceCounting.h`)

10.3 Class ListOfReferenceCountedObjects

This is a template list class for holding holding reference counted objects.

The class is declared as

```
template<class T>
class ListOfReferenceCountedObjects : public ReferenceCounting
```

so that `T` is the generic name of the class whose instances will be placed in the list.

10.3.1 Constructors

`DifferentialAndBoundaryOperators( )`	Default constructor
`ListOfReferenceCountedObjects()`	default constructor
`ListOfReferenceCountedObjects(int numberOfElements)`	Create a list with a given number of elements

10.3.2 Public Member Functions

Here are the public member functions.

`ListOfReferenceCountedObjects& operator=(ListOfReferenceCountedObjects&)`

`void addElement(int index)`	Add an object to the list
`void addElement()`	Add an object to the end of the list
`void addElement(T & t, int index)`	Add an object and reference to t
`void addElement(T & t )`	Add an object to the end, and reference to t
`int getLength()`	Get length of list.
`T& operator[](int index)`	Reference the object at a given location.
`void deleteElement(T & X)`	Find an element on the list and delete it
`void deleteElement(int index)`	Delete the element at the given index
`void deleteElement()`	Delete the element appearing last in the list
`void swapElements(int i, int j)`	Swap two elements (for sorting)
`int getIndex(T & X)`	get the index for an element
`void reference(ListOfReferenceCountedObjects<T> & list )`	reference one list to another
`void breakReference()`	break any references with this list

10.3.3 Examples

This class supports two types of reference counting. The suggested way to do reference counting is to use the reference and breakReference member functions as shown in the examples below. If instead you want to keep pointers to ListOfReferenceObjects then you can use the incrementReferenceCount(), decrementReferenceCount() and getReferenceCount() member functions (derived from the ReferenceCounting in order to mange references). With this latter mode of reference counting it is up to you to call delete when the reference count reaches zero.

Typical Usage:

```
floatArray a(10),b(5);
ListOfReferenceCountedObjects<floatArray> list,list2;
list.addElement();      // add an element
list[0].reference(a);   // reference to array a
list.addElement(b);     // add element and reference to b in one step
list[1]=b;
list[0]=1.;             //  same as a=1.;

list2.reference(list);  // list2 and list are now the same
list2[0]=2.;            // same as list[0]=2.;
list2.breakReference();
list2[0]=5.;            // does not change list[0]
```

Index