

# pbdbASE

December 13, 2013

---

pbdbASE-package

*Core pbd Classes and Methods*

---

## Description

A package contains the basic methods for dealing with distributed data types, as well as the data types themselves.

## Details

Package: pbdBASE  
Type: Package  
License: MPL  
LazyLoad: yes

This package requires an MPI library (OpenMPI, MPICH2, or LAM/MPI).

## Author(s)

Drew Schmidt <schmidt AT math.utk.edu>, Wei-Chen Chen, George Ostrouchov, and Pragneshkumar Patel.

## References

Programming with Big Data in R Website: <http://r-pbd.org/>

---

InitGrid

*Initialize Process Grid*

---

## Description

Manages the creation of BLACS context grids.

## Usage

```
init.grid(NPROW, NPCOL, ICTXT, ..., quiet = FALSE)
blacs_gridinit(ICTXT, NPROW, NPCOL, ..., quiet = FALSE)
```

## Arguments

NPROW	number of process rows. Can be missing; see details.
NPCOL	number of process columns. Can be missing; see details.
ICTXT	BLACS context number.
...	additional arguments.
quiet	logical; controls whether or not information about grid size should be printed.

## Details

`blacs_gridinit()` is for experienced users only. It is a shallow wrapper of the BLACS routine `BLACS_GRIDINIT`, with the addition of creating the `.__blacs_gridinfo_ICTXT` objects, as described below.

The remainder of this section applies only to `init.grid()`.

If `ICTXT` is missing, three variables will be created in the `.pbdBASEEnv` environment:

```
.__blacs_gridinfo_0
.__blacs_gridinfo_1
.__blacs_gridinfo_2
```

These variables store the BLACS process grid information for the BLACS context corresponding to the trailing digit of the variable. Most users should invoke `init.grid()` in this fashion, namely with `ICTXT` missing, and only do so once.

Contexts 0, 1, and 2 are reserved. Additional custom contexts are possible to create, but they must be integers  $\geq 3$ .

Context 0 is the “full” process grid of `NPROW` by `NPCOL` processes; context 1 is the process grid consisting of 1 process row and `NPROW*NPCOL` processes columns; context 2 is the process grid consisting of `NPROW*NPCOL` processes rows and 1 process column. These contexts can be redundant depending on the number of processes available.

BLACS contexts have important internal use, and advanced users familiar with ScaLAPACK might find some advantage in directly manipulating these process grids. Most users should not need to directly manage BLACS contexts, in this function or elsewhere.

If the `NPROW` and `NPCOL` values are missing, then a best process grid will be chosen for the user based on the total available number of processes. Here “best” means as close to a square grid as possible.

The variables `.__blacs_gridinfo_ICTXT` are just storage mechanisms to avoid needing to directly invoke the BLACS routine `BLACS_GRIDINFO`.

Additionally, another variable is created in the `.pbdBASEEnv` environment, namely `.__blacs_initialized`. Its existence is to alert `finalize()` to shut down BLACS communicators, if necessary, to prevent memory leaks.

**Value**

Silently returns 0 when successful. Additionally, several variables are created in the .pbdBASEEnv environment. See Details section.

**See Also**

[BLACS](#)

**Examples**

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdBASE, quiet = TRUE)
init.grid()

finalize()

## End(Not run)
```

---

Gridexit	<i>gridexit</i>
----------	-----------------

---

**Description**

Frees a BLACS context.

**Usage**

```
gridexit(ICTXT, ..., override = FALSE)
```

**Arguments**

ICTXT	BLACS context number.
...	additional arguments.
override	logical; if TRUE, ignores normal check preventing the closing of ICTXT values of 0, 1, and 2.

**Details**

For advanced users only.

The function frees the requested BLACS context. It is a trivial wrapper for the BLACS routine BLACS\_GRIDEXIT. Also removes the object `.__blacs_gridinfo_ICTXT`.

Contexts 0, 1, and 2 can not be freed in this way unless the argument `override=FALSE`. This will probably break something and I do not recommend it.

**Value**

Silently returns 0 when successful. Silently returns 1 when requested ICTXT does not exist.

**See Also**

[InitGrid](#)

---

 BLACS

---

*Get BLACS Context Grid Information*


---

**Description**

Grabs the existing BLACS context grid information.

**Usage**

```
blacs(ICTXT = 0)
```

**Arguments**

ICTXT                  BLACS context number.

**Details**

BLACS contexts have important internal use, and advanced users familiar with ScaLAPACK might find some advantage in directly manipulating these process grids. Most users should not need to directly manage BLACS contexts, in this function or elsewhere.

The function effectively serves as a shorthand for

```
eval(parse(text=paste("__blacs_gridinfo_", ICTXT, sep="")))
```

**Value**

Returns a list with 5 elements: NPROW and NPCOL, the number of process rows and columns respectively; ICTXT, the associated BLACS context number; MYROW and MYCOL, the current process' row and column position in the process grid.

**See Also**

[InitGrid](#)

**Examples**

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdBASE, quiet = TRUE)
init.grid()

mygrid <- blacs(0)

comm.print(mygrid)

finalize()

## End(Not run)
```

---

MinCTXT

*Get BLACS Context Grid Information*

---

**Description**

Finds the smallest integers for creating a new BLACS context.

**Usage**

```
minctxt(after = 0)
```

**Arguments**

after                      ignores all values below this integer as possibilities

**Details**

For advanced users only.

Returns the smallest integer which could become a new BLACS context value.

For example, if contexts 0, 1, and 2 are taken, and after=0, then the function returns 3. If 0, 1, 2, and 5 are taken, the function returns 3 if after=0, but returns 6 if after=4.

The function is useful when a transitory grid is needed, such as for reading in data onto a subset of processors before distributing out to the full grid.

**Value**

Returns the minimum value.

**See Also**

[InitGrid](#)

---

**BLACS Exit***BLACS Exit*

---

**Description**

Shuts down all BLACS communicators.

**Usage**

```
blacsexit(CONT = TRUE)
```

**Arguments**

CONT                    logical; determines whether or not to shut down *all* MPI communicators

**Details**

If the user wishes to shut down BLACS communicators but still have access to MPI, then call this function with CONT=TRUE. Calling blacsexit(CONT=FALSE) will shut down all MPI communicators, equivalent to calling

```
> blacsexit(CONT=TRUE) > finalize(mpi.finalize=TRUE)
```

This function is automatically invoked if BLACS communicators are running and finalize() is called.

**Value**

Has an invisible return of 0 when successful.

**See Also**

[InitGrid](#)

**Examples**

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdBASE, quiet = TRUE)
init.grid()

blacsexit()

finalize()

## End(Not run)
```

**Description**

Grabs the existing BLACS context grid information.

**Usage**

```
pnum(ICTXT, PROW, PCOL)
pcoord(ICTXT, PNUM)
```

**Arguments**

ICTXT	BLACS context number.
PROW, PCOL	BLACS grid location row/column
PNUM	process rank

**Details**

For advanced users only. These functions are simple recreations of the BLACS routines BLACS\_PNUM and BLACS\_PCOORD. The former gets the process number associated with the BLACS process grid location c(MYPROW, MYPCOL), while the latter does the reverse.

**Value**

pnum returns an integer; pcoord returns a list containing elements PROW and PCOL.

**See Also**

[BLACS](#), [InitGrid](#)

**Examples**

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdBASE, quiet = TRUE)
init.grid()

blacs_ <- blacs(ICTXT = 0)

# get the ICTXT = 0 BLACS coordinates for process 0
myCoords <- pcoord(ICTXT = 0, PNUM = 0)

comm.print(myCoords)
```

```
finalize()

## End(Not run)
```

---

BASE Global Environment	<i>Global Environment for the pbdBASE Package</i>
-------------------------	---

---

**Description**

The environment for the pbdBASE package where "global" variables are stored.

**Details**

The `._blacs_gridinfo_` and `._blacs_initialized` objects are stored in this environment.

**See Also**

`InitGrid`

---

Ownership	<i>Determining Local Ownership of a Distributed Matrix</i>
-----------	--

---

**Description**

`aa`

**Usage**

```
base.ownany(dim, bldim, ICTXT = 0)
numroc(dim, bldim, ICTXT = 0, fixme = TRUE)
```

**Arguments**

- |                    |  |
|--------------------|--|
| <code>dim</code>   | global dimension                                       |
| <code>bldim</code> | blocking dimension                                     |
| <code>ICTXT</code> | BLACS context  |
| <code>fixme</code> | logical, controls correction of local dimension return |



## Details

For advanced users only.

`numroc()` is a re-implementation at the R level of the ScaLAPACK subroutine NUMROC, which returns the local dimension of the matrix storage, i.e. the dimension for the Data slot of the distributed matrix on that process. The `fixme=` option, if `TRUE`, returns a minimum of 1 for each dimension. If `fixme=FALSE`, then values less than 1 for either dimension are possible, and in this case indicate a lack of local ownership of the global matrix.

`ownany()` is a simple wrapper of `numroc`. The return is the answer to the question 'do I own any of the global matrix?'. Passing a distributed matrix is allowed, but often it is convenient to determine that information without even having a distributed matrix on hand. In this case, explicitly passing the appropriate information to the arguments `dim=`, `bldim=` (and `CTXT=` as necessary, since it defaults to 0) while leaving `x` missing will produce the desired result. See the examples below for more clarity.

The return for each function is local.

## See Also

[BLACS](#), [InitGrid](#)

## Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdBASE, quiet = TRUE)
init.grid()

iown <- ownany(dim=c(4, 4), bldim=c(2, 2), CTXT=0)
comm.print(iown, all.rank=T)

finalize()

## End(Not run)
```

# Index

## \*Topic **BLACS**

- BLACS, [4](#)
- BLACS Exit, [6](#)
- Coords, [7](#)
- Gridexit, [3](#)
- InitGrid, [1](#)
- MinCTXT, [5](#)
- Ownership, [8](#)

## \*Topic **Distributing Data**

- Ownership, [8](#)

## \*Topic **Package**

- pbdbase-package, [1](#)
- .pbdbaseenv (BASE Global Environment), [8](#)

- BASE Global Environment, [8](#)
- base.ownany (Ownership), [8](#)
- BLACS, [3](#), [4](#), [7](#), [9](#)
- blacs (BLACS), [4](#)
- BLACS Exit, [6](#)
- blacs\_gridinit (InitGrid), [1](#)
- blacsexit (BLACS Exit), [6](#)

- Coords, [7](#)

- Gridexit, [3](#)
- gridexit (Gridexit), [3](#)

- init.grid (InitGrid), [1](#)
- InitGrid, [1](#), [4-7](#), [9](#)

- MinCTXT, [5](#)
- minctxt (MinCTXT), [5](#)

- numroc (Ownership), [8](#)

- Ownership, [8](#)
- Ownership (Ownership), [8](#)

- pbdbase-package, [1](#)
- pcoord (Coords), [7](#)
- pnum (Coords), [7](#)