

hypre Reference Manual

— Version 1.14.0b —

Contents

1	Struct System Interface — <i>A structured-grid conceptual interface</i>	5
1.1	Struct Grids —	5
1.2	Struct Stencils —	6
1.3	Struct Matrices —	7
1.4	Struct Vectors —	12
2	SStruct System Interface — <i>A semi-structured-grid conceptual interface</i>	16
2.1	SStruct Grids —	16
2.2	SStruct Stencils —	20
2.3	SStruct Graphs —	21
2.4	SStruct Matrices —	22
2.5	SStruct Vectors —	27
3	IJ System Interface — <i>A linear-algebraic conceptual interface</i>	33
3.1	IJ Matrices —	33
3.2	IJ Vectors —	39
4	Struct Solvers — <i>Linear solvers for structured grids</i>	44
4.1	Struct Solvers —	44
4.2	Struct Jacobi Solver —	44
4.3	Struct PFMG Solver —	47
4.4	Struct SMG Solver —	50
4.5	Struct PCG Solver —	52
4.6	Struct GMRES Solver —	54
4.7	Struct BiCGSTAB Solver —	56
4.8	Struct Hybrid Solver —	57
5	SStruct Solvers — <i>Linear solvers for semi-structured grids</i>	62
5.1	SStruct Solvers —	62
5.2	SStruct PCG Solver —	62
5.3	SStruct GMRES Solver —	65
5.4	SStruct BiCGSTAB Solver —	67
5.5	SStruct SysPFMG Solver —	69
5.6	SStruct Split Solver —	72
5.7	SStruct FAC Solver —	74
5.8	SStruct Maxwell Solver —	79
6	ParCSR Solvers — <i>Linear solvers for sparse matrix systems</i>	83
6.1	ParCSR Solvers —	83
6.2	ParCSR BoomerAMG Solver and Preconditioner —	84
6.3	ParCSR ParaSails Preconditioner —	103
6.4	ParCSR Euclid Preconditioner —	107

6.5	ParCSR Pilut Preconditioner —	109
6.6	ParCSR AMS Solver and Preconditioner —	110
6.7	ParCSR Hybrid Solver —	117
6.8	ParCSR PCG Solver —	128
6.9	ParCSR GMRES Solver —	130
6.10	ParCSR BiCGSTAB Solver —	131
7	Krylov Solvers — <i>A basic interface for Krylov solvers</i>	134
7.1	Krylov Solvers —	134
7.2	PCG Solver —	134
7.3	GMRES Solver —	136
7.4	BiCGSTAB Solver —	138
7.5	CGNR Solver —	139
8	HYPRE's Finite Element Interface — <i>A finite element-based conceptual interface</i> .	141
8.1	HYPRE FEI functions —	141
8.2	HYPRE FEI Matrix functions —	149
8.3	HYPRE FEI Vector functions —	150
8.4	Solver parameters —	152
	Class Graph	157

Copyright (c) 2006 The Regents of the University of California. Produced at the Lawrence Livermore National Laboratory. Written by the HYPRE team. UCRL-CODE-222953. All rights reserved.

This file is part of HYPRE (see <http://www.llnl.gov/CASC/hypre/>). Please see the COPY-RIGHT_and_LICENSE file for the copyright notice, disclaimer, contact information and the GNU Lesser General Public License.

HYPRE is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License (as published by the Free Software Foundation) version 2.1 dated February 1999.

HYPRE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the IMPLIED WARRANTY OF MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the terms and conditions of the GNU General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

1

Struct System Interface

Names

1.1	Struct Grids	5
1.2	Struct Stencils	6
1.3	Struct Matrices	7
1.4	Struct Vectors	12

This interface represents a structured-grid conceptual view of a linear system.

1.1

Struct Grids

Names

	typedef struct hypre_StructGrid_struct* HYPRE_StructGrid	
	<i>A grid object is constructed out of several “boxes”, defined on a global abstract index space</i>	
	int	
	HYPRE_StructGridCreate (MPIComm comm, int ndim, HYPRE_StructGrid *grid)	
	<i>Create an ndim-dimensional grid object</i>	
1.1.1	int	
	HYPRE_StructGridDestroy (HYPRE_StructGrid grid)	
	<i>Destroy a grid object</i>	6
	int	
	HYPRE_StructGridSetExtents (HYPRE_StructGrid grid, int *ilower, int *iupper)	
	<i>Set the extents for a box on the grid</i>	
	int	
	HYPRE_StructGridAssemble (HYPRE_StructGrid grid)	
	<i>Finalize the construction of the grid before using</i>	
1.1.2	int	

HYPRE_StructGridSetPeriodic (HYPRE_StructGrid grid, int *periodic)
Set the periodicity for the grid 6

int
HYPRE_StructGridSetNumGhost (HYPRE_StructGrid grid,
int *num_ghost)
Set the ghost layer in the grid object

1.1.1

int **HYPRE_StructGridDestroy** (HYPRE_StructGrid grid)

Destroy a grid object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

1.1.2

int **HYPRE_StructGridSetPeriodic** (HYPRE_StructGrid grid, int *periodic)

Set the periodicity for the grid.

The argument `periodic` is an `ndim`-dimensional integer array that contains the periodicity for each dimension. A zero value for a dimension means non-periodic, while a nonzero value means periodic and contains the actual period. For example, periodicity in the first and third dimensions for a 10x11x12 grid is indicated by the array [10,0,12].

NOTE: Some of the solvers in `hypr` have power-of-two restrictions on the size of the periodic dimensions.

1.2

Struct Stencils

Names

typedef struct hypr_StructStencil_struct* **HYPRE_StructStencil**
The stencil object

int

HYPRE_StructStencilCreate (int ndim, int size,
 HYPRE_StructStencil *stencil)
*Create a stencil object for the specified number of spatial dimensions and
 stencil entries*

int

HYPRE_StructStencilDestroy (HYPRE_StructStencil stencil)
Destroy a stencil object

1.2.1

int

HYPRE_StructStencilSetElement (HYPRE_StructStencil stencil, int entry,
 int *offset)
Set a stencil entry

7

1.2.1

int
HYPRE_StructStencilSetElement (HYPRE_StructStencil stencil, int entry, int
 *offset)

Set a stencil entry.

NOTE: The name of this routine will eventually be changed to `HYPRE_StructStencilSetEntry`.

1.3

Struct Matrices

Names

typedef struct hypre_StructMatrix_struct* **HYPRE_StructMatrix**
The matrix object

int

HYPRE_StructMatrixCreate (MPI_Comm comm, HYPRE_StructGrid grid,
 HYPRE_StructStencil stencil,
 HYPRE_StructMatrix *matrix)
Create a matrix object

int

HYPRE_StructMatrixDestroy (HYPRE_StructMatrix matrix)
Destroy a matrix object

int

HYPRE_StructMatrixInitialize (HYPRE_StructMatrix matrix)
Prepare a matrix object for setting coefficient values

1.3.1

int

	HYPRE_StructMatrixSetValues (HYPRE_StructMatrix matrix, int *index, int nentries, int *entries, double *values) <i>Set matrix coefficients index by index</i>	9
1.3.2	int HYPRE_StructMatrixAddToValues (HYPRE_StructMatrix matrix, int *index, int nentries, int *entries, double *values) <i>Add to matrix coefficients index by index</i>	9
1.3.3	int HYPRE_StructMatrixSetConstantValues (HYPRE_StructMatrix matrix, int nentries, int *entries, double *values) <i>Set matrix coefficients which are constant over the grid</i>	9
1.3.4	int HYPRE_StructMatrixAddToConstantValues (HYPRE_StructMatrix matrix, int nentries, int *entries, double *values) <i>Add to matrix coefficients which are constant over the grid</i>	10
1.3.5	int HYPRE_StructMatrixSetBoxValues (HYPRE_StructMatrix matrix, int *ilower, int *iupper, int nentries, int *entries, double *values) <i>Set matrix coefficients a box at a time</i>	10
1.3.6	int HYPRE_StructMatrixAddToBoxValues (HYPRE_StructMatrix matrix, int *ilower, int *iupper, int nentries, int *entries, double *values) <i>Add to matrix coefficients a box at a time</i>	10
	int HYPRE_StructMatrixAssemble (HYPRE_StructMatrix matrix) <i>Finalize the construction of the matrix before using</i>	
1.3.7	int HYPRE_StructMatrixSetSymmetric (HYPRE_StructMatrix matrix, int symmetric) <i>Define symmetry properties of the matrix</i>	11
1.3.8	int HYPRE_StructMatrixSetConstantEntries (HYPRE_StructMatrix matrix, int nentries, int *entries) <i>Specify which stencil entries are constant over the grid</i>	11
	int HYPRE_StructMatrixSetNumGhost (HYPRE_StructMatrix matrix, int *num_ghost) <i>Set the ghost layer in the matrix</i>	
1.3.9	int HYPRE_StructMatrixPrint (const char *filename, HYPRE_StructMatrix matrix, int all) <i>Print the matrix to file</i>	11
1.3.10	int	

HYPRE_StructMatrixMatvec (double alpha, HYPRE_StructMatrix A,
HYPRE_StructVector x, double beta,
HYPRE_StructVector y)

Matvec operator 12

1.3.1

int
HYPRE_StructMatrixSetValues (HYPRE_StructMatrix matrix, int *index, int
nentries, int *entries, double *values)

Set matrix coefficients index by index. The **values** array is of length **nentries**.

NOTE: For better efficiency, use HYPRE_StructMatrixSetBoxValues (→1.3.5, *page 10*) to set coefficients a box at a time.

1.3.2

int
HYPRE_StructMatrixAddToValues (HYPRE_StructMatrix matrix, int *index,
int nentries, int *entries, double *values)

Add to matrix coefficients index by index. The **values** array is of length **nentries**.

NOTE: For better efficiency, use HYPRE_StructMatrixAddToBoxValues (→1.3.6, *page 10*) to set coefficients a box at a time.

1.3.3

int
HYPRE_StructMatrixSetConstantValues (HYPRE_StructMatrix matrix, int
nentries, int *entries, double *values)

Set matrix coefficients which are constant over the grid. The **values** array is of length **nentries**.

1.3.4

```
int
HYPRE_StructMatrixAddToConstantValues (HYPRE_StructMatrix matrix,
int nentries, int *entries, double *values)
```

Add to matrix coefficients which are constant over the grid. The `values` array is of length `nentries`.

1.3.5

```
int
HYPRE_StructMatrixSetBoxValues (HYPRE_StructMatrix matrix, int
*ilower, int *iupper, int nentries, int *entries, double *values)
```

Set matrix coefficients a box at a time. The data in `values` is ordered as follows:

```
m = 0;
for (k = ilower[2]; k <= iupper[2]; k++)
  for (j = ilower[1]; j <= iupper[1]; j++)
    for (i = ilower[0]; i <= iupper[0]; i++)
      for (entry = 0; entry < nentries; entry++)
        {
          values[m] = ...;
          m++;
        }
```

1.3.6

```
int
HYPRE_StructMatrixAddToBoxValues (HYPRE_StructMatrix matrix, int
*ilower, int *iupper, int nentries, int *entries, double *values)
```

Add to matrix coefficients a box at a time. The data in `values` is ordered as in `HYPRE_StructMatrixSetBoxValues` (→1.3.5, *page 10*).

1.3.7

```
int
HYPRE_StructMatrixSetSymmetric (HYPRE_StructMatrix matrix, int
symmetric)
```

Define symmetry properties of the matrix. By default, matrices are assumed to be nonsymmetric. Significant storage savings can be made if the matrix is symmetric.

1.3.8

```
int
HYPRE_StructMatrixSetConstantEntries ( HYPRE_StructMatrix matrix, int
nentries, int *entries )
```

Specify which stencil entries are constant over the grid. Declaring entries to be “constant over the grid” yields significant memory savings because the value for each declared entry will only be stored once. However, not all solvers are able to utilize this feature.

Presently supported:

- no entries constant (this function need not be called)
- all entries constant
- all but the diagonal entry constant

1.3.9

```
int
HYPRE_StructMatrixPrint (const char *filename, HYPRE_StructMatrix
matrix, int all)
```

Print the matrix to file. This is mainly for debugging purposes.

1.3.10

```
int
HYPRE_StructMatrixMatvec ( double alpha, HYPRE_StructMatrix A,
HYPRE_StructVector x, double beta, HYPRE_StructVector y )
```

Matvec operator. This operation is $y = \alpha Ax + \beta y$. Note that you can do a simple matrix-vector multiply by setting $\alpha = 1$ and $\beta = 0$.

1.4

Struct Vectors

Names

	typedef struct hypre_StructVector_struct* HYPRE_StructVector <i>The vector object</i>	
	int HYPRE_StructVectorCreate (MPI_Comm comm, HYPRE_StructGrid grid, HYPRE_StructVector *vector) <i>Create a vector object</i>	
	int HYPRE_StructVectorDestroy (HYPRE_StructVector vector) <i>Destroy a vector object</i>	
	int HYPRE_StructVectorInitialize (HYPRE_StructVector vector) <i>Prepare a vector object for setting coefficient values</i>	
1.4.1	int HYPRE_StructVectorClearGhostValues (HYPRE_StructVector vector) <i>Clears the ghostvalues of vector object</i>	13
1.4.2	int HYPRE_StructVectorSetValues (HYPRE_StructVector vector, int *index, double value) <i>Set vector coefficients index by index</i>	13
1.4.3	int HYPRE_StructVectorAddToValues (HYPRE_StructVector vector, int *index, double value) <i>Add to vector coefficients index by index</i>	14
1.4.4	int HYPRE_StructVectorSetBoxValues (HYPRE_StructVector vector, int *ilower, int *iupper, double *values) <i>Set vector coefficients a box at a time</i>	14
1.4.5	int	

	HYPRE_StructVectorAddToBoxValues (HYPRE_StructVector vector, int *ilower, int *iupper, double *values) <i>Add to vector coefficients a box at a time</i>	14
	int HYPRE_StructVectorAssemble (HYPRE_StructVector vector) <i>Finalize the construction of the vector before using</i>	
1.4.6	int HYPRE_StructVectorGetValues (HYPRE_StructVector vector, int *index, double *value) <i>Get vector coefficients index by index</i>	15
1.4.7	int HYPRE_StructVectorGetBoxValues (HYPRE_StructVector vector, int *ilower, int *iupper, double *values) <i>Get vector coefficients a box at a time</i>	15
1.4.8	int HYPRE_StructVectorPrint (const char *filename, HYPRE_StructVector vector, int all) <i>Print the vector to file</i>	15

1.4.1

```
int HYPRE_StructVectorClearGhostValues (HYPRE_StructVector vector)
```

Clears the ghostvalues of vector object. Beneficial to users that re-assemble a vector object (e.g., in time-stepping).

1.4.2

```
int  
HYPRE_StructVectorSetValues (HYPRE_StructVector vector, int *index,  
double value)
```

Set vector coefficients index by index.

NOTE: For better efficiency, use HYPRE_StructVectorSetBoxValues (→1.4.4, *page 14*) to set coefficients a box at a time.

1.4.3

```
int
HYPRE_StructVectorAddToValues (HYPRE_StructVector vector, int *index,
double value)
```

Add to vector coefficients index by index.

NOTE: For better efficiency, use `HYPRE_StructVectorAddToBoxValues` ([→1.4.5, page 14](#)) to set coefficients a box at a time.

1.4.4

```
int
HYPRE_StructVectorSetBoxValues (HYPRE_StructVector vector, int *ilower,
int *iupper, double *values)
```

Set vector coefficients a box at a time. The data in `values` is ordered as follows:

```
m = 0;
for (k = ilower[2]; k <= iupper[2]; k++)
  for (j = ilower[1]; j <= iupper[1]; j++)
    for (i = ilower[0]; i <= iupper[0]; i++)
    {
      values[m] = ...;
      m++;
    }
```

1.4.5

```
int
HYPRE_StructVectorAddToBoxValues (HYPRE_StructVector vector, int
*ilower, int *iupper, double *values)
```

Add to vector coefficients a box at a time. The data in `values` is ordered as in `HYPRE_StructVectorSetBoxValues` ([→1.4.4, page 14](#)).

1.4.6

```
int
HYPRE_StructVectorGetValues (HYPRE_StructVector vector, int *index,
double *value)
```

Get vector coefficients index by index.

NOTE: For better efficiency, use `HYPRE_StructVectorGetBoxValues` ([→1.4.7, page 15](#)) to get coefficients a box at a time.

1.4.7

```
int
HYPRE_StructVectorGetBoxValues (HYPRE_StructVector vector, int *ilower,
int *iupper, double *values)
```

Get vector coefficients a box at a time. The data in `values` is ordered as in `HYPRE_StructVectorSetBoxValues` ([→1.4.4, page 14](#)).

1.4.8

```
int
HYPRE_StructVectorPrint (const char *filename, HYPRE_StructVector vector,
int all)
```

Print the vector to file. This is mainly for debugging purposes.

2

extern **SStruct System Interface**

Names

2.1	SStruct Grids	16
2.2	SStruct Stencils	20
2.3	SStruct Graphs	21
2.4	SStruct Matrices	22
2.5	SStruct Vectors	27

This interface represents a semi-structured-grid conceptual view of a linear system.

2.1

SStruct Grids

Names

2.1.1	typedef struct hypre_SStructGrid_struct* HYPRE_SStructGrid <i>A grid object is constructed out of several structured “parts” and an optional unstructured “part”</i>	17
2.1.2	typedef enum hypre_SStructVariable_enum HYPRE_SStructVariable <i>An enumerated type that supports cell centered, node centered, face centered, and edge centered variables</i>	17
	int HYPRE_SStructGridCreate (MPI_Comm comm, int ndim, int nparts, HYPRE_SStructGrid *grid) <i>Create an ndim-dimensional grid object with nparts structured parts</i>	
2.1.3	int HYPRE_SStructGridDestroy (HYPRE_SStructGrid grid) <i>Destroy a grid object</i>	18
	int HYPRE_SStructGridSetExtents (HYPRE_SStructGrid grid, int part, int *lower, int *upper) <i>Set the extents for a box on a structured part of the grid</i>	
	int	

	HYPRE_SStructGridSetVariables (HYPRE_SStructGrid grid, int part, int nvars, HYPRE_SStructVariable *vartypes) <i>Describe the variables that live on a structured part of the grid</i>	
2.1.4	int HYPRE_SStructGridAddVariables (HYPRE_SStructGrid grid, int part, int *index, int nvars, HYPRE_SStructVariable *vartypes) <i>Describe additional variables that live at a particular index</i>	19
2.1.5	int HYPRE_SStructGridSetNeighborBox (HYPRE_SStructGrid grid, int part, int *ilower, int *iupper, int nbor_part, int *nbor_ilower, int *nbor_iupper, int *index_map) <i>Describe how regions just outside of a part relate to other parts</i>	19
2.1.6	int HYPRE_SStructGridAddUnstructuredPart (HYPRE_SStructGrid grid, int ilower, int iupper) <i>Add an unstructured part to the grid</i>	19
	int HYPRE_SStructGridAssemble (HYPRE_SStructGrid grid) <i>Finalize the construction of the grid before using</i>	
2.1.7	int HYPRE_SStructGridSetPeriodic (HYPRE_SStructGrid grid, int part, int *periodic) <i>Set the periodicity a particular part</i>	20
	int HYPRE_SStructGridSetNumGhost (HYPRE_SStructGrid grid, int *num_ghost) <i>Setting ghost in the sgrids</i>	

2.1.1

```
#define HYPRE_SStructGrid
```

A grid object is constructed out of several structured “parts” and an optional unstructured “part”. Each structured part has its own abstract index space.

2.1.2

```
#define HYPRE_SStructVariable
```

An enumerated type that supports cell centered, node centered, face centered, and edge centered variables. Face centered variables are split into x-face, y-face, and z-face variables, and edge centered variables are split into x-edge, y-edge, and z-edge variables. The edge centered variable types are only used in 3D. In 2D, edge centered variables are handled by the face centered types.

Variables are referenced relative to an abstract (cell centered) index in the following way:

- cell centered variables are aligned with the index;
- node centered variables are aligned with the cell corner at relative index $(1/2, 1/2, 1/2)$;
- x-face, y-face, and z-face centered variables are aligned with the faces at relative indexes $(1/2, 0, 0)$, $(0, 1/2, 0)$, and $(0, 0, 1/2)$, respectively;
- x-edge, y-edge, and z-edge centered variables are aligned with the edges at relative indexes $(0, 1/2, 1/2)$, $(1/2, 0, 1/2)$, and $(1/2, 1/2, 0)$, respectively.

The supported identifiers are:

- `HYPRE_SSTRUCT_VARIABLE_CELL`
- `HYPRE_SSTRUCT_VARIABLE_NODE`
- `HYPRE_SSTRUCT_VARIABLE_XFACE`
- `HYPRE_SSTRUCT_VARIABLE_YFACE`
- `HYPRE_SSTRUCT_VARIABLE_ZFACE`
- `HYPRE_SSTRUCT_VARIABLE_XEDGE`
- `HYPRE_SSTRUCT_VARIABLE_YEDGE`
- `HYPRE_SSTRUCT_VARIABLE_ZEDGE`

NOTE: Although variables are referenced relative to a unique abstract cell-centered index, some variables are associated with multiple grid cells. For example, node centered variables in 3D are associated with 8 cells (away from boundaries). Although grid cells are distributed uniquely to different processes, variables may be owned by multiple processes because they may be associated with multiple cells.

2.1.3

```
int HYPRE_SStructGridDestroy (HYPRE_SStructGrid grid)
```

Destroy a grid object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

2.1.4

```
int
HYPRE_SStructGridAddVariables (HYPRE_SStructGrid grid, int part, int
*index, int nvars, HYPRE_SStructVariable *vartypes)
```

Describe additional variables that live at a particular index. These variables are appended to the array of variables set in `HYPRE_SStructGridSetVariables` (\rightarrow *page 17*), and are referenced as such.

2.1.5

```
int
HYPRE_SStructGridSetNeighborBox (HYPRE_SStructGrid grid, int part, int
*ilower, int *iupper, int nbor_part, int *nbor_ilower, int *nbor_iupper, int
*index_map)
```

Describe how regions just outside of a part relate to other parts. This is done a box at a time.

The indexes `ilower` and `iupper` map directly to the indexes `nbor_ilower` and `nbor_iupper`. Although, it is required that indexes increase from `ilower` to `iupper`, indexes may increase and/or decrease from `nbor_ilower` to `nbor_iupper`.

The `index_map` describes the mapping of indexes 0, 1, and 2 on part `part` to the corresponding indexes on part `nbor_part`. For example, triple (1, 2, 0) means that indexes 0, 1, and 2 on part `part` map to indexes 1, 2, and 0 on part `nbor_part`, respectively.

NOTE: All parts related to each other via this routine must have an identical list of variables and variable types. For example, if part 0 has only two variables on it, a cell centered variable and a node centered variable, and we declare part 1 to be a neighbor of part 0, then part 1 must also have only two variables on it, and they must be of type cell and node.

2.1.6

```
int
HYPRE_SStructGridAddUnstructuredPart (HYPRE_SStructGrid grid, int
ilower, int iupper)
```

Add an unstructured part to the grid. The variables in the unstructured part of the grid are referenced by a global rank between 0 and the total number of unstructured variables minus one. Each process owns some unique consecutive range of variables, defined by `ilower` and `iupper`.

NOTE: This is just a placeholder. This part of the interface is not finished.

2.1.7

```
int
HYPRE_SStructGridSetPeriodic (HYPRE_SStructGrid grid, int part, int
*periodic)
```

Set the periodicity a particular part.

The argument `periodic` is an `ndim`-dimensional integer array that contains the periodicity for each dimension. A zero value for a dimension means non-periodic, while a nonzero value means periodic and contains the actual period. For example, periodicity in the first and third dimensions for a 10x11x12 part is indicated by the array [10,0,12].

NOTE: Some of the solvers in `hypre` have power-of-two restrictions on the size of the periodic dimensions.

2.2

SStruct Stencils

Names

```
typedef struct hypre_SStructStencil_struct* HYPRE_SStructStencil
The stencil object
```

```
int
HYPRE_SStructStencilCreate (int ndim, int size,
HYPRE_SStructStencil *stencil)
Create a stencil object for the specified number of spatial dimensions and stencil entries
```

```
int
HYPRE_SStructStencilDestroy (HYPRE_SStructStencil stencil)
Destroy a stencil object
```

```
int
HYPRE_SStructStencilSetEntry (HYPRE_SStructStencil stencil, int entry,
int *offset, int var)
Set a stencil entry
```

2.3

SStruct Graphs

Names

typedef struct hypre_SStructGraph_struct* **HYPRE_SStructGraph**
The graph object is used to describe the nonzero structure of a matrix

int

HYPRE_SStructGraphCreate (MPI_Comm comm,
 HYPRE_SStructGrid grid,
 HYPRE_SStructGraph *graph)

Create a graph object

int

HYPRE_SStructGraphDestroy (HYPRE_SStructGraph graph)

Destroy a graph object

int

HYPRE_SStructGraphSetStencil (HYPRE_SStructGraph graph, int part,
 int var, HYPRE_SStructStencil stencil)

Set the stencil for a variable on a structured part of the grid

2.3.1

int

HYPRE_SStructGraphAddEntries (HYPRE_SStructGraph graph, int part,
 int *index, int var, int to_part,
 int *to_index, int to_var)

Add a non-stencil graph entry at a particular index 21

2.3.2

int

HYPRE_SStructGraphSetObjectType (HYPRE_SStructGraph graph,
 int type)

Set the storage type of the associated matrix object 22

int

HYPRE_SStructGraphAssemble (HYPRE_SStructGraph graph)

Finalize the construction of the graph before using

2.3.1

int

HYPRE_SStructGraphAddEntries (HYPRE_SStructGraph graph, int part, int
 *index, int var, int to_part, int *to_index, int to_var)

Add a non-stencil graph entry at a particular index. This graph entry is appended to the existing graph entries, and is referenced as such.

NOTE: Users are required to set graph entries on all processes that own the associated variables. This means that some data will be multiply defined.

2.3.2

```
int
HYPRE_SStructGraphSetObjectType (HYPRE_SStructGraph graph, int
type)
```

Set the storage type of the associated matrix object. It is used before AddEntries and Assemble to compute the right ranks in the graph.

NOTE: This routine is only necessary for implementation reasons, and will eventually be removed.

See Also: HYPRE_SStructMatrixSetObjectType (→2.4.6, page 26)

2.4

SStruct Matrices

Names

```
typedef struct hypre_SStructMatrix_struct* HYPRE_SStructMatrix
    The matrix object
```

```
int
HYPRE_SStructMatrixCreate (MPI_Comm comm,
    HYPRE_SStructGraph graph,
    HYPRE_SStructMatrix *matrix)
    Create a matrix object
```

```
int
HYPRE_SStructMatrixDestroy (HYPRE_SStructMatrix matrix)
    Destroy a matrix object
```

```
int
HYPRE_SStructMatrixInitialize (HYPRE_SStructMatrix matrix)
    Prepare a matrix object for setting coefficient values
```

2.4.1 int
HYPRE_SStructMatrixSetValues (HYPRE_SStructMatrix matrix, int part,
 int *index, int var, int nentries,
 int *entries, double *values)
Set matrix coefficients index by index 23

2.4.2 int
HYPRE_SStructMatrixAddToValues (HYPRE_SStructMatrix matrix,
 int part, int *index, int var,
 int nentries, int *entries,
 double *values)
Add to matrix coefficients index by index 24

2.4.3 int

	HYPRE_SStructMatrixSetBoxValues (HYPRE_SStructMatrix matrix, int part, int *ilower, int *iupper, int var, int nentries, int *entries, double *values)	
	<i>Set matrix coefficients a box at a time</i>	25
2.4.4	int HYPRE_SStructMatrixAddToBoxValues (HYPRE_SStructMatrix matrix, int part, int *ilower, int *iupper, int var, int nentries, int *entries, double *values)	
	<i>Add to matrix coefficients a box at a time</i>	25
	int HYPRE_SStructMatrixAssemble (HYPRE_SStructMatrix matrix) <i>Finalize the construction of the matrix before using</i>	
2.4.5	int HYPRE_SStructMatrixSetSymmetric (HYPRE_SStructMatrix matrix, int part, int var, int to_var, int symmetric)	
	<i>Define symmetry properties for the stencil entries in the matrix</i>	26
	int HYPRE_SStructMatrixSetNSSymmetric (HYPRE_SStructMatrix matrix, int symmetric) <i>Define symmetry properties for all non-stencil matrix entries</i>	
2.4.6	int HYPRE_SStructMatrixSetObjectType (HYPRE_SStructMatrix matrix, int type)	
	<i>Set the storage type of the matrix object to be constructed</i>	26
2.4.7	int HYPRE_SStructMatrixGetObject (HYPRE_SStructMatrix matrix, void **object)	
	<i>Get a reference to the constructed matrix object</i>	26
	int HYPRE_SStructMatrixSetComplex (HYPRE_SStructMatrix matrix) <i>Set the matrix to be complex</i>	
2.4.8	int HYPRE_SStructMatrixPrint (const char *filename, HYPRE_SStructMatrix matrix, int all)	
	<i>Print the matrix to file</i>	27

2.4.1

```
int
HYPRE_SStructMatrixSetValues (HYPRE_SStructMatrix matrix, int part, int
*index, int var, int nentries, int *entries, double *values)
```

Set matrix coefficients index by index. The **values** array is of length **nentries**.

NOTE: For better efficiency, use `HYPRE_SStructMatrixSetBoxValues` (→2.4.3, *page 25*) to set coefficients a box at a time.

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

NOTE: The entries in this routine must all be of the same type: either stencil or non-stencil, but not both. Also, if they are stencil entries, they must all represent couplings to the same variable type (there are no such restrictions for non-stencil entries).

If the matrix is complex, then **values** consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: `HYPRE_SStructMatrixSetComplex` (→ *page 23*)

2.4.2

```
int
HYPRE_SStructMatrixAddToValues (HYPRE_SStructMatrix matrix, int part,
int *index, int var, int nentries, int *entries, double *values)
```

Add to matrix coefficients index by index. The **values** array is of length **nentries**.

NOTE: For better efficiency, use `HYPRE_SStructMatrixAddToBoxValues` (→2.4.4, *page 25*) to set coefficients a box at a time.

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

NOTE: The entries in this routine must all be of the same type: either stencil or non-stencil, but not both. Also, if they are stencil entries, they must all represent couplings to the same variable type.

If the matrix is complex, then **values** consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: `HYPRE_SStructMatrixSetComplex` (→ *page 23*)

2.4.3

```
int
HYPRE_SStructMatrixSetBoxValues (HYPRE_SStructMatrix matrix, int
part, int *ilower, int *iupper, int var, int nentries, int *entries, double *values)
```

Set matrix coefficients a box at a time. The data in `values` is ordered as follows:

```
m = 0;
for (k = ilower[2]; k <= iupper[2]; k++)
  for (j = ilower[1]; j <= iupper[1]; j++)
    for (i = ilower[0]; i <= iupper[0]; i++)
      for (entry = 0; entry < nentries; entry++)
        {
          values[m] = ...;
          m++;
        }
```

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

NOTE: The entries in this routine must all be of the same type: either stencil or non-stencil, but not both. Also, if they are stencil entries, they must all represent couplings to the same variable type (there are no such restrictions for non-stencil entries).

If the matrix is complex, then `values` consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: `HYPRE_SStructMatrixSetComplex` ([→ page 23](#))

2.4.4

```
int
HYPRE_SStructMatrixAddToBoxValues (HYPRE_SStructMatrix matrix, int
part, int *ilower, int *iupper, int var, int nentries, int *entries, double *values)
```

Add to matrix coefficients a box at a time. The data in `values` is ordered as in `HYPRE_SStructMatrixSetBoxValues` ([→2.4.3, page 25](#)).

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

NOTE: The entries in this routine must all be of stencil type. Also, they must all represent couplings to the same variable type.

If the matrix is complex, then `values` consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: `HYPRE_SStructMatrixSetComplex` ([→ page 23](#))

2.4.5

```
int
HYPRE_SStructMatrixSetSymmetric (HYPRE_SStructMatrix matrix, int
part, int var, int to_var, int symmetric)
```

Define symmetry properties for the stencil entries in the matrix. The boolean argument `symmetric` is applied to stencil entries on part `part` that couple variable `var` to variable `to_var`. A value of -1 may be used for `part`, `var`, or `to_var` to specify “all”. For example, if `part` and `to_var` are set to -1, then the boolean is applied to stencil entries on all parts that couple variable `var` to all other variables.

By default, matrices are assumed to be nonsymmetric. Significant storage savings can be made if the matrix is symmetric.

2.4.6

```
int
HYPRE_SStructMatrixSetObjectType (HYPRE_SStructMatrix matrix, int
type)
```

Set the storage type of the matrix object to be constructed. Currently, `type` can be either `HYPRE_SSTRUCT` (the default), `HYPRE_STRUCT`, or `HYPRE_PARCSR`.

See Also: `HYPRE_SStructMatrixGetObject` ([→2.4.7, page 26](#))

2.4.7

```
int
HYPRE_SStructMatrixGetObject (HYPRE_SStructMatrix matrix, void
**object)
```

Get a reference to the constructed matrix object.

See Also: `HYPRE_SStructMatrixSetObjectType` (→2.4.6, page 26)

2.4.8

```
int
HYPRE_SStructMatrixPrint (const char *filename, HYPRE_SStructMatrix
matrix, int all)
```

Print the matrix to file. This is mainly for debugging purposes.

2.5

SStruct Vectors

Names

```
typedef struct hypre_SStructVector_struct* HYPRE_SStructVector
    The vector object
```

```
int
HYPRE_SStructVectorCreate (MPI.Comm comm,
    HYPRE_SStructGrid grid,
    HYPRE_SStructVector *vector)
    Create a vector object
```

```
int
HYPRE_SStructVectorDestroy (HYPRE_SStructVector vector)
    Destroy a vector object
```

```
int
HYPRE_SStructVectorInitialize (HYPRE_SStructVector vector)
    Prepare a vector object for setting coefficient values
```

- 2.5.1 `int`
HYPRE_SStructVectorSetValues (HYPRE_SStructVector vector, int part,
 int *index, int var, double *value)
Set vector coefficients index by index 28
- 2.5.2 `int`
HYPRE_SStructVectorAddToValues (HYPRE_SStructVector vector,
 int part, int *index, int var,
 double *value)
Add to vector coefficients index by index 29
- 2.5.3 `int`

	HYPRE_SStructVectorSetBoxValues (HYPRE_SStructVector vector, int part, int *ilower, int *iupper, int var, double *values)	
	<i>Set vector coefficients a box at a time</i>	29
2.5.4	int HYPRE_SStructVectorAddToBoxValues (HYPRE_SStructVector vector, int part, int *ilower, int *iupper, int var, double *values)	
	<i>Add to vector coefficients a box at a time</i>	30
	int HYPRE_SStructVectorAssemble (HYPRE_SStructVector vector)	
	<i>Finalize the construction of the vector before using</i>	
2.5.5	int HYPRE_SStructVectorGather (HYPRE_SStructVector vector)	
	<i>Gather vector data so that efficient GetValues can be done</i>	30
2.5.6	int HYPRE_SStructVectorGetValues (HYPRE_SStructVector vector, int part, int *index, int var, double *value)	
	<i>Get vector coefficients index by index</i>	31
2.5.7	int HYPRE_SStructVectorGetBoxValues (HYPRE_SStructVector vector, int part, int *ilower, int *iupper, int var, double *values)	
	<i>Get vector coefficients a box at a time</i>	31
2.5.8	int HYPRE_SStructVectorSetObjectType (HYPRE_SStructVector vector, int type)	
	<i>Set the storage type of the vector object to be constructed</i>	31
2.5.9	int HYPRE_SStructVectorGetObject (HYPRE_SStructVector vector, void **object)	
	<i>Get a reference to the constructed vector object</i>	32
	int HYPRE_SStructVectorSetComplex (HYPRE_SStructVector vector)	
	<i>Set the vector to be complex</i>	
2.5.10	int HYPRE_SStructVectorPrint (const char *filename, HYPRE_SStructVector vector, int all)	
	<i>Print the vector to file</i>	32

2.5.1

int HYPRE_SStructVectorSetValues (HYPRE_SStructVector vector, int part, int *index, int var, double *value)
--

Set vector coefficients index by index.

NOTE: For better efficiency, use `HYPRE_SStructVectorSetBoxValues` (→2.5.3, *page 29*) to set coefficients a box at a time.

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

If the vector is complex, then `value` consists of a pair of doubles representing the real and imaginary parts of the complex value.

See Also: `HYPRE_SStructVectorSetComplex` (→ *page 28*)

2.5.2

```
int
HYPRE_SStructVectorAddToValues (HYPRE_SStructVector vector, int part,
int *index, int var, double *value)
```

Add to vector coefficients index by index.

NOTE: For better efficiency, use `HYPRE_SStructVectorAddToBoxValues` (→2.5.4, *page 30*) to set coefficients a box at a time.

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

If the vector is complex, then `value` consists of a pair of doubles representing the real and imaginary parts of the complex value.

See Also: `HYPRE_SStructVectorSetComplex` (→ *page 28*)

2.5.3

```
int
HYPRE_SStructVectorSetBoxValues (HYPRE_SStructVector vector, int part,
int *ilower, int *iupper, int var, double *values)
```

Set vector coefficients a box at a time. The data in `values` is ordered as follows:

```

m = 0;
for (k = ilower[2]; k <= iupper[2]; k++)
  for (j = ilower[1]; j <= iupper[1]; j++)
    for (i = ilower[0]; i <= iupper[0]; i++)
      {
        values[m] = ...;
        m++;
      }

```

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

If the vector is complex, then `values` consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: `HYPRE_SStructVectorSetComplex` (*→ page 28*)

2.5.4

```

int
HYPRE_SStructVectorAddToBoxValues (HYPRE_SStructVector vector, int
part, int *ilower, int *iupper, int var, double *values)

```

Add to vector coefficients a box at a time. The data in `values` is ordered as in `HYPRE_SStructVectorSetBoxValues` (*→2.5.3, page 29*).

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

If the vector is complex, then `values` consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: `HYPRE_SStructVectorSetComplex` (*→ page 28*)

2.5.5

```

int HYPRE_SStructVectorGather (HYPRE_SStructVector vector)

```

Gather vector data so that efficient `GetValues` can be done. This routine must be called prior to calling `GetValues` to insure that correct and consistent values are returned, especially for non cell-centered data that is shared between more than one processor.

2.5.6

```
int
HYPRE_SStructVectorGetValues (HYPRE_SStructVector vector, int part, int
*index, int var, double *value)
```

Get vector coefficients index by index.

NOTE: For better efficiency, use `HYPRE_SStructVectorGetBoxValues` (→2.5.7, *page 31*) to get coefficients a box at a time.

NOTE: Users may only get values on processes that own the associated variables.

If the vector is complex, then `value` consists of a pair of doubles representing the real and imaginary parts of the complex value.

See Also: `HYPRE_SStructVectorSetComplex` (→ *page 28*)

2.5.7

```
int
HYPRE_SStructVectorGetBoxValues (HYPRE_SStructVector vector, int part,
int *ilower, int *iupper, int var, double *values)
```

Get vector coefficients a box at a time. The data in `values` is ordered as in `HYPRE_SStructVectorSetBoxValues` (→2.5.3, *page 29*).

NOTE: Users may only get values on processes that own the associated variables.

If the vector is complex, then `values` consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: `HYPRE_SStructVectorSetComplex` (→ *page 28*)

2.5.8

```
int
HYPRE_SStructVectorSetObjectType (HYPRE_SStructVector vector, int
type)
```

Set the storage type of the vector object to be constructed. Currently, `type` can be either `HYPRE_SSTRUCT` (the default), `HYPRE_STRUCT`, or `HYPRE_PARCSR`.

See Also: `HYPRE_SStructVectorGetObject` (→2.5.9, *page 32*)

2.5.9

```
int
HYPRE_SStructVectorGetObject (HYPRE_SStructVector vector, void
**object)
```

Get a reference to the constructed vector object.

See Also: `HYPRE_SStructVectorSetObjectType` (→2.5.8, *page 31*)

2.5.10

```
int
HYPRE_SStructVectorPrint (const char *filename, HYPRE_SStructVector
vector, int all)
```

Print the vector to file. This is mainly for debugging purposes.

3

extern **IJ System Interface**

Names

3.1	IJ Matrices	33
3.2	IJ Vectors	39

This interface represents a linear-algebraic conceptual view of a linear system. The 'I' and 'J' in the name are meant to be mnemonic for the traditional matrix notation A(I,J).

3.1

IJ Matrices

Names

	typedef struct hypre_IJMatrix_struct* HYPRE_IJMatrix <i>The matrix object</i>	
3.1.1	int HYPRE_IJMatrixCreate (MPI_Comm comm, int ilower, int iupper, int jlower, int jupper, HYPRE_IJMatrix *matrix) <i>Create a matrix object</i>	35
3.1.2	int HYPRE_IJMatrixDestroy (HYPRE_IJMatrix matrix) <i>Destroy a matrix object</i>	35
3.1.3	int HYPRE_IJMatrixInitialize (HYPRE_IJMatrix matrix) <i>Prepare a matrix object for setting coefficient values</i>	35
3.1.4	int HYPRE_IJMatrixSetValues (HYPRE_IJMatrix matrix, int nrows, int *ncols, const int *rows, const int *cols, const double *values) <i>Sets values for nrows rows or partial rows of the matrix</i>	36
3.1.5	int HYPRE_IJMatrixAddToValues (HYPRE_IJMatrix matrix, int nrows, int *ncols, const int *rows, const int *cols, const double *values) <i>Adds to values for nrows rows or partial rows of the matrix</i>	36
	int	

	HYPRE_IJMatrixAssemble (HYPRE_IJMatrix matrix)	
	<i>Finalize the construction of the matrix before using</i>	
	int	
	HYPRE_IJMatrixGetRowCounts (HYPRE_IJMatrix matrix, int nrows, int *rows, int *ncols)	
	<i>Gets number of nonzeros elements for nrows rows specified in rows and returns them in ncols, which needs to be allocated by the user</i>	
3.1.6	int	
	HYPRE_IJMatrixGetValues (HYPRE_IJMatrix matrix, int nrows, int *ncols, int *rows, int *cols, double *values)	
	<i>Gets values for nrows rows or partial rows of the matrix</i>	36
3.1.7	int	
	HYPRE_IJMatrixSetObjectType (HYPRE_IJMatrix matrix, int type)	
	<i>Set the storage type of the matrix object to be constructed</i>	37
	int	
	HYPRE_IJMatrixGetObjectType (HYPRE_IJMatrix matrix, int *type)	
	<i>Get the storage type of the constructed matrix object</i>	
	int	
	HYPRE_IJMatrixGetLocalRange (HYPRE_IJMatrix matrix, int *ilower, int *iupper, int *jlower, int *jupper)	
	<i>Gets range of rows owned by this processor and range of column partitioning for this processor</i>	
3.1.8	int	
	HYPRE_IJMatrixGetObject (HYPRE_IJMatrix matrix, void **object)	
	<i>Get a reference to the constructed matrix object</i>	37
3.1.9	int	
	HYPRE_IJMatrixSetRowSizes (HYPRE_IJMatrix matrix, const int *sizes)	
	<i>(Optional) Set the max number of nonzeros to expect in each row</i>	37
3.1.10	int	
	HYPRE_IJMatrixSetDiagOffdSizes (HYPRE_IJMatrix matrix, const int *diag_sizes, const int *offdiag_sizes)	
	<i>(Optional) Set the max number of nonzeros to expect in each row of the diagonal and off-diagonal blocks</i>	37
3.1.11	int	
	HYPRE_IJMatrixSetMaxOffProcElmts (HYPRE_IJMatrix matrix, int max_off_proc_elmts)	
	<i>(Optional) Sets the maximum number of elements that are expected to be set (or added) on other processors from this processor This routine can signifi- cantly improve the efficiency of matrix construction, and should always be utilized if possible</i>	38
3.1.12	int	
	HYPRE_IJMatrixRead (const char *filename, MPI_Comm comm, int type, HYPRE_IJMatrix *matrix)	
	<i>Read the matrix from file</i>	38
3.1.13	int	
	HYPRE_IJMatrixPrint (HYPRE_IJMatrix matrix, const char *filename)	
	<i>Print the matrix to file</i>	38

3.1.1

```
int  
HYPRE_IJMatrixCreate (MPI_Comm comm, int ilower, int iupper, int jlower,  
int jupper, HYPRE_IJMatrix *matrix)
```

Create a matrix object. Each process owns some unique consecutive range of rows, indicated by the global row indices `ilower` and `iupper`. The row data is required to be such that the value of `ilower` on any process p be exactly one more than the value of `iupper` on process $p - 1$. Note that the first row of the global matrix may start with any integer value. In particular, one may use zero- or one-based indexing.

For square matrices, `jlower` and `jupper` typically should match `ilower` and `iupper`, respectively. For rectangular matrices, `jlower` and `jupper` should define a partitioning of the columns. This partitioning must be used for any vector v that will be used in matrix-vector products with the rectangular matrix. The matrix data structure may use `jlower` and `jupper` to store the diagonal blocks (rectangular in general) of the matrix separately from the rest of the matrix.

Collective.

3.1.2

```
int HYPRE_IJMatrixDestroy (HYPRE_IJMatrix matrix)
```

Destroy a matrix object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

3.1.3

```
int HYPRE_IJMatrixInitialize (HYPRE_IJMatrix matrix)
```

Prepare a matrix object for setting coefficient values. This routine will also re-initialize an already assembled matrix, allowing users to modify coefficient values.

3.1.4

```
int  
HYPRE_IJMatrixSetValues (HYPRE_IJMatrix matrix, int nrows, int *ncols,  
const int *rows, const int *cols, const double *values)
```

Sets values for `nrows` rows or partial rows of the matrix. The arrays `ncols` and `rows` are of dimension `nrows` and contain the number of columns in each row and the row indices, respectively. The array `cols` contains the column indices for each of the `rows`, and is ordered by rows. The data in the `values` array corresponds directly to the column entries in `cols`. Erases any previous values at the specified locations and replaces them with new ones, or, if there was no value there before, inserts a new one.

Not collective.

3.1.5

```
int  
HYPRE_IJMatrixAddToValues (HYPRE_IJMatrix matrix, int nrows, int  
*ncols, const int *rows, const int *cols, const double *values)
```

Adds to values for `nrows` rows or partial rows of the matrix. Usage details are analogous to `HYPRE_IJMatrixSetValues` ([→3.1.4, page 36](#)). Adds to any previous values at the specified locations, or, if there was no value there before, inserts a new one.

Not collective.

3.1.6

```
int  
HYPRE_IJMatrixGetValues (HYPRE_IJMatrix matrix, int nrows, int *ncols,  
int *rows, int *cols, double *values)
```

Gets values for `nrows` rows or partial rows of the matrix. Usage details are analogous to `HYPRE_IJMatrixSetValues` ([→3.1.4, page 36](#)).

3.1.7

```
int HYPRE_IJMatrixSetObjectType (HYPRE_IJMatrix matrix, int type)
```

Set the storage type of the matrix object to be constructed. Currently, `type` can only be `HYPRE_PARCSR`.

Not collective, but must be the same on all processes.

See Also: `HYPRE_IJMatrixGetObject` ([→3.1.8, page 37](#))

3.1.8

```
int HYPRE_IJMatrixGetObject (HYPRE_IJMatrix matrix, void **object)
```

Get a reference to the constructed matrix object.

See Also: `HYPRE_IJMatrixSetObjectType` ([→3.1.7, page 37](#))

3.1.9

```
int HYPRE_IJMatrixSetRowSizes (HYPRE_IJMatrix matrix, const int *sizes)
```

(Optional) Set the max number of nonzeros to expect in each row. The array `sizes` contains estimated sizes for each row on this process. This call can significantly improve the efficiency of matrix construction, and should always be utilized if possible.

Not collective.

3.1.10

```
int
HYPRE_IJMatrixSetDiagOffdSizes (HYPRE_IJMatrix matrix, const int
*diag_sizes, const int *offdiag_sizes)
```

(Optional) Set the max number of nonzeros to expect in each row of the diagonal and off-diagonal blocks. The diagonal block is the submatrix whose column numbers correspond to rows owned by this process, and the off-diagonal block is everything else. The arrays `diag_sizes` and `offdiag_sizes` contain estimated sizes for each row of the diagonal and off-diagonal blocks, respectively. This routine can significantly improve the efficiency of matrix construction, and should always be utilized if possible.

Not collective.

3.1.11

```
int
HYPRE_IJMatrixSetMaxOffProcElmts (HYPRE_IJMatrix matrix, int
max_off_proc_elmts)
```

(Optional) Sets the maximum number of elements that are expected to be set (or added) on other processors from this processor. This routine can significantly improve the efficiency of matrix construction, and should always be utilized if possible.

Not collective.

3.1.12

```
int
HYPRE_IJMatrixRead (const char *filename, MPI_Comm comm, int type,
HYPRE_IJMatrix *matrix)
```

Read the matrix from file. This is mainly for debugging purposes.

3.1.13

```
int HYPRE_IJMatrixPrint (HYPRE_IJMatrix matrix, const char *filename)
```

Print the matrix to file. This is mainly for debugging purposes.

3.2

IJ Vectors

Names

	typedef struct hypre_IJVector_struct* HYPRE_IJVector <i>The vector object</i>	
3.2.1	int HYPRE_IJVectorCreate (MPI_Comm comm, int jlower, int jupper, HYPRE_IJVector *vector) <i>Create a vector object</i>	40
3.2.2	int HYPRE_IJVectorDestroy (HYPRE_IJVector vector) <i>Destroy a vector object</i>	40
3.2.3	int HYPRE_IJVectorInitialize (HYPRE_IJVector vector) <i>Prepare a vector object for setting coefficient values</i>	41
3.2.4	int HYPRE_IJVectorSetMaxOffProcElmts (HYPRE_IJVector vector, int max_off_proc_elmts) <i>(Optional) Sets the maximum number of elements that are expected to be set (or added) on other processors from this processor This routine can significantly improve the efficiency of matrix construction, and should always be utilized if possible</i>	41
3.2.5	int HYPRE_IJVectorSetValues (HYPRE_IJVector vector, int nvalues, const int *indices, const double *values) <i>Sets values in vector</i>	41
3.2.6	int HYPRE_IJVectorAddToValues (HYPRE_IJVector vector, int nvalues, const int *indices, const double *values) <i>Adds to values in vector</i>	41
	int HYPRE_IJVectorAssemble (HYPRE_IJVector vector) <i>Finalize the construction of the vector before using</i>	
3.2.7	int HYPRE_IJVectorGetValues (HYPRE_IJVector vector, int nvalues, const int *indices, double *values) <i>Gets values in vector</i>	42
3.2.8	int HYPRE_IJVectorSetObjectType (HYPRE_IJVector vector, int type) <i>Set the storage type of the vector object to be constructed</i>	42
	int HYPRE_IJVectorGetObjectType (HYPRE_IJVector vector, int *type) <i>Get the storage type of the constructed vector object</i>	
	int	

	HYPRE_IJVectorGetLocalRange (HYPRE_IJVector vector, int *jlower, int *jupper) <i>Returns range of the part of the vector owned by this processor</i>	
3.2.9	int HYPRE_IJVectorGetObject (HYPRE_IJVector vector, void **object) <i>Get a reference to the constructed vector object</i>	42
3.2.10	int HYPRE_IJVectorRead (const char *filename, MPI_Comm comm, int type, HYPRE_IJVector *vector) <i>Read the vector from file</i>	43
3.2.11	int HYPRE_IJVectorPrint (HYPRE_IJVector vector, const char *filename) <i>Print the vector to file</i>	43

3.2.1

```
int
HYPRE_IJVectorCreate (MPI_Comm comm, int jlower, int jupper,
HYPRE_IJVector *vector)
```

Create a vector object. Each process owns some unique consecutive range of vector unknowns, indicated by the global indices `jlower` and `jupper`. The data is required to be such that the value of `jlower` on any process p be exactly one more than the value of `jupper` on process $p - 1$. Note that the first index of the global vector may start with any integer value. In particular, one may use zero- or one-based indexing.

Collective.

3.2.2

```
int HYPRE_IJVectorDestroy (HYPRE_IJVector vector)
```

Destroy a vector object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

3.2.3

```
int HYPRE_IJVectorInitialize (HYPRE_IJVector vector)
```

Prepare a vector object for setting coefficient values. This routine will also re-initialize an already assembled vector, allowing users to modify coefficient values.

3.2.4

```
int  
HYPRE_IJVectorSetMaxOffProcElmts (HYPRE_IJVector vector, int  
max_off_proc_elmts)
```

(Optional) Sets the maximum number of elements that are expected to be set (or added) on other processors from this processor. This routine can significantly improve the efficiency of matrix construction, and should always be utilized if possible.

Not collective.

3.2.5

```
int  
HYPRE_IJVectorSetValues (HYPRE_IJVector vector, int nvalues, const int  
*indices, const double *values)
```

Sets values in vector. The arrays `values` and `indices` are of dimension `nvalues` and contain the vector values to be set and the corresponding global vector indices, respectively. Erases any previous values at the specified locations and replaces them with new ones.

Not collective.

3.2.6

```
int  
HYPRE_IJVectorAddToValues (HYPRE_IJVector vector, int nvalues, const int  
*indices, const double *values)
```

Adds to values in vector. Usage details are analogous to `HYPRE_IJVectorSetValues` ([→3.2.5, page 41](#)).

Not collective.

3.2.7

```
int
HYPRE_IJVectorGetValues (HYPRE_IJVector vector, int nvalues, const int
*indices, double *values)
```

Gets values in vector. Usage details are analogous to `HYPRE_IJVectorSetValues` ([→3.2.5, page 41](#)).

Not collective.

3.2.8

```
int HYPRE_IJVectorSetObjectType (HYPRE_IJVector vector, int type)
```

Set the storage type of the vector object to be constructed. Currently, `type` can only be `HYPRE_PARCSR`.

Not collective, but must be the same on all processes.

See Also: `HYPRE_IJVectorGetObject` ([→3.2.9, page 42](#))

3.2.9

```
int HYPRE_IJVectorGetObject (HYPRE_IJVector vector, void **object)
```

Get a reference to the constructed vector object.

See Also: `HYPRE_IJVectorSetObjectType` ([→3.2.8, page 42](#))

3.2.10

```
int  
HYPRE_IJVectorRead (const char *filename, MPI_Comm comm, int type,  
HYPRE_IJVector *vector)
```

Read the vector from file. This is mainly for debugging purposes.

3.2.11

```
int HYPRE_IJVectorPrint (HYPRE_IJVector vector, const char *filename)
```

Print the vector to file. This is mainly for debugging purposes.

4

extern **Struct Solvers**

Names

4.1	Struct Solvers	44
	
4.2	Struct Jacobi Solver	44
	
4.3	Struct PFMG Solver	47
	
4.4	Struct SMG Solver	50
	
4.5	Struct PCG Solver	52
	
4.6	Struct GMRES Solver	54
	
4.7	Struct BiCGSTAB Solver	56
	
4.8	Struct Hybrid Solver	57
	

These solvers use matrix/vector storage schemes that are tailored to structured grid problems.

4.1

Struct Solvers

Names

```
typedef struct hypre_StructSolver_struct* HYPRE_StructSolver
    The solver object
```

4.2

Struct Jacobi Solver

Names

	int	HYPRE_StructJacobiCreate (MPI_Comm comm, HYPRE_StructSolver *solver)	
		<i>Create a solver object</i>	
4.2.1	int	HYPRE_StructJacobiDestroy (HYPRE_StructSolver solver)	
		<i>Destroy a solver object</i>	46
4.2.2	int	HYPRE_StructJacobiSetup (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)	
		<i>Prepare to solve the system</i>	46
	int	HYPRE_StructJacobiSolve (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)	
		<i>Solve the system</i>	
	int	HYPRE_StructJacobiSetTol (HYPRE_StructSolver solver, double tol)	
		<i>(Optional) Set the convergence tolerance</i>	
	int	HYPRE_StructJacobiSetMaxIter (HYPRE_StructSolver solver, int max_iter)	
		<i>(Optional) Set maximum number of iterations</i>	
4.2.3	int	HYPRE_StructJacobiSetZeroGuess (HYPRE_StructSolver solver)	
		<i>(Optional) Use a zero initial guess</i>	46
4.2.4	int	HYPRE_StructJacobiSetNonZeroGuess (HYPRE_StructSolver solver)	
		<i>(Optional) Use a nonzero initial guess</i>	46
	int	HYPRE_StructJacobiGetNumIterations (HYPRE_StructSolver solver, int *num_iterations)	
		<i>Return the number of iterations taken</i>	
	int	HYPRE_StructJacobiGetFinalRelativeResidualNorm	
		(HYPRE_StructSolver solver, double *norm)	
		<i>Return the norm of the final relative residual</i>	

4.2.1

```
int HYPRE_StructJacobiDestroy (HYPRE_StructSolver solver)
```

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

4.2.2

```
int  
HYPRE_StructJacobiSetup (HYPRE_StructSolver solver, HYPRE_StructMatrix  
A, HYPRE_StructVector b, HYPRE_StructVector x)
```

Prepare to solve the system. The coefficient data in **b** and **x** is ignored here, but information about the layout of the data may be used.

4.2.3

```
int HYPRE_StructJacobiSetZeroGuess (HYPRE_StructSolver solver)
```

(Optional) Use a zero initial guess. This allows the solver to cut corners in the case where a zero initial guess is needed (e.g., for preconditioning) to reduce computational cost.

4.2.4

```
int HYPRE_StructJacobiSetNonZeroGuess (HYPRE_StructSolver solver)
```

(Optional) Use a nonzero initial guess. This is the default behavior, but this routine allows the user to switch back after using **SetZeroGuess**.

4.3

Struct PFMG Solver

Names

	int	HYPRE_StructPFMGCreate (MPI.Comm comm, HYPRE_StructSolver *solver)	
		<i>Create a solver object</i>	
	int	HYPRE_StructPFMGDestroy (HYPRE_StructSolver solver)	
		<i>Destroy a solver object</i>	
4.3.1	int	HYPRE_StructPFMGSetup (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)	
		<i>Prepare to solve the system</i>	48
	int	HYPRE_StructPFMGsolve (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)	
		<i>Solve the system</i>	
	int	HYPRE_StructPFMGSetTol (HYPRE_StructSolver solver, double tol)	
		<i>(Optional) Set the convergence tolerance</i>	
	int	HYPRE_StructPFMGSetMaxIter (HYPRE_StructSolver solver, int max_iter)	
		<i>(Optional) Set maximum number of iterations</i>	
	int	HYPRE_StructPFMGSetMaxLevels (HYPRE_StructSolver solver, int max_levels)	
		<i>(Optional) Set maximum number of multigrid grid levels</i>	
	int	HYPRE_StructPFMGSetRelChange (HYPRE_StructSolver solver, int rel_change)	
		<i>(Optional) Additionally require that the relative difference in successive it- erates be small</i>	
4.3.2	int	HYPRE_StructPFMGSetZeroGuess (HYPRE_StructSolver solver)	
		<i>(Optional) Use a zero initial guess</i>	49
4.3.3	int	HYPRE_StructPFMGSetNonZeroGuess (HYPRE_StructSolver solver)	
		<i>(Optional) Use a nonzero initial guess</i>	49
4.3.4	int		

	HYPRE_StructPFMGSetRelaxType (HYPRE_StructSolver solver, int relax_type)	
	<i>(Optional) Set relaxation type</i>	49
4.3.5	int HYPRE_StructPFMGSetRAPType (HYPRE_StructSolver solver, int rap_type)	
	<i>(Optional) Set type of coarse-grid operator to use</i>	49
	int HYPRE_StructPFMGSetNumPreRelax (HYPRE_StructSolver solver, int num_pre_relax)	
	<i>(Optional) Set number of relaxation sweeps before coarse-grid correction</i>	
	int HYPRE_StructPFMGSetNumPostRelax (HYPRE_StructSolver solver, int num_post_relax)	
	<i>(Optional) Set number of relaxation sweeps after coarse-grid correction</i>	
4.3.6	int HYPRE_StructPFMGSetSkipRelax (HYPRE_StructSolver solver, int skip_relax)	
	<i>(Optional) Skip relaxation on certain grids for isotropic problems</i>	50
	int HYPRE_StructPFMGSetLogging (HYPRE_StructSolver solver, int logging)	
	<i>(Optional) Set the amount of logging to do</i>	
	int HYPRE_StructPFMGSetPrintLevel (HYPRE_StructSolver solver, int print_level)	
	<i>(Optional) Set the amount of printing to do to the screen</i>	
	int HYPRE_StructPFMGGetNumIterations (HYPRE_StructSolver solver, int *num_iterations)	
	<i>Return the number of iterations taken</i>	
	int HYPRE_StructPFMGGetFinalRelativeResidualNorm	
	(HYPRE_StructSolver solver, double *norm)	
	<i>Return the norm of the final relative residual</i>	

4.3.1

```

int
HYPRE_StructPFMGSetup (HYPRE_StructSolver solver,
HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)

```

Prepare to solve the system. The coefficient data in **b** and **x** is ignored here, but information about the layout of the data may be used.

4.3.2

```
int HYPRE_StructPFMGSetZeroGuess (HYPRE_StructSolver solver)
```

(Optional) Use a zero initial guess. This allows the solver to cut corners in the case where a zero initial guess is needed (e.g., for preconditioning) to reduce computational cost.

4.3.3

```
int HYPRE_StructPFMGSetNonZeroGuess (HYPRE_StructSolver solver)
```

(Optional) Use a nonzero initial guess. This is the default behavior, but this routine allows the user to switch back after using `SetZeroGuess`.

4.3.4

```
int  
HYPRE_StructPFMGSetRelaxType (HYPRE_StructSolver solver, int  
relax_type)
```

(Optional) Set relaxation type.

Current relaxation methods set by `relax_type` are:

- 0 – Jacobi
- 1 – Weighted Jacobi (default)
- 2 – Red/Black Gauss-Seidel (symmetric: RB pre-relaxation, BR post-relaxation)
- 3 – Red/Black Gauss-Seidel (nonsymmetric: RB pre- and post-relaxation)

4.3.5

```
int  
HYPRE_StructPFMGSetRAPType (HYPRE_StructSolver solver, int rap_type)
```

(Optional) Set type of coarse-grid operator to use.

Current operators set by `rap_type` are:

- 0 – Galerkin (default)
- 1 – non-Galerkin 5-pt or 7-pt stencils

Both operators are constructed algebraically. The non-Galerkin option maintains a 5-pt stencil in 2D and a 7-pt stencil in 3D on all grid levels. The stencil coefficients are computed by averaging techniques.

4.3.6

```
int
HYPRE_StructPFMGSetSkipRelax (HYPRE_StructSolver solver, int
skip_relax)
```

(Optional) Skip relaxation on certain grids for isotropic problems. This can greatly improve efficiency by eliminating unnecessary relaxations when the underlying problem is isotropic.

4.4

Struct SMG Solver

Names

```
int
HYPRE_StructSMGCreate (MPI_Comm comm,
                        HYPRE_StructSolver *solver)
    Create a solver object
```

```
int
HYPRE_StructSMGDestroy (HYPRE_StructSolver solver)
    Destroy a solver object
```

```
4.4.1 int
HYPRE_StructSMGSetup (HYPRE_StructSolver solver,
                        HYPRE_StructMatrix A,
                        HYPRE_StructVector b, HYPRE_StructVector x)
    Prepare to solve the system ..... 51
```

```
int
HYPRE_StructSMGSolve (HYPRE_StructSolver solver,
                        HYPRE_StructMatrix A, HYPRE_StructVector b,
                        HYPRE_StructVector x)
    Solve the system
```

```
int
HYPRE_StructSMGSetTol (HYPRE_StructSolver solver, double tol)
    (Optional) Set the convergence tolerance
```

```
int
```

- HYPRE_StructSMGSetMaxIter** (HYPRE_StructSolver solver, int max_iter)
(Optional) Set maximum number of iterations
- int
HYPRE_StructSMGSetRelChange (HYPRE_StructSolver solver,
int rel_change)
(Optional) Additionally require that the relative difference in successive iterations be small
- 4.4.2 int
HYPRE_StructSMGSetZeroGuess (HYPRE_StructSolver solver)
(Optional) Use a zero initial guess 52
- 4.4.3 int
HYPRE_StructSMGSetNonZeroGuess (HYPRE_StructSolver solver)
(Optional) Use a nonzero initial guess 52
- int
HYPRE_StructSMGSetNumPreRelax (HYPRE_StructSolver solver,
int num_pre_relax)
(Optional) Set number of relaxation sweeps before coarse-grid correction
- int
HYPRE_StructSMGSetNumPostRelax (HYPRE_StructSolver solver,
int num_post_relax)
(Optional) Set number of relaxation sweeps after coarse-grid correction
- int
HYPRE_StructSMGSetLogging (HYPRE_StructSolver solver, int logging)
(Optional) Set the amount of logging to do
- int
HYPRE_StructSMGSetPrintLevel (HYPRE_StructSolver solver,
int print_level)
(Optional) Set the amount of printing to do to the screen
- int
HYPRE_StructSMGGetNumIterations (HYPRE_StructSolver solver,
int *num_iterations)
Return the number of iterations taken
- int
HYPRE_StructSMGGetFinalRelativeResidualNorm (HYPRE_StructSolver
solver,
double *norm)
Return the norm of the final relative residual

4.4.1

int
HYPRE_StructSMGSetup (HYPRE_StructSolver solver, HYPRE_StructMatrix
A, HYPRE_StructVector b, HYPRE_StructVector x)

Prepare to solve the system. The coefficient data in **b** and **x** is ignored here, but information about the layout of the data may be used.

4.4.2

```
int HYPRE_StructSMGSetZeroGuess (HYPRE_StructSolver solver)
```

(Optional) Use a zero initial guess. This allows the solver to cut corners in the case where a zero initial guess is needed (e.g., for preconditioning) to reduce computational cost.

4.4.3

```
int HYPRE_StructSMGSetNonZeroGuess (HYPRE_StructSolver solver)
```

(Optional) Use a nonzero initial guess. This is the default behavior, but this routine allows the user to switch back after using **SetZeroGuess**.

4.5

Struct PCG Solver

Names

```
int
HYPRE_StructPCGCreate (MPI_Comm comm,
                      HYPRE_StructSolver *solver)
```

Create a solver object

```
int
HYPRE_StructPCGDestroy (HYPRE_StructSolver solver)
```

Destroy a solver object

```
4.5.1 int
HYPRE_StructPCGSetup (HYPRE_StructSolver solver,
                     HYPRE_StructMatrix A,
                     HYPRE_StructVector b, HYPRE_StructVector x)
```

Prepare to solve the system 54

```
int
HYPRE_StructPCGSolve (HYPRE_StructSolver solver,
                     HYPRE_StructMatrix A, HYPRE_StructVector b,
                     HYPRE_StructVector x)
```

Solve the system

```
int
```

HYPRE_StructPCGSetTol (HYPRE_StructSolver solver, double tol)
(Optional) Set the convergence tolerance

int

HYPRE_StructPCGSetMaxIter (HYPRE_StructSolver solver, int max_iter)
(Optional) Set maximum number of iterations

int

HYPRE_StructPCGSetTwoNorm (HYPRE_StructSolver solver,
int two_norm)
(Optional) Use the two-norm in stopping criteria

int

HYPRE_StructPCGSetRelChange (HYPRE_StructSolver solver,
int rel_change)
(Optional) Additionally require that the relative difference in successive iterates be small

int

HYPRE_StructPCGSetPrecond (HYPRE_StructSolver solver,
HYPRE_PtrToStructSolverFcn precond,
HYPRE_PtrToStructSolverFcn
precond_setup,
HYPRE_StructSolver precond_solver)
(Optional) Set the preconditioner to use

int

HYPRE_StructPCGSetLogging (HYPRE_StructSolver solver, int logging)
(Optional) Set the amount of logging to do

int

HYPRE_StructPCGSetPrintLevel (HYPRE_StructSolver solver, int level)
(Optional) Set the amount of printing to do to the screen

int

HYPRE_StructPCGGetNumIterations (HYPRE_StructSolver solver,
int *num_iterations)
Return the number of iterations taken

int

HYPRE_StructPCGGetFinalRelativeResidualNorm (HYPRE_StructSolver
solver,
double *norm)
Return the norm of the final relative residual

int

HYPRE_StructPCGGetResidual (HYPRE_StructSolver solver,
void **residual)
Return the residual

int

HYPRE_StructDiagScaleSetup (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector y,
HYPRE_StructVector x)
Setup routine for diagonal preconditioning

int

HYPRE_StructDiagScale (HYPRE_StructSolver solver,
 HYPRE_StructMatrix HA,
 HYPRE_StructVector Hy,
 HYPRE_StructVector Hx)

Solve routine for diagonal preconditioning

4.5.1

```
int
HYPRE_StructPCGSetup (HYPRE_StructSolver solver, HYPRE_StructMatrix
A, HYPRE_StructVector b, HYPRE_StructVector x)
```

Prepare to solve the system. The coefficient data in **b** and **x** is ignored here, but information about the layout of the data may be used.

4.6

Struct GMRES Solver

Names

```
int
HYPRE_StructGMRESCreate ( MPI_Comm comm,
                          HYPRE_StructSolver *solver )
```

Create a solver object

```
int
HYPRE_StructGMRESDestroy ( HYPRE_StructSolver solver )
```

Destroy a solver object

4.6.1

```
int
HYPRE_StructGMRESSetup ( HYPRE_StructSolver solver,
                          HYPRE_StructMatrix A,
                          HYPRE_StructVector b,
                          HYPRE_StructVector x )
```

Prepare to solve the system 55

```
int
HYPRE_StructGMRESSolve ( HYPRE_StructSolver solver,
                          HYPRE_StructMatrix A,
                          HYPRE_StructVector b,
                          HYPRE_StructVector x )
```

Solve the system

```
int
```

HYPRE_StructGMRESSetTol (HYPRE_StructSolver solver, double tol)
(Optional) Set the convergence tolerance

int
HYPRE_StructGMRESSetMaxIter (HYPRE_StructSolver solver,
int max_iter)
(Optional) Set maximum number of iterations

int
HYPRE_StructGMRESSetPrecond (HYPRE_StructSolver solver,
HYPRE_PtrToStructSolverFcn precond,
HYPRE_PtrToStructSolverFcn
precond_setup,
HYPRE_StructSolver precond_solver)
(Optional) Set the preconditioner to use

int
HYPRE_StructGMRESSetLogging (HYPRE_StructSolver solver,
int logging)
(Optional) Set the amount of logging to do

int
HYPRE_StructGMRESSetPrintLevel (HYPRE_StructSolver solver,
int level)
(Optional) Set the amount of printing to do to the screen

int
HYPRE_StructGMRESGetNumIterations (HYPRE_StructSolver solver,
int *num_iterations)
Return the number of iterations taken

int
HYPRE_StructGMRESGetFinalRelativeResidualNorm (
HYPRE_StructSolver
solver,
double *norm)
Return the norm of the final relative residual

int
HYPRE_StructGMRESGetResidual (HYPRE_StructSolver solver,
void **residual)
Return the residual

4.6.1

int
HYPRE_StructGMRESSetup (HYPRE_StructSolver solver,
HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)

Prepare to solve the system. The coefficient data in **b** and **x** is ignored here, but information about the layout of the data may be used.

HYPRE_StructBiCGSTABSetPrintLevel (HYPRE_StructSolver solver,
int level)
(Optional) Set the amount of printing to do to the screen

int
HYPRE_StructBiCGSTABGetNumIterations (HYPRE_StructSolver
solver, int *num.iterations)
Return the number of iterations taken

int
HYPRE_StructBiCGSTABGetFinalRelativeResidualNorm (
HYPRE_StructSolver
solver,
double *norm
)
Return the norm of the final relative residual

int
HYPRE_StructBiCGSTABGetResidual (HYPRE_StructSolver solver,
void **residual)
Return the residual

4.7.1

int
HYPRE_StructBiCGSTABSetup (HYPRE_StructSolver solver,
HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)

Prepare to solve the system. The coefficient data in **b** and **x** is ignored here, but information about the layout of the data may be used.

4.8

Struct Hybrid Solver

Names

int
HYPRE_StructHybridCreate (MPI_Comm comm,
HYPRE_StructSolver *solver)
Create a solver object

int
HYPRE_StructHybridDestroy (HYPRE_StructSolver solver)
Destroy a solver object

4.8.1 int

-
- HYPRE_StructHybridSetup** (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x)
Prepare to solve the system 59
- int
HYPRE_StructHybridSolve (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x)
Solve the system
- int
HYPRE_StructHybridSetTol (HYPRE_StructSolver solver, double tol)
(Optional) Set the convergence tolerance
- 4.8.2 int
HYPRE_StructHybridSetConvergenceTol (HYPRE_StructSolver solver,
double cf_tol)
(Optional) Set an accepted convergence tolerance for diagonal scaling (DS)
60
- 4.8.3 int
HYPRE_StructHybridSetDSCGMaxIter (HYPRE_StructSolver solver,
int ds_max_its)
(Optional) Set maximum number of iterations for diagonal scaling (DS) . 60
- 4.8.4 int
HYPRE_StructHybridSetPCGMaxIter (HYPRE_StructSolver solver,
int pre_max_its)
(Optional) Set maximum number of iterations for general preconditioner (PRE) 60
- int
HYPRE_StructHybridSetTwoNorm (HYPRE_StructSolver solver,
int two_norm)
(Optional) Use the two-norm in stopping criteria
- int
HYPRE_StructHybridSetRelChange (HYPRE_StructSolver solver,
int rel_change)
(Optional) Additionally require that the relative difference in successive iterates be small
- 4.8.5 int
HYPRE_StructHybridSetSolverType (HYPRE_StructSolver solver,
int solver_type)
(Optional) Set the type of Krylov solver to use 60
- int
HYPRE_StructHybridSetKDim (HYPRE_StructSolver solver, int k_dim)
(Optional) Set the maximum size of the Krylov space when using GMRES
- int

HYPRE_StructHybridSetPrecond (HYPRE_StructSolver solver,
HYPRE_PtrToStructSolverFcn precond,
HYPRE_PtrToStructSolverFcn
precond_setup,
HYPRE_StructSolver precond_solver)
(Optional) Set the preconditioner to use

int
HYPRE_StructHybridSetLogging (HYPRE_StructSolver solver, int logging)
(Optional) Set the amount of logging to do

int
HYPRE_StructHybridSetPrintLevel (HYPRE_StructSolver solver,
int print_level)
(Optional) Set the amount of printing to do to the screen

int
HYPRE_StructHybridGetNumIterations (HYPRE_StructSolver solver,
int *num_its)
Return the number of iterations taken

int
HYPRE_StructHybridGetDSCGNumIterations (HYPRE_StructSolver
solver, int *ds_num_its)
Return the number of diagonal scaling iterations taken

int
HYPRE_StructHybridGetPCGNumIterations (HYPRE_StructSolver
solver, int *pre_num_its)
Return the number of general preconditioning iterations taken

int
HYPRE_StructHybridGetFinalRelativeResidualNorm
(HYPRE_StructSolver
solver,
double *norm)
Return the norm of the final relative residual

4.8.1

int
HYPRE_StructHybridSetup (HYPRE_StructSolver solver,
HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)

Prepare to solve the system. The coefficient data in **b** and **x** is ignored here, but information about the layout of the data may be used.

4.8.2

```
int
HYPRE_StructHybridSetConvergenceTol (HYPRE_StructSolver solver,
double cf_tol)
```

(Optional) Set an accepted convergence tolerance for diagonal scaling (DS). The solver will switch preconditioners if the convergence of DS is slower than `cf_tol`.

4.8.3

```
int
HYPRE_StructHybridSetDSCGMaxIter (HYPRE_StructSolver solver, int
ds_max_its)
```

(Optional) Set maximum number of iterations for diagonal scaling (DS). The solver will switch preconditioners if DS reaches `ds_max_its`.

4.8.4

```
int
HYPRE_StructHybridSetPCGMaxIter (HYPRE_StructSolver solver, int
pre_max_its)
```

(Optional) Set maximum number of iterations for general preconditioner (PRE). The solver will stop if PRE reaches `pre_max_its`.

4.8.5

```
int
HYPRE_StructHybridSetSolverType (HYPRE_StructSolver solver, int
solver_type)
```

(Optional) Set the type of Krylov solver to use.

Current krylov methods set by `solver_type` are:

- 0 – PCG (default)
- 1 – GMRES
- 2 – BiCGSTAB

5

extern **SStruct Solvers****Names**

5.1	SStruct Solvers	62
	
5.2	SStruct PCG Solver	62
	
5.3	SStruct GMRES Solver	65
	
5.4	SStruct BiCGSTAB Solver	67
	
5.5	SStruct SysPFMG Solver	69
	
5.6	SStruct Split Solver	72
	
5.7	SStruct FAC Solver	74
	
5.8	SStruct Maxwell Solver	79
	

These solvers use matrix/vector storage schemes that are tailored to semi-structured grid problems.

5.1

SStruct Solvers**Names**

```
typedef struct hypre_SStructSolver_struct* HYPRE_SStructSolver
    The solver object
```

5.2

SStruct PCG Solver**Names**

	int	HYPRE_SStructPCGCreate (MPI_Comm comm, HYPRE_SStructSolver *solver)	
		<i>Create a solver object</i>	
5.2.1	int	HYPRE_SStructPCGDestroy (HYPRE_SStructSolver solver)	
		<i>Destroy a solver object</i>	64
5.2.2	int	HYPRE_SStructPCGSetup (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)	
		<i>Prepare to solve the system</i>	64
	int	HYPRE_SStructPCGSolve (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)	
		<i>Solve the system</i>	
	int	HYPRE_SStructPCGSetTol (HYPRE_SStructSolver solver, double tol)	
		<i>(Optional) Set the convergence tolerance</i>	
	int	HYPRE_SStructPCGSetMaxIter (HYPRE_SStructSolver solver, int max_iter)	
		<i>(Optional) Set maximum number of iterations</i>	
	int	HYPRE_SStructPCGSetTwoNorm (HYPRE_SStructSolver solver, int two_norm)	
		<i>(Optional) Use the two-norm in stopping criteria</i>	
	int	HYPRE_SStructPCGSetRelChange (HYPRE_SStructSolver solver, int rel_change)	
		<i>(Optional) Additionally require that the relative difference in successive iterates be small</i>	
	int	HYPRE_SStructPCGSetPrecond (HYPRE_SStructSolver solver, HYPRE_PtrToSStructSolverFcn precondition, HYPRE_PtrToSStructSolverFcn precond_setup, void *precond_solver)	
		<i>(Optional) Set the preconditioner to use</i>	
	int	HYPRE_SStructPCGSetLogging (HYPRE_SStructSolver solver, int logging)	
		<i>(Optional) Set the amount of logging to do</i>	
	int	HYPRE_SStructPCGSetPrintLevel (HYPRE_SStructSolver solver, int level)	
		<i>(Optional) Set the amount of printing to do to the screen</i>	
	int		

HYPRE_SStructPCGGetNumIterations (HYPRE_SStructSolver solver,
int *num_iterations)

Return the number of iterations taken

int

HYPRE_SStructPCGGetFinalRelativeResidualNorm (HYPRE_SStructSolver
solver,
double *norm)

Return the norm of the final relative residual

int

HYPRE_SStructPCGGetResidual (HYPRE_SStructSolver solver,
void **residual)

Return the residual

int

HYPRE_SStructDiagScaleSetup (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector y,
HYPRE_SStructVector x)

Setup routine for diagonal preconditioning

int

HYPRE_SStructDiagScale (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector y,
HYPRE_SStructVector x)

Solve routine for diagonal preconditioning

5.2.1

```
int HYPRE_SStructPCGDestroy (HYPRE_SStructSolver solver)
```

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

5.2.2

```
int
HYPRE_SStructPCGSetup (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)
```


Prepare to solve the system. The coefficient data in **b** and **x** is ignored here, but information about the layout of the data may be used.

5.3

SStruct GMRES Solver

Names

	int	HYPRE_SStructGMRESCreate (MPI_Comm comm, HYPRE_SStructSolver *solver)	
		<i>Create a solver object</i>	
5.3.1	int	HYPRE_SStructGMRESDestroy (HYPRE_SStructSolver solver)	
		<i>Destroy a solver object</i>	66
5.3.2	int	HYPRE_SStructGMRESSetup (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)	
		<i>Prepare to solve the system</i>	66
	int	HYPRE_SStructGMRESSolve (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)	
		<i>Solve the system</i>	
	int	HYPRE_SStructGMRESSTol (HYPRE_SStructSolver solver, double tol)	
		<i>(Optional) Set the convergence tolerance</i>	
	int	HYPRE_SStructGMRESSTMaxIter (HYPRE_SStructSolver solver, int max_iter)	
		<i>(Optional) Set maximum number of iterations</i>	
	int	HYPRE_SStructGMRESSTKDim (HYPRE_SStructSolver solver, int k_dim)	
		<i>(Optional) Set the maximum size of the Krylov space</i>	
	int	HYPRE_SStructGMRESSTPrecond (HYPRE_SStructSolver solver, HYPRE_PtrToSStructSolverFcn precondition, HYPRE_PtrToSStructSolverFcn precondition_setup, void *precond_solver)	
		<i>(Optional) Set the preconditioner to use</i>	
	int		

HYPRE_SStructGMRESSetLogging (HYPRE_SStructSolver solver,
int logging)
(Optional) Set the amount of logging to do

int
HYPRE_SStructGMRESSetPrintLevel (HYPRE_SStructSolver solver,
int print_level)
(Optional) Set the amount of printing to do to the screen

int
HYPRE_SStructGMRESGetNumIterations (HYPRE_SStructSolver solver,
int *num_iterations)
Return the number of iterations taken

int
HYPRE_SStructGMRESGetFinalRelativeResidualNorm
(HYPRE_SStructSolver solver,
double *norm)
Return the norm of the final relative residual

int
HYPRE_SStructGMRESGetResidual (HYPRE_SStructSolver solver,
void **residual)
Return the residual

5.3.1

int **HYPRE_SStructGMRESDestroy** (HYPRE_SStructSolver solver)

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

5.3.2

int
HYPRE_SStructGMRESSetup (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)

Prepare to solve the system. The coefficient data in **b** and **x** is ignored here, but information about the layout of the data may be used.

SStruct BiCGSTAB Solver

Names

	int	HYPRE_SStructBiCGSTABCreate (MPI_Comm comm, HYPRE_SStructSolver *solver)	
		<i>Create a solver object</i>	
5.4.1	int	HYPRE_SStructBiCGSTABDestroy (HYPRE_SStructSolver solver)	
		<i>Destroy a solver object</i>	68
5.4.2	int	HYPRE_SStructBiCGSTABSetup (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)	
		<i>Prepare to solve the system</i>	68
	int	HYPRE_SStructBiCGSTABSolve (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)	
		<i>Solve the system</i>	
	int	HYPRE_SStructBiCGSTABSetTol (HYPRE_SStructSolver solver, double tol)	
		<i>(Optional) Set the convergence tolerance</i>	
	int	HYPRE_SStructBiCGSTABSetMaxIter (HYPRE_SStructSolver solver, int max_iter)	
		<i>(Optional) Set maximum number of iterations</i>	
	int	HYPRE_SStructBiCGSTABSetPrecond (HYPRE_SStructSolver solver, HYPRE_PtrToSStructSolverFcn precond, HYPRE_PtrToSStructSolverFcn precond_setup, void *precond_solver)	
		<i>(Optional) Set the preconditioner to use</i>	
	int	HYPRE_SStructBiCGSTABSetLogging (HYPRE_SStructSolver solver, int logging)	
		<i>(Optional) Set the amount of logging to do</i>	
	int		

HYPRE_SStructBiCGSTABSetPrintLevel (HYPRE_SStructSolver solver,
int level)
(Optional) Set the amount of printing to do to the screen

int
HYPRE_SStructBiCGSTABGetNumIterations (HYPRE_SStructSolver
solver,
int *num_iterations)
Return the number of iterations taken

int
HYPRE_SStructBiCGSTABGetFinalRelativeResidualNorm
(HYPRE_SStructSolver
solver,
double
*norm)
Return the norm of the final relative residual

int
HYPRE_SStructBiCGSTABGetResidual (HYPRE_SStructSolver solver,
void **residual)
Return the residual

5.4.1

int **HYPRE_SStructBiCGSTABDestroy** (HYPRE_SStructSolver solver)

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

5.4.2

int
HYPRE_SStructBiCGSTABSetup (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)

Prepare to solve the system. The coefficient data in **b** and **x** is ignored here, but information about the layout of the data may be used.

5.5

SStruct SysPFMG Solver

Names

	int	HYPRE_SStructSysPFMGCreate (MPI_Comm comm, HYPRE_SStructSolver *solver)	
		<i>Create a solver object</i>	
5.5.1	int	HYPRE_SStructSysPFMGDestroy (HYPRE_SStructSolver solver)	
		<i>Destroy a solver object</i>	70
5.5.2	int	HYPRE_SStructSysPFMGSetup (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)	
		<i>Prepare to solve the system</i>	71
	int	HYPRE_SStructSysPFMGsSolve (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)	
		<i>Solve the system</i>	
	int	HYPRE_SStructSysPFMGSetTol (HYPRE_SStructSolver solver, double tol)	
		<i>(Optional) Set the convergence tolerance</i>	
	int	HYPRE_SStructSysPFMGSetMaxIter (HYPRE_SStructSolver solver, int max_iter)	
		<i>(Optional) Set maximum number of iterations</i>	
	int	HYPRE_SStructSysPFMGSetRelChange (HYPRE_SStructSolver solver, int rel_change)	
		<i>(Optional) Additionally require that the relative difference in successive it- erates be small</i>	
5.5.3	int	HYPRE_SStructSysPFMGSetZeroGuess (HYPRE_SStructSolver solver)	
		<i>(Optional) Use a zero initial guess</i>	71
5.5.4	int	HYPRE_SStructSysPFMGSetNonZeroGuess (HYPRE_SStructSolver solver)	
		<i>(Optional) Use a nonzero initial guess</i>	71
5.5.5	int	HYPRE_SStructSysPFMGSetRelaxType (HYPRE_SStructSolver solver, int relax_type)	
		<i>(Optional) Set relaxation type</i>	71
	int		

- HYPRE_SStructSysPFMGSetNumPreRelax** (HYPRE_SStructSolver solver,
int num_pre_relax)
(Optional) Set number of relaxation sweeps before coarse-grid correction
- int
HYPRE_SStructSysPFMGSetNumPostRelax (HYPRE_SStructSolver
solver, int num_post_relax)
(Optional) Set number of relaxation sweeps after coarse-grid correction
- 5.5.6 int
HYPRE_SStructSysPFMGSetSkipRelax (HYPRE_SStructSolver solver,
int skip_relax)
(Optional) Skip relaxation on certain grids for isotropic problems 72
- int
HYPRE_SStructSysPFMGSetLogging (HYPRE_SStructSolver solver,
int logging)
(Optional) Set the amount of logging to do
- int
HYPRE_SStructSysPFMGSetPrintLevel (HYPRE_SStructSolver solver,
int print_level)
(Optional) Set the amount of printing to do to the screen
- int
HYPRE_SStructSysPFMGGetNumIterations (HYPRE_SStructSolver
solver, int *num_iterations)
Return the number of iterations taken
- int
HYPRE_SStructSysPFMGGetFinalRelativeResidualNorm (
HYPRE_SStructSolver
solver,
double
*norm)
Return the norm of the final relative residual

5.5.1

```
int HYPRE_SStructSysPFMGDestroy (HYPRE_SStructSolver solver)
```

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

5.5.2

```
int
HYPRE_SStructSysPFMGSetup (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)
```

Prepare to solve the system. The coefficient data in `b` and `x` is ignored here, but information about the layout of the data may be used.

5.5.3

```
int HYPRE_SStructSysPFMGSetZeroGuess (HYPRE_SStructSolver solver)
```

(Optional) Use a zero initial guess. This allows the solver to cut corners in the case where a zero initial guess is needed (e.g., for preconditioning) to reduce computational cost.

5.5.4

```
int
HYPRE_SStructSysPFMGSetNonZeroGuess (HYPRE_SStructSolver solver)
```

(Optional) Use a nonzero initial guess. This is the default behavior, but this routine allows the user to switch back after using `SetZeroGuess`.

5.5.5

```
int
HYPRE_SStructSysPFMGSetRelaxType (HYPRE_SStructSolver solver, int
relax_type)
```

(Optional) Set relaxation type.

Current relaxation methods set by `relax_type` are:

- 0 – Jacobi
- 1 – Weighted Jacobi (default)
- 2 – Red/Black Gauss-Seidel (symmetric: RB pre-relaxation, BR post-relaxation)

5.5.6

```
int
HYPRE_SStructSysPFMGSetSkipRelax (HYPRE_SStructSolver solver, int
skip_relax)
```

(Optional) Skip relaxation on certain grids for isotropic problems. This can greatly improve efficiency by eliminating unnecessary relaxations when the underlying problem is isotropic.

5.6

SStruct Split Solver

Names

- ```
int
HYPRE_SStructSplitCreate (MPI_Comm comm,
 HYPRE_SStructSolver *solver)
 Create a solver object
```
- 5.6.1
- ```
int
HYPRE_SStructSplitDestroy (HYPRE_SStructSolver solver)
    Destroy a solver object ..... 73
```
- 5.6.2
- ```
int
HYPRE_SStructSplitSetup (HYPRE_SStructSolver solver,
 HYPRE_SStructMatrix A,
 HYPRE_SStructVector b,
 HYPRE_SStructVector x)
 Prepare to solve the system 73
```
- ```
int
HYPRE_SStructSplitSolve (HYPRE_SStructSolver solver,
                           HYPRE_SStructMatrix A,
                           HYPRE_SStructVector b,
                           HYPRE_SStructVector x)
    Solve the system
```
- ```
int
HYPRE_SStructSplitSetTol (HYPRE_SStructSolver solver, double tol)
 (Optional) Set the convergence tolerance
```
- ```
int
```

	HYPRE_SStructSplitSetMaxIter (HYPRE_SStructSolver solver, int max_iter) <i>(Optional) Set maximum number of iterations</i>	
5.6.3	int HYPRE_SStructSplitSetZeroGuess (HYPRE_SStructSolver solver) <i>(Optional) Use a zero initial guess</i>	74
5.6.4	int HYPRE_SStructSplitSetNonZeroGuess (HYPRE_SStructSolver solver) <i>(Optional) Use a nonzero initial guess</i>	74
5.6.5	int HYPRE_SStructSplitSetStructSolver (HYPRE_SStructSolver solver, int ssolver) <i>(Optional) Set up the type of diagonal struct solver</i>	74
	int HYPRE_SStructSplitGetNumIterations (HYPRE_SStructSolver solver, int *num_iterations) <i>Return the number of iterations taken</i>	
	int HYPRE_SStructSplitGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, double *norm) <i>Return the norm of the final relative residual</i>	

5.6.1

int **HYPRE_SStructSplitDestroy** (HYPRE_SStructSolver solver)

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

5.6.2

int
HYPRE_SStructSplitSetup (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)

Prepare to solve the system. The coefficient data in **b** and **x** is ignored here, but information about the layout of the data may be used.

5.6.3

```
int HYPRE_SStructSplitSetZeroGuess (HYPRE_SStructSolver solver)
```

(Optional) Use a zero initial guess. This allows the solver to cut corners in the case where a zero initial guess is needed (e.g., for preconditioning) to reduce computational cost.

5.6.4

```
int HYPRE_SStructSplitSetNonZeroGuess (HYPRE_SStructSolver solver)
```

(Optional) Use a nonzero initial guess. This is the default behavior, but this routine allows the user to switch back after using `SetZeroGuess`.

5.6.5

```
int
HYPRE_SStructSplitSetStructSolver (HYPRE_SStructSolver solver, int
ssolver )
```

(Optional) Set up the type of diagonal struct solver. Either `ssolver` is set to `HYPRE_SMG` or `HYPRE_PFMG`.

5.7

SStruct FAC Solver

Names

	int	HYPRE_SStructFACCreate (MPI_Comm comm, HYPRE_SStructSolver *solver) <i>Create a solver object</i>	
5.7.1	int	HYPRE_SStructFACDestroy2 (HYPRE_SStructSolver solver) <i>Destroy a solver object</i>	77
5.7.2	int		

-
- HYPRE_SStructFACAMR_RAP** (HYPRE_SStructMatrix A,
int (*rfactors)[3],
HYPRE_SStructMatrix *fac_A)
*Re-distribute the composite matrix so that the amr hierachy is approximately
nested* 77
- int
HYPRE_SStructFACSetup2 (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector b,
HYPRE_SStructVector x)
Set up the FAC solver structure
- int
HYPRE_SStructFACSolve3 (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector b,
HYPRE_SStructVector x)
Solve the system
- int
HYPRE_SStructFACSetPLevels (HYPRE_SStructSolver solver, int nparts,
int *plevels)
Set up amr structure
- int
HYPRE_SStructFACSetPRefinements (HYPRE_SStructSolver solver,
int nparts, int (*rfactors)[3])
Set up amr refinement factors
- 5.7.3 int
HYPRE_SStructFACZeroCFSten (HYPRE_SStructMatrix A,
HYPRE_SStructGrid grid, int part,
int rfactors[3])
(Optional, but user must make sure that they do this function otherwise . 77
- 5.7.4 int
HYPRE_SStructFACZeroFCSten (HYPRE_SStructMatrix A,
HYPRE_SStructGrid grid, int part)
(Optional, but user must make sure that they do this function otherwise . 77
- 5.7.5 int
HYPRE_SStructFACZeroAMRMatrixData (HYPRE_SStructMatrix A,
int part_crse, int rfactors[3])
(Optional, but user must make sure that they do this function otherwise . 78
- 5.7.6 int
HYPRE_SStructFACZeroAMRVectorData (HYPRE_SStructVector b,
int *plevels, int (*rfactors)[3])
(Optional, but user must make sure that they do this function otherwise . 78
- int
HYPRE_SStructFACSetMaxLevels (HYPRE_SStructSolver solver,
int max_levels)
(Optional) Set maximum number of FAC levels
- int

	HYPRE_SStructFACSetTol (HYPRE_SStructSolver solver, double tol) (Optional) Set the convergence tolerance	
	int HYPRE_SStructFACSetMaxIter (HYPRE_SStructSolver solver, int max_iter) (Optional) Set maximum number of iterations	
	int HYPRE_SStructFACSetRelChange (HYPRE_SStructSolver solver, int rel_change) (Optional) Additionally require that the relative difference in successive it- erates be small	
5.7.7	int HYPRE_SStructFACSetZeroGuess (HYPRE_SStructSolver solver) (Optional) Use a zero initial guess	78
5.7.8	int HYPRE_SStructFACSetNonZeroGuess (HYPRE_SStructSolver solver) (Optional) Use a nonzero initial guess	78
5.7.9	int HYPRE_SStructFACSetRelaxType (HYPRE_SStructSolver solver, int relax_type) (Optional) Set relaxation type	79
	int HYPRE_SStructFACSetNumPreRelax (HYPRE_SStructSolver solver, int num_pre_relax) (Optional) Set number of relaxation sweeps before coarse-grid correction	
	int HYPRE_SStructFACSetNumPostRelax (HYPRE_SStructSolver solver, int num_post_relax) (Optional) Set number of relaxation sweeps after coarse-grid correction	
5.7.10	int HYPRE_SStructFACSetCoarseSolverType (HYPRE_SStructSolver solver, int csolver_type) (Optional) Set coarsest solver type	79
	int HYPRE_SStructFACSetLogging (HYPRE_SStructSolver solver, int logging) (Optional) Set the amount of logging to do	
	int HYPRE_SStructFACGetNumIterations (HYPRE_SStructSolver solver, int *num_iterations) Return the number of iterations taken	
	int HYPRE_SStructFACGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, double *norm) Return the norm of the final relative residual	

5.7.1

```
int HYPRE_SStructFACDestroy2 ( HYPRE_SStructSolver solver )
```

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

5.7.2

```
int
HYPRE_SStructFACAMR_RAP ( HYPRE_SStructMatrix A, int (*rfactors)[3],
HYPRE_SStructMatrix *fac_A )
```

Re-distribute the composite matrix so that the amr hierarchy is approximately nested. Coarse underlying operators are also formed.

5.7.3

```
int
HYPRE_SStructFACZeroCFSten (HYPRE_SStructMatrix A,
HYPRE_SStructGrid grid, int part, int rfactors[3])
```

(Optional, but user must make sure that they do this function otherwise.) Zero off the coarse level stencils reaching into a fine level grid.

5.7.4

```
int
HYPRE_SStructFACZeroFCSten (HYPRE_SStructMatrix A,
HYPRE_SStructGrid grid, int part)
```

(Optional, but user must make sure that they do this function otherwise.) Zero off the fine level stencils reaching into a coarse level grid.

5.7.5

```
int
HYPRE_SStructFACZeroAMRMatrixData (HYPRE_SStructMatrix A, int
part_crse, int rfactors[3])
```

(Optional, but user must make sure that they do this function otherwise.) Places the identity in the coarse grid matrix underlying the fine patches. Required between each pair of amr levels.

5.7.6

```
int
HYPRE_SStructFACZeroAMRVectorData (HYPRE_SStructVector b, int
*plevels, int (*rfactors)[3] )
```

(Optional, but user must make sure that they do this function otherwise.) Places zeros in the coarse grid vector underlying the fine patches. Required between each pair of amr levels.

5.7.7

```
int HYPRE_SStructFACSetZeroGuess (HYPRE_SStructSolver solver)
```

(Optional) Use a zero initial guess. This allows the solver to cut corners in the case where a zero initial guess is needed (e.g., for preconditioning) to reduce computational cost.

5.7.8

```
int HYPRE_SStructFACSetNonZeroGuess (HYPRE_SStructSolver solver)
```

(Optional) Use a nonzero initial guess. This is the default behavior, but this routine allows the user to switch back after using `SetZeroGuess`.

5.7.9

```
int
HYPRE_SStructFACSetRelaxType (HYPRE_SStructSolver solver, int
relax_type)
```

(Optional) Set relaxation type. See `HYPRE_SStructSysPFMGSetRelaxType` (→5.5.5, *page 71*) for appropriate values of `relax_type`.

5.7.10

```
int
HYPRE_SStructFACSetCoarseSolverType (HYPRE_SStructSolver solver, int
csolver_type)
```

(Optional) Set coarsest solver type.

Current solver types set by `csolver_type` are:

- 1 – SysPFMG-PCG (default)
- 2 – SysPFMG

5.8

SStruct Maxwell Solver

Names

- ```
int
HYPRE_SStructMaxwellCreate (MPI.Comm comm,
 HYPRE_SStructSolver *solver)
 Create a solver object
```
- 5.8.1    int  
**HYPRE\_SStructMaxwellDestroy** ( HYPRE\_SStructSolver solver )  
*Destroy a solver object* ..... 81
- 5.8.2    int  
**HYPRE\_SStructMaxwellSetup** (HYPRE\_SStructSolver solver,  
 HYPRE\_SStructMatrix A,  
 HYPRE\_SStructVector b,  
 HYPRE\_SStructVector x)  
*Prepare to solve the system* ..... 82
- 5.8.3    int

- 
- HYPRE\_SStructMaxwellSolve** (HYPRE\_SStructSolver solver,  
HYPRE\_SStructMatrix A,  
HYPRE\_SStructVector b,  
HYPRE\_SStructVector x)  
*Solve the system* ..... 82
- 5.8.4 int  
**HYPRE\_SStructMaxwellSolve2** (HYPRE\_SStructSolver solver,  
HYPRE\_SStructMatrix A,  
HYPRE\_SStructVector b,  
HYPRE\_SStructVector x)  
*Solve the system* ..... 82
- int  
**HYPRE\_SStructMaxwellSetGrad** (HYPRE\_SStructSolver solver,  
HYPRE\_ParCSRMatrix T)  
*Sets the gradient operator in the Maxwell solver*
- int  
**HYPRE\_SStructMaxwellSetRfactors** (HYPRE\_SStructSolver solver,  
int rfactors[3])  
*Sets the coarsening factor*
- int  
**HYPRE\_SStructMaxwellPhysBdy** (HYPRE\_SStructGrid \*grid\_l,  
int num\_levels, int rfactors[3],  
int \*\*\*BdryRanks\_ptr,  
int \*\*BdryRanksCnt\_ptr )  
*Finds the physical boundary row ranks on all levels*
- int  
**HYPRE\_SStructMaxwellEliminateRowsCols** (HYPRE\_ParCSRMatrix  
parA, int nrows, int \*rows )  
*Eliminates the rows and cols corresponding to the physical boundary in a  
parcsr matrix*
- int  
**HYPRE\_SStructMaxwellZeroVector** (HYPRE\_ParVector b, int \*rows,  
int nrows )  
*Zeros the rows corresponding to the physical boundary in a par vector*
- int  
**HYPRE\_SStructMaxwellSetSetConstantCoef** (HYPRE\_SStructSolver  
solver, int flag)  
*(Optional) Set the constant coefficient flag- Nedelec interpolation used*
- 5.8.5 int  
**HYPRE\_SStructMaxwellGrad** (HYPRE\_SStructGrid grid,  
HYPRE\_ParCSRMatrix \*T)  
*(Optional) Creates a gradient matrix from the grid* ..... 82
- int  
**HYPRE\_SStructMaxwellSetTol** (HYPRE\_SStructSolver solver, double tol)  
*(Optional) Set the convergence tolerance*
- int



---

**HYPRE\_SStructMaxwellSetMaxIter** (HYPRE\_SStructSolver solver,  
int max\_iter)  
*(Optional) Set maximum number of iterations*

int  
**HYPRE\_SStructMaxwellSetRelChange** (HYPRE\_SStructSolver solver,  
int rel\_change)  
*(Optional) Additionally require that the relative difference in successive iterations be small*

int  
**HYPRE\_SStructMaxwellSetNumPreRelax** (HYPRE\_SStructSolver solver,  
int num\_pre\_relax)  
*(Optional) Set number of relaxation sweeps before coarse-grid correction*

int  
**HYPRE\_SStructMaxwellSetNumPostRelax** (HYPRE\_SStructSolver solver,  
int num\_post\_relax)  
*(Optional) Set number of relaxation sweeps after coarse-grid correction*

int  
**HYPRE\_SStructMaxwellSetLogging** (HYPRE\_SStructSolver solver,  
int logging)  
*(Optional) Set the amount of logging to do*

int  
**HYPRE\_SStructMaxwellGetNumIterations** (HYPRE\_SStructSolver solver,  
int \*num\_iterations)  
*Return the number of iterations taken*

int  
**HYPRE\_SStructMaxwellGetFinalRelativeResidualNorm** (HYPRE\_SStructSolver solver,  
double \*norm)  
*Return the norm of the final relative residual*

### 5.8.1

```
int HYPRE_SStructMaxwellDestroy (HYPRE_SStructSolver solver)
```

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

## 5.8.2

```
int
HYPRE_SStructMaxwellSetup (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)
```

Prepare to solve the system. The coefficient data in **b** and **x** is ignored here, but information about the layout of the data may be used.

## 5.8.3

```
int
HYPRE_SStructMaxwellSolve (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)
```

Solve the system. Full coupling of the augmented system used throughout the multigrid hierarchy.

## 5.8.4

```
int
HYPRE_SStructMaxwellSolve2 (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)
```

Solve the system. Full coupling of the augmented system used only on the finest level, i.e., the node and edge multigrid cycles are coupled only on the finest level.

## 5.8.5

```
int
HYPRE_SStructMaxwellGrad (HYPRE_SStructGrid grid,
HYPRE_ParCSRMatrix *T)
```

(Optional) Creates a gradient matrix from the grid. This presupposes a particular orientation of the edge elements.

6

extern **ParCSR Solvers**

## Names

|      |                                                   |     |
|------|---------------------------------------------------|-----|
| 6.1  | <b>ParCSR Solvers</b>                             | 83  |
|      | .....                                             |     |
| 6.2  | <b>ParCSR BoomerAMG Solver and Preconditioner</b> | 84  |
|      | .....                                             |     |
| 6.3  | <b>ParCSR ParaSails Preconditioner</b>            | 103 |
|      | .....                                             |     |
| 6.4  | <b>ParCSR Euclid Preconditioner</b>               | 107 |
|      | .....                                             |     |
| 6.5  | <b>ParCSR Pilut Preconditioner</b>                | 109 |
|      | .....                                             |     |
| 6.6  | <b>ParCSR AMS Solver and Preconditioner</b>       | 110 |
|      | .....                                             |     |
| 6.7  | <b>ParCSR Hybrid Solver</b>                       | 117 |
|      | .....                                             |     |
| 6.8  | <b>ParCSR PCG Solver</b>                          | 128 |
|      | .....                                             |     |
| 6.9  | <b>ParCSR GMRES Solver</b>                        | 130 |
|      | .....                                             |     |
| 6.10 | <b>ParCSR BiCGSTAB Solver</b>                     | 131 |
|      | .....                                             |     |

These solvers use matrix/vector storage schemes that are tailored for general sparse matrix systems.

6.1

**ParCSR Solvers**

## Names

```
#define HYPRE_SOLVER_STRUCT
 The solver object
```

## 6.2

## ParCSR BoomerAMG Solver and Preconditioner

## Names

|       |     |                                                                                                                                                                                                                                                 |    |
|-------|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
|       | int | <b>HYPRE_BoomerAMGCreate</b> (HYPRE_Solver *solver)<br><i>Create a solver object</i>                                                                                                                                                            |    |
|       | int | <b>HYPRE_BoomerAMGDestroy</b> (HYPRE_Solver solver)<br><i>Destroy a solver object</i>                                                                                                                                                           |    |
| 6.2.1 | int | <b>HYPRE_BoomerAMGSetup</b> (HYPRE_Solver solver,<br>HYPRE_ParCSRMatrix A,<br>HYPRE_ParVector b, HYPRE_ParVector x)<br><i>Set up the BoomerAMG solver or preconditioner</i> .....                                                               | 89 |
| 6.2.2 | int | <b>HYPRE_BoomerAMGSolve</b> (HYPRE_Solver solver,<br>HYPRE_ParCSRMatrix A,<br>HYPRE_ParVector b, HYPRE_ParVector x)<br><i>Solve the system or apply AMG as a preconditioner</i> .....                                                           | 89 |
| 6.2.3 | int | <b>HYPRE_BoomerAMGSolveT</b> (HYPRE_Solver solver,<br>HYPRE_ParCSRMatrix A,<br>HYPRE_ParVector b, HYPRE_ParVector x)<br><i>Solve the transpose system <math>A^T x = b</math> or apply AMG as a preconditioner to the transpose system</i> ..... | 89 |
| 6.2.4 | int | <b>HYPRE_BoomerAMGSetTol</b> (HYPRE_Solver solver, double tol)<br><i>(Optional) Set the convergence tolerance, if BoomerAMG is used as a solver</i> .....                                                                                       | 90 |
| 6.2.5 | int | <b>HYPRE_BoomerAMGSetMaxIter</b> (HYPRE_Solver solver, int max_iter)<br><i>(Optional) Sets maximum number of iterations, if BoomerAMG is used as a solver</i> .....                                                                             | 90 |
| 6.2.6 | int | <b>HYPRE_BoomerAMGSetMaxLevels</b> (HYPRE_Solver solver, int max_levels)<br><i>(Optional) Sets maximum number of multigrid levels</i> .....                                                                                                     | 90 |
| 6.2.7 | int | <b>HYPRE_BoomerAMGSetStrongThreshold</b> (HYPRE_Solver solver,<br>double strong_threshold)<br><i>(Optional) Sets AMG strength threshold</i> .....                                                                                               | 90 |
| 6.2.8 | int | <b>HYPRE_BoomerAMGSetMaxRowSum</b> (HYPRE_Solver solver,<br>double max_row_sum)<br><i>(Optional) Sets a parameter to modify the definition of strength for diagonal dominant portions of the matrix</i> .....                                   | 91 |
| 6.2.9 | int |                                                                                                                                                                                                                                                 |    |

|        |                                                                                                                                                                                                               |    |
|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
|        | <b>HYPRE_BoomerAMGSetCoarsenType</b> (HYPRE_Solver solver,<br>int coarsen_type)<br><i>(Optional) Defines which parallel coarsening algorithm is used</i> .....                                                | 91 |
|        | int<br><b>HYPRE_BoomerAMGSetMeasureType</b> (HYPRE_Solver solver,<br>int measure_type)<br><i>(Optional) Defines whether local or global measures are used</i>                                                 |    |
| 6.2.10 | int<br><b>HYPRE_BoomerAMGSetCycleType</b> (HYPRE_Solver solver, int cycle_type)<br><i>(Optional) Defines the type of cycle</i> .....                                                                          | 92 |
| 6.2.11 | int<br><b>HYPRE_BoomerAMGSetNumGridSweeps</b> (HYPRE_Solver solver,<br>int *num_grid_sweeps)<br><i>(Optional) Defines the number of sweeps for the fine and coarse grid, the<br/>up and down cycle</i> .....  | 92 |
| 6.2.12 | int<br><b>HYPRE_BoomerAMGSetNumSweeps</b> (HYPRE_Solver solver,<br>int num_sweeps)<br><i>(Optional) Sets the number of sweeps</i> .....                                                                       | 92 |
| 6.2.13 | int<br><b>HYPRE_BoomerAMGSetCycleNumSweeps</b> (HYPRE_Solver solver,<br>int num_sweeps, int k)<br><i>(Optional) Sets the number of sweeps at a specified cycle</i> .....                                      | 92 |
| 6.2.14 | int<br><b>HYPRE_BoomerAMGSetGridRelaxType</b> (HYPRE_Solver solver,<br>int *grid_relax_type)<br><i>(Optional) Defines which smoother is used on the fine and coarse grid, the<br/>up and down cycle</i> ..... | 93 |
| 6.2.15 | int<br><b>HYPRE_BoomerAMGSetRelaxType</b> (HYPRE_Solver solver, int relax_type)<br><i>(Optional) Defines the smoother to be used</i> .....                                                                    | 93 |
| 6.2.16 | int<br><b>HYPRE_BoomerAMGSetCycleRelaxType</b> (HYPRE_Solver solver,<br>int relax_type, int k)<br><i>(Optional) Defines the smoother at a given cycle</i> .....                                               | 93 |
| 6.2.17 | int<br><b>HYPRE_BoomerAMGSetRelaxOrder</b> (HYPRE_Solver solver,<br>int relax_order)<br><i>(Optional) Defines in which order the points are relaxed</i> .....                                                 | 94 |
| 6.2.18 | int<br><b>HYPRE_BoomerAMGSetGridRelaxPoints</b> (HYPRE_Solver solver,<br>int **grid_relax_points)<br><i>(Optional) Defines in which order the points are relaxed</i> .....                                    | 94 |
| 6.2.19 | int<br><b>HYPRE_BoomerAMGSetRelaxWeight</b> (HYPRE_Solver solver,<br>double *relax_weight)<br><i>(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid<br/>SOR</i> .....                    | 94 |
| 6.2.20 | int                                                                                                                                                                                                           |    |

---

|        |     |                                                                                                                                                                                                                          |    |
|--------|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
|        |     | <b>HYPRE_BoomerAMGSetRelaxWt</b> (HYPRE_Solver solver,<br>double relax_weight)<br><i>(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid<br/>SOR on all levels</i> .....                             | 95 |
| 6.2.21 | int | <b>HYPRE_BoomerAMGSetLevelRelaxWt</b> (HYPRE_Solver solver,<br>double relax_weight, int level)<br><i>(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid<br/>SOR on the user defined level</i> ..... | 95 |
| 6.2.22 | int | <b>HYPRE_BoomerAMGSetOmega</b> (HYPRE_Solver solver, double *omega)<br><i>(Optional) Defines the outer relaxation weight for hybrid SOR</i> .....                                                                        | 95 |
| 6.2.23 | int | <b>HYPRE_BoomerAMGSetOuterWt</b> (HYPRE_Solver solver, double omega)<br><i>(Optional) Defines the outer relaxation weight for hybrid SOR and SSOR<br/>on all levels</i> .....                                            | 96 |
| 6.2.24 | int | <b>HYPRE_BoomerAMGSetLevelOuterWt</b> (HYPRE_Solver solver,<br>double omega, int level)<br><i>(Optional) Defines the outer relaxation weight for hybrid SOR or SSOR on<br/>the user defined level</i> .....              | 96 |
|        | int | <b>HYPRE_BoomerAMGSetDebugFlag</b> (HYPRE_Solver solver, int debug_flag)<br><i>(Optional)</i>                                                                                                                            |    |
|        | int | <b>HYPRE_BoomerAMGGetResidual</b> (HYPRE_Solver solver,<br>HYPRE_ParVector * residual)<br><i>Returns the residual</i>                                                                                                    |    |
|        | int | <b>HYPRE_BoomerAMGGetNumIterations</b> (HYPRE_Solver solver,<br>int *num_iterations)<br><i>Returns the number of iterations taken</i>                                                                                    |    |
|        | int | <b>HYPRE_BoomerAMGGetFinalRelativeResidualNorm</b> (HYPRE_Solver<br>solver, double<br>*rel_resid_norm)<br><i>Returns the norm of the final relative residual</i>                                                         |    |
| 6.2.25 | int | <b>HYPRE_BoomerAMGSetTruncFactor</b> (HYPRE_Solver solver,<br>double trunc_factor)<br><i>(Optional) Defines a truncation factor for the interpolation</i> .....                                                          | 96 |
| 6.2.26 | int | <b>HYPRE_BoomerAMGSetPMaxElmts</b> (HYPRE_Solver solver,<br>int P_max_elmts)<br><i>(Optional) Defines the maximal number of elements per row for the inter-<br/>polation</i> .....                                       | 96 |
| 6.2.27 | int |                                                                                                                                                                                                                          |    |

|        |                                                                                                                                                                                                                                              |     |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
|        | <b>HYPRE_BoomerAMGSetSCommPkgSwitch</b> (HYPRE_Solver solver,<br>double S_commpkg_switch)<br><i>(Optional) Defines the largest strength threshold for which the strength matrix S uses the communication package of the operator A</i> ..... | 97  |
| 6.2.28 | int<br><b>HYPRE_BoomerAMGSetSmoothType</b> (HYPRE_Solver solver,<br>int smooth_type)<br><i>(Optional) Enables the use of more complex smoothers</i> .....                                                                                    | 97  |
| 6.2.29 | int<br><b>HYPRE_BoomerAMGSetSmoothNumLevels</b> (HYPRE_Solver solver,<br>int smooth_num_levels)<br><i>(Optional) Sets the number of levels for more complex smoothers</i> .....                                                              | 97  |
| 6.2.30 | int<br><b>HYPRE_BoomerAMGSetSmoothNumSweeps</b> (HYPRE_Solver solver,<br>int smooth_num_sweeps)<br><i>(Optional) Sets the number of sweeps for more complex smoothers</i> .....                                                              | 98  |
| 6.2.31 | int<br><b>HYPRE_BoomerAMGSetPrintLevel</b> (HYPRE_Solver solver, int print_level)<br><i>(Optional) Requests automatic printing of setup and solve information</i> ...                                                                        | 98  |
| 6.2.32 | int<br><b>HYPRE_BoomerAMGSetLogging</b> (HYPRE_Solver solver, int logging)<br><i>(Optional) Requests additional computations for diagnostic and similar data to be logged by the user</i> .....                                              | 98  |
| 6.2.33 | int<br><b>HYPRE_BoomerAMGSetNumFunctions</b> (HYPRE_Solver solver,<br>int num_functions)<br><i>(Optional) Sets the size of the system of PDEs, if using the systems version</i> .....                                                        | 99  |
| 6.2.34 | int<br><b>HYPRE_BoomerAMGSetNodal</b> (HYPRE_Solver solver, int nodal)<br><i>(Optional) Sets whether to use the nodal systems version</i> .....                                                                                              | 99  |
| 6.2.35 | int<br><b>HYPRE_BoomerAMGSetDofFunc</b> (HYPRE_Solver solver, int *dof_func)<br><i>(Optional) Sets the mapping that assigns the function to each variable, if using the systems version</i> .....                                            | 99  |
| 6.2.36 | int<br><b>HYPRE_BoomerAMGSetAggNumLevels</b> (HYPRE_Solver solver,<br>int agg_num_levels)<br><i>(Optional) Defines the number of levels of aggressive coarsening</i> .....                                                                   | 99  |
| 6.2.37 | int<br><b>HYPRE_BoomerAMGSetNumPaths</b> (HYPRE_Solver solver, int num_paths)<br><i>(Optional) Defines the degree of aggressive coarsening</i> .....                                                                                         | 100 |
| 6.2.38 | int<br><b>HYPRE_BoomerAMGSetVariant</b> (HYPRE_Solver solver, int variant)<br><i>(Optional) Defines which variant of the Schwarz method is used</i> .....                                                                                    | 100 |
| 6.2.39 | int                                                                                                                                                                                                                                          |     |

|        |                                                                                                                                                                                         |     |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
|        | <b>HYPRE_BoomerAMGSetOverlap</b> (HYPRE_Solver solver, int overlap)<br><i>(Optional) Defines the overlap for the Schwarz method</i> .....                                               | 100 |
| 6.2.40 | int<br><b>HYPRE_BoomerAMGSetDomainType</b> (HYPRE_Solver solver,<br>int domain_type)<br><i>(Optional) Defines the type of domain used for the Schwarz method</i> .....                  | 100 |
|        | int<br><b>HYPRE_BoomerAMGSetSchwarzRlxWeight</b> (HYPRE_Solver solver,<br>double schwarz_rlx_weight)<br><i>(Optional) Defines a smoothing parameter for the additive Schwarz method</i> |     |
| 6.2.41 | int<br><b>HYPRE_BoomerAMGSetSym</b> (HYPRE_Solver solver, int sym)<br><i>(Optional) Defines symmetry for ParaSAILS</i> .....                                                            | 101 |
| 6.2.42 | int<br><b>HYPRE_BoomerAMGSetLevel</b> (HYPRE_Solver solver, int level)<br><i>(Optional) Defines number of levels for ParaSAILS</i> .....                                                | 101 |
| 6.2.43 | int<br><b>HYPRE_BoomerAMGSetThreshold</b> (HYPRE_Solver solver,<br>double threshold)<br><i>(Optional) Defines threshold for ParaSAILS</i> .....                                         | 101 |
| 6.2.44 | int<br><b>HYPRE_BoomerAMGSetFilter</b> (HYPRE_Solver solver, double filter)<br><i>(Optional) Defines filter for ParaSAILS</i> .....                                                     | 101 |
| 6.2.45 | int<br><b>HYPRE_BoomerAMGSetDropTol</b> (HYPRE_Solver solver, double drop_tol)<br><i>(Optional) Defines drop tolerance for PILUT</i> .....                                              | 102 |
| 6.2.46 | int<br><b>HYPRE_BoomerAMGSetMaxNzPerRow</b> (HYPRE_Solver solver,<br>int max_nz_per_row)<br><i>(Optional) Defines maximal number of nonzeros for PILUT</i> .....                        | 102 |
| 6.2.47 | int<br><b>HYPRE_BoomerAMGSetEuclidFile</b> (HYPRE_Solver solver, char *euclidfile)<br><i>(Optional) Defines name of an input file for Euclid parameters</i> .....                       | 102 |
| 6.2.48 | int<br><b>HYPRE_BoomerAMGSetGSMG</b> (HYPRE_Solver solver, int gsmg)<br><i>(Optional) Specifies the use of GSMG - geometrically smooth coarsening and interpolation</i> .....           | 102 |
|        | int<br><b>HYPRE_BoomerAMGSetNumSamples</b> (HYPRE_Solver solver,<br>int num_samples)<br><i>(Optional) Defines the number of sample vectors used in GSMG or LS interpolation</i>         |     |

Parallel unstructured algebraic multigrid solver and preconditioner



## 6.2.1

```
int
HYPRE_BoomerAMGSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)
```

Set up the BoomerAMG solver or preconditioner. If used as a preconditioner, this function should be passed to the iterative solver `SetPrecond` function.

**Parameters:**

- `solver` — [IN] object to be set up.
- `A` — [IN] ParCSR matrix used to construct the solver/preconditioner.
- `b` — Ignored by this function.
- `x` — Ignored by this function.

## 6.2.2

```
int
HYPRE_BoomerAMGSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)
```

Solve the system or apply AMG as a preconditioner. If used as a preconditioner, this function should be passed to the iterative solver `SetPrecond` function.

**Parameters:**

- `solver` — [IN] solver or preconditioner object to be applied.
- `A` — [IN] ParCSR matrix, matrix of the linear system to be solved
- `b` — [IN] right hand side of the linear system to be solved
- `x` — [OUT] approximated solution of the linear system to be solved

## 6.2.3

```
int
HYPRE_BoomerAMGSolveT (HYPRE_Solver solver, HYPRE_ParCSRMatrix
A, HYPRE_ParVector b, HYPRE_ParVector x)
```

Solve the transpose system  $A^T x = b$  or apply AMG as a preconditioner to the transpose system . If used as a preconditioner, this function should be passed to the iterative solver `SetPrecond` function.

**Parameters:**

- `solver` — [IN] solver or preconditioner object to be applied.
- `A` — [IN] ParCSR matrix
- `b` — [IN] right hand side of the linear system to be solved
- `x` — [OUT] approximated solution of the linear system to be solved

#### 6.2.4

```
int HYPRE_BoomerAMGSetTol (HYPRE_Solver solver, double tol)
```

(Optional) Set the convergence tolerance, if BoomerAMG is used as a solver. If it is used as a preconditioner, this function has no effect. The default is 1.e-7.

#### 6.2.5

```
int HYPRE_BoomerAMGSetMaxIter (HYPRE_Solver solver, int max_iter)
```

(Optional) Sets maximum number of iterations, if BoomerAMG is used as a solver. If it is used as a preconditioner, this function has no effect. The default is 20.

#### 6.2.6

```
int
HYPRE_BoomerAMGSetMaxLevels (HYPRE_Solver solver, int max_levels)
```

(Optional) Sets maximum number of multigrid levels. The default is 25.

#### 6.2.7

```
int
HYPRE_BoomerAMGSetStrongThreshold (HYPRE_Solver solver, double
strong_threshold)
```

(Optional) Sets AMG strength threshold. The default is 0.25. For 2d Laplace operators, 0.25 is a good value, for 3d Laplace operators, 0.5 or 0.6 is a better value. For elasticity problems, a large strength threshold, such as 0.9, is often better.

### 6.2.8

```
int
HYPRE_BoomerAMGSetMaxRowSum (HYPRE_Solver solver, double
max_row_sum)
```

(Optional) Sets a parameter to modify the definition of strength for diagonal dominant portions of the matrix. The default is 0.9. If max\_row\_sum is 1, no checking for diagonally dominant rows is performed.

### 6.2.9

```
int
HYPRE_BoomerAMGSetCoarsenType (HYPRE_Solver solver, int
coarsen_type)
```

(Optional) Defines which parallel coarsening algorithm is used. There are the following options for coarsen\_type:

|    |                                                                                                                                                                   |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | CLJP-coarsening (a parallel coarsening algorithm using independent sets.                                                                                          |
| 1  | classical Ruge-Stueben coarsening on each processor, no boundary treatment (not recommended!)                                                                     |
| 3  | classical Ruge-Stueben coarsening on each processor, followed by a third pass, which adds coarse points on the boundaries                                         |
| 6  | Falgout coarsening (uses 1 first, followed by CLJP using the interior coarse points generated by 1 as its first independent set)                                  |
| 7  | CLJP-coarsening (using a fixed random vector, for debugging purposes only)                                                                                        |
| 8  | PMIS-coarsening (a parallel coarsening algorithm using independent sets, generating lower complexities than CLJP, might also lead to slower convergence)          |
| 9  | PMIS-coarsening (using a fixed random vector, for debugging purposes only)                                                                                        |
| 10 | HMIS-coarsening (uses one pass Ruge-Stueben on each processor independently, followed by PMIS using the interior C-points generated as its first independent set) |
| 11 | one-pass Ruge-Stueben coarsening on each processor, no boundary treatment (not recommended!)                                                                      |

The default is 6.

**6.2.10**

```
int
HYPRE_BoomerAMGSetCycleType (HYPRE_Solver solver, int cycle_type)
```

(Optional) Defines the type of cycle. For a V-cycle, set `cycle_type` to 1, for a W-cycle set `cycle_type` to 2. The default is 1.

**6.2.11**

```
int
HYPRE_BoomerAMGSetNumGridSweeps (HYPRE_Solver solver, int
*num_grid_sweeps)
```

(Optional) Defines the number of sweeps for the fine and coarse grid, the up and down cycle.

Note: This routine will be phased out!!!! Use `HYPRE_BoomerAMGSetNumSweeps` or `HYPRE_BoomerAMGSetCycleNumSweeps` instead.

**6.2.12**

```
int
HYPRE_BoomerAMGSetNumSweeps (HYPRE_Solver solver, int num_sweeps)
```

(Optional) Sets the number of sweeps. On the finest level, the up and the down cycle the number of sweeps are set to `num_sweeps` and on the coarsest level to 1. The default is 1.

**6.2.13**

```
int
HYPRE_BoomerAMGSetCycleNumSweeps (HYPRE_Solver solver, int
num_sweeps, int k)
```

(Optional) Sets the number of sweeps at a specified cycle. There are the following options for `k`:

|                    |         |
|--------------------|---------|
| the finest level   | if k=0  |
| the down cycle     | if k=1  |
| the up cycle       | if k=2  |
| the coarsest level | if k=3. |

**6.2.14**

```
int
HYPRE_BoomerAMGSetGridRelaxType (HYPRE_Solver solver, int
*grid_relax_type)
```

(Optional) Defines which smoother is used on the fine and coarse grid, the up and down cycle.

Note: This routine will be phased out!!!! Use `HYPRE_BoomerAMGSetRelaxType` or `HYPRE_BoomerAMGSetCycleRelaxType` instead.

**6.2.15**

```
int
HYPRE_BoomerAMGSetRelaxType (HYPRE_Solver solver, int relax_type)
```

(Optional) Defines the smoother to be used. It uses the given smoother on the fine grid, the up and the down cycle and sets the solver on the coarsest level to Gaussian elimination (9). The default is Gauss-Seidel (3).

There are the following options for `relax_type`:

|   |                                                                        |
|---|------------------------------------------------------------------------|
| 0 | Jacobi                                                                 |
| 1 | Gauss-Seidel, sequential (very slow!)                                  |
| 2 | Gauss-Seidel, interior points in parallel, boundary sequential (slow!) |
| 3 | hybrid Gauss-Seidel or SOR, forward solve                              |
| 4 | hybrid Gauss-Seidel or SOR, backward solve                             |
| 5 | hybrid chaotic Gauss-Seidel (works only with OpenMP)                   |
| 6 | hybrid symmetric Gauss-Seidel or SSOR                                  |
| 9 | Gaussian elimination (only on coarsest level)                          |

**6.2.16**

```
int
HYPRE_BoomerAMGSetCycleRelaxType (HYPRE_Solver solver, int
relax_type, int k)
```

(Optional) Defines the smoother at a given cycle. For options of `relax_type` see description of `HYPRE_BoomerAMGSetRelaxType`). Options for `k` are

|                    |                       |
|--------------------|-----------------------|
| the finest level   | if <code>k=0</code>   |
| the down cycle     | if <code>k=1</code>   |
| the up cycle       | if <code>k=2</code>   |
| the coarsest level | if <code>k=3</code> . |

### 6.2.17

```
int
HYPRE_BoomerAMGSetRelaxOrder (HYPRE_Solver solver, int relax_order)
```

(Optional) Defines in which order the points are relaxed. There are the following options for `relax_order`:

|   |                                                                                                                                                                                                                                                                                                                  |
|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0 | the points are relaxed in natural or lexicographic order on each processor                                                                                                                                                                                                                                       |
| 1 | CF-relaxation is used, i.e on the fine grid and the down cycle the coarse points are relaxed first, followed by the fine points; on the up cycle the F-points are relaxed first, followed by the C-points. On the coarsest level, if an iterative scheme is used, the points are relaxed in lexicographic order. |

The default is 1 (CF-relaxation).

### 6.2.18

```
int
HYPRE_BoomerAMGSetGridRelaxPoints (HYPRE_Solver solver, int
**grid_relax_points)
```

(Optional) Defines in which order the points are relaxed.

Note: This routine will be phased out!!!! Use `HYPRE_BoomerAMGSetRelaxOrder` instead.

### 6.2.19

```
int
HYPRE_BoomerAMGSetRelaxWeight (HYPRE_Solver solver, double
*relax_weight)
```

(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR.

Note: This routine will be phased out!!!! Use `HYPRE_BoomerAMGSetRelaxWt` or `HYPRE_BoomerAMGSetLevelRelaxWt` instead.

### 6.2.20

```
int
HYPRE_BoomerAMGSetRelaxWt (HYPRE_Solver solver, double
relax_weight)
```

(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR on all levels.

|                                  |                                                                                                                                                                              |
|----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>relax_weight &gt; 0</code> | this assigns the given relaxation weight on all levels                                                                                                                       |
| <code>relax_weight = 0</code>    | the weight is determined on each level with the estimate $\frac{3}{4\ D^{-1/2}AD^{-1/2}\ }$ , where $D$ is the diagonal matrix of $A$ (this should only be used with Jacobi) |
| <code>relax_weight = -k</code>   | the relaxation weight is determined with at most $k$ CG steps on each level this should only be used for symmetric positive definite problems)                               |

The default is 1.

### 6.2.21

```
int
HYPRE_BoomerAMGSetLevelRelaxWt (HYPRE_Solver solver, double
relax_weight, int level)
```

(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR on the user defined level. Note that the finest level is denoted 0, the next coarser level 1, etc. For nonpositive `relax_weight`, the parameter is determined on the given level as described for `HYPRE_BoomerAMGSetRelaxWt`. The default is 1.

### 6.2.22

```
int HYPRE_BoomerAMGSetOmega (HYPRE_Solver solver, double *omega)
```

(Optional) Defines the outer relaxation weight for hybrid SOR. Note: This routine will be phased out!!!! Use `HYPRE_BoomerAMGSetOuterWt` or `HYPRE_BoomerAMGSetLevelOuterWt` instead.

**6.2.23**

```
int HYPRE_BoomerAMGSetOuterWt (HYPRE_Solver solver, double omega)
```

(Optional) Defines the outer relaxation weight for hybrid SOR and SSOR on all levels.

|            |                                                                                                                                                                             |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| omega > 0  | this assigns the same outer relaxation weight omega on each level                                                                                                           |
| omega = -k | an outer relaxation weight is determined with at most k CG steps on each level<br>(this only makes sense for symmetric positive definite problems and smoothers, e.g. SSOR) |

The default is 1.

**6.2.24**

```
int
HYPRE_BoomerAMGSetLevelOuterWt (HYPRE_Solver solver, double
omega, int level)
```

(Optional) Defines the outer relaxation weight for hybrid SOR or SSOR on the user defined level. Note that the finest level is denoted 0, the next coarser level 1, etc. For nonpositive omega, the parameter is determined on the given level as described for `HYPRE_BoomerAMGSetOuterWt`. The default is 1.

**6.2.25**

```
int
HYPRE_BoomerAMGSetTruncFactor (HYPRE_Solver solver, double
trunc_factor)
```

(Optional) Defines a truncation factor for the interpolation. The default is 0.

**6.2.26**

```
int
HYPRE_BoomerAMGSetPMaxElmts (HYPRE_Solver solver, int
P_max_elmts)
```



(Optional) Defines the maximal number of elements per row for the interpolation. The default is 0.

### 6.2.27

```
int
HYPRE_BoomerAMGSetSCommPkgSwitch (HYPRE_Solver solver, double
S_commpkg_switch)
```

(Optional) Defines the largest strength threshold for which the strength matrix S uses the communication package of the operator A. If the strength threshold is larger than this values, a communication package is generated for S. This can save memory and decrease the amount of data that needs to be communicated, if S is substantially sparser than A. The default is 1.0.

### 6.2.28

```
int
HYPRE_BoomerAMGSetSmoothType (HYPRE_Solver solver, int
smooth_type)
```

(Optional) Enables the use of more complex smoothers. The following options exist for smooth\_type:

| value | smoother          | routines needed to set smoother parameters                                                                             |
|-------|-------------------|------------------------------------------------------------------------------------------------------------------------|
| 6     | Schwarz smoothers | HYPRE_BoomerAMGSetDomainType, HYPRE_BoomerAMGSetOverlap, HYPRE_BoomerAMGSetVariant, HYPRE_BoomerAMGSetSchwarzRlxWeight |
| 7     | Pilut             | HYPRE_BoomerAMGSetDropTol, HYPRE_BoomerAMGSetMaxNzPerRow                                                               |
| 8     | ParaSails         | HYPRE_BoomerAMGSetSym, HYPRE_BoomerAMGSetLevel, HYPRE_BoomerAMGSetFilter, HYPRE_BoomerAMGSetThreshold                  |
| 9     | Euclid            | HYPRE_BoomerAMGSetEuclidFile                                                                                           |

The default is 6. Also, if no smoother parameters are set via the routines mentioned in the table above, default values are used.

### 6.2.29

```
int
HYPRE_BoomerAMGSetSmoothNumLevels (HYPRE_Solver solver, int
smooth_num_levels)
```

(Optional) Sets the number of levels for more complex smoothers. The smoothers, as defined by `HYPRE_BoomerAMGSetSmoothType`, will be used on level 0 (the finest level) through level `smooth_num_levels-1`. The default is 0, i.e. no complex smoothers are used.

### 6.2.30

```
int
HYPRE_BoomerAMGSetSmoothNumSweeps (HYPRE_Solver solver, int
smooth_num_sweeps)
```

(Optional) Sets the number of sweeps for more complex smoothers. The default is 1.

### 6.2.31

```
int
HYPRE_BoomerAMGSetPrintLevel (HYPRE_Solver solver, int print_level)
```

(Optional) Requests automatic printing of setup and solve information.

|   |                                        |
|---|----------------------------------------|
| 0 | no printout (default)                  |
| 1 | print setup information                |
| 2 | print solve information                |
| 3 | print both setup and solve information |

Note, that if one desires to print information and uses

BoomerAMG as a preconditioner, suggested `print_level` is 1 to avoid excessive output, and use `print_level` of solver for solve phase information.

### 6.2.32

```
int HYPRE_BoomerAMGSetLogging (HYPRE_Solver solver, int logging)
```

(Optional) Requests additional computations for diagnostic and similar data to be logged by the user. Default to 0 for do nothing. The latest residual will be available if `logging > 1`.

**6.2.33**

```
int
HYPRE_BoomerAMGSetNumFunctions (HYPRE_Solver solver, int
num_functions)
```

(Optional) Sets the size of the system of PDEs, if using the systems version. The default is 1.

**6.2.34**

```
int HYPRE_BoomerAMGSetNodal (HYPRE_Solver solver, int nodal)
```

(Optional) Sets whether to use the nodal systems version. The default is 0.

**6.2.35**

```
int HYPRE_BoomerAMGSetDofFunc (HYPRE_Solver solver, int *dof_func)
```

(Optional) Sets the mapping that assigns the function to each variable, if using the systems version. If no assignment is made and the number of functions is  $k > 1$ , the mapping generated is  $(0,1,\dots,k-1,0,1,\dots,k-1,\dots)$ .

**6.2.36**

```
int
HYPRE_BoomerAMGSetAggNumLevels (HYPRE_Solver solver, int
agg_num_levels)
```

(Optional) Defines the number of levels of aggressive coarsening. The default is 0, i.e. no aggressive coarsening.

**6.2.37**

```
int
HYPRE_BoomerAMGSetNumPaths (HYPRE_Solver solver, int num_paths)
```

(Optional) Defines the degree of aggressive coarsening. The default is 1.

**6.2.38**

```
int HYPRE_BoomerAMGSetVariant (HYPRE_Solver solver, int variant)
```

(Optional) Defines which variant of the Schwarz method is used. The following options exist for variant:

|   |                                                                                 |
|---|---------------------------------------------------------------------------------|
| 0 | hybrid multiplicative Schwarz method (no overlap across processor boundaries)   |
| 1 | hybrid additive Schwarz method (no overlap across processor boundaries)         |
| 2 | additive Schwarz method                                                         |
| 3 | hybrid multiplicative Schwarz method (with overlap across processor boundaries) |

The default is 0.

**6.2.39**

```
int HYPRE_BoomerAMGSetOverlap (HYPRE_Solver solver, int overlap)
```

(Optional) Defines the overlap for the Schwarz method. The following options exist for overlap:

|   |                                                                   |
|---|-------------------------------------------------------------------|
| 0 | no overlap                                                        |
| 1 | minimal overlap (default)                                         |
| 2 | overlap generated by including all neighbors of domain boundaries |

**6.2.40**

```
int
HYPRE_BoomerAMGSetDomainType (HYPRE_Solver solver, int
domain_type)
```

(Optional) Defines the type of domain used for the Schwarz method. The following options exist for `domain_type`:

|   |                                                           |
|---|-----------------------------------------------------------|
| 0 | each point is a domain                                    |
| 1 | each node is a domain (only of interest in "systems" AMG) |
| 2 | each domain is generated by agglomeration (default)       |

#### 6.2.41

```
int HYPRE_BoomerAMGSetSym (HYPRE_Solver solver, int sym)
```

(Optional) Defines symmetry for ParaSAILS. For further explanation see description of ParaSAILS.

#### 6.2.42

```
int HYPRE_BoomerAMGSetLevel (HYPRE_Solver solver, int level)
```

(Optional) Defines number of levels for ParaSAILS. For further explanation see description of ParaSAILS.

#### 6.2.43

```
int
HYPRE_BoomerAMGSetThreshold (HYPRE_Solver solver, double threshold)
```

(Optional) Defines threshold for ParaSAILS. For further explanation see description of ParaSAILS.

#### 6.2.44

```
int HYPRE_BoomerAMGSetFilter (HYPRE_Solver solver, double filter)
```

(Optional) Defines filter for ParaSAILS. For further explanation see description of ParaSAILS.

**6.2.45**

```
int HYPRE_BoomerAMGSetDropTol (HYPRE_Solver solver, double drop_tol)
```

(Optional) Defines drop tolerance for PILUT. For further explanation see description of PILUT.

**6.2.46**

```
int
HYPRE_BoomerAMGSetMaxNzPerRow (HYPRE_Solver solver, int
max_nz_per_row)
```

(Optional) Defines maximal number of nonzeros for PILUT. For further explanation see description of PILUT.

**6.2.47**

```
int
HYPRE_BoomerAMGSetEuclidFile (HYPRE_Solver solver, char *euclidfile)
```

(Optional) Defines name of an input file for Euclid parameters. For further explanation see description of Euclid.

**6.2.48**

```
int HYPRE_BoomerAMGSetGSMG (HYPRE_Solver solver, int gsmg)
```

(Optional) Specifies the use of GSMG - geometrically smooth coarsening and interpolation. Currently any nonzero value for gsmg will lead to the use of GSMG. The default is 0, i.e. (GSMG is not used)

## 6.3

## ParCSR ParaSails Preconditioner

### Names

|       |     |                                                                                                                                                                            |     |
|-------|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
|       | int | <b>HYPRE_ParaSailsCreate</b> (MPI_Comm comm, HYPRE_Solver *solver)<br><i>Create a ParaSails preconditioner</i>                                                             |     |
|       | int | <b>HYPRE_ParaSailsDestroy</b> (HYPRE_Solver solver)<br><i>Destroy a ParaSails preconditioner</i>                                                                           |     |
| 6.3.1 | int | <b>HYPRE_ParaSailsSetup</b> (HYPRE_Solver solver, HYPRE_ParCSRMatrix A,<br>HYPRE_ParVector b, HYPRE_ParVector x)<br><i>Set up the ParaSails preconditioner</i> .....       | 104 |
| 6.3.2 | int | <b>HYPRE_ParaSailsSolve</b> (HYPRE_Solver solver, HYPRE_ParCSRMatrix A,<br>HYPRE_ParVector b, HYPRE_ParVector x)<br><i>Apply the ParaSails preconditioner</i> .....        | 104 |
| 6.3.3 | int | <b>HYPRE_ParaSailsSetParams</b> (HYPRE_Solver solver, double thresh,<br>int nlevels)<br><i>Set the threshold and levels parameter for the ParaSails preconditioner</i> ... | 104 |
| 6.3.4 | int | <b>HYPRE_ParaSailsSetFilter</b> (HYPRE_Solver solver, double filter)<br><i>Set the filter parameter for the ParaSails preconditioner</i> .....                             | 105 |
| 6.3.5 | int | <b>HYPRE_ParaSailsSetSym</b> (HYPRE_Solver solver, int sym)<br><i>Set the symmetry parameter for the ParaSails preconditioner</i> .....                                    | 105 |
| 6.3.6 | int | <b>HYPRE_ParaSailsSetLoadbal</b> (HYPRE_Solver solver, double loadbal)<br><i>Set the load balance parameter for the ParaSails preconditioner</i> .....                     | 105 |
| 6.3.7 | int | <b>HYPRE_ParaSailsSetReuse</b> (HYPRE_Solver solver, int reuse)<br><i>Set the pattern reuse parameter for the ParaSails preconditioner</i> .....                           | 106 |
| 6.3.8 | int | <b>HYPRE_ParaSailsSetLogging</b> (HYPRE_Solver solver, int logging)<br><i>Set the logging parameter for the ParaSails preconditioner</i> .....                             | 106 |
| 6.3.9 | int | <b>HYPRE_ParaSailsBuildIJMatrix</b> (HYPRE_Solver solver,<br>HYPRE_IJMatrix *pij_A)<br><i>Build IJ Matrix of the sparse approximate inverse (factor)</i> .....             | 107 |

Parallel sparse approximate inverse preconditioner for the ParCSR matrix format.

**6.3.1**

```
int
HYPRE_ParaSailsSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)
```

Set up the ParaSails preconditioner. This function should be passed to the iterative solver `SetPrecond` function.

**Parameters:**

- `solver` — [IN] Preconditioner object to set up.
- `A` — [IN] ParCSR matrix used to construct the preconditioner.
- `b` — Ignored by this function.
- `x` — Ignored by this function.

**6.3.2**

```
int
HYPRE_ParaSailsSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)
```

Apply the ParaSails preconditioner. This function should be passed to the iterative solver `SetPrecond` function.

**Parameters:**

- `solver` — [IN] Preconditioner object to apply.
- `A` — Ignored by this function.
- `b` — [IN] Vector to precondition.
- `x` — [OUT] Preconditioned vector.

**6.3.3**

```
int
HYPRE_ParaSailsSetParams (HYPRE_Solver solver, double thresh, int nlevels)
```

Set the threshold and levels parameter for the ParaSails preconditioner. The accuracy and cost of ParaSails are parameterized by these two parameters. Lower values of the threshold parameter and higher values of levels parameter lead to more accurate, but more expensive preconditioners.



**Parameters:**

- `solver` — [IN] Preconditioner object for which to set parameters.
- `thresh` — [IN] Value of threshold parameter,  $0 \leq \text{thresh} \leq 1$ . The default value is 0.1.
- `nlevels` — [IN] Value of levels parameter,  $0 \leq \text{nlevels}$ . The default value is 1.

#### 6.3.4

```
int HYPRE_ParaSailsSetFilter (HYPRE_Solver solver, double filter)
```

Set the filter parameter for the ParaSails preconditioner.

**Parameters:**

- `solver` — [IN] Preconditioner object for which to set filter parameter.
- `filter` — [IN] Value of filter parameter. The filter parameter is used to drop small nonzeros in the preconditioner, to reduce the cost of applying the preconditioner. Values from 0.05 to 0.1 are recommended. The default value is 0.1.

#### 6.3.5

```
int HYPRE_ParaSailsSetSym (HYPRE_Solver solver, int sym)
```

Set the symmetry parameter for the ParaSails preconditioner.

**Parameters:**

- `solver` — [IN] Preconditioner object for which to set symmetry parameter.
- `sym` — [IN] Value of the symmetry parameter:

| value | meaning                                                                 |
|-------|-------------------------------------------------------------------------|
| 0     | nonsymmetric and/or indefinite problem, and nonsymmetric preconditioner |
| 1     | SPD problem, and SPD (factored) preconditioner                          |
| 2     | nonsymmetric, definite problem, and SPD (factored) preconditioner       |

#### 6.3.6

```
int HYPRE_ParaSailsSetLoadbal (HYPRE_Solver solver, double loadbal)
```

Set the load balance parameter for the ParaSails preconditioner.

**Parameters:**

`solver` — [IN] Preconditioner object for which to set the load balance parameter.

`loadbal` — [IN] Value of the load balance parameter,  $0 \leq \text{loadbal} \leq 1$ . A zero value indicates that no load balance is attempted; a value of unity indicates that perfect load balance will be attempted. The recommended value is 0.9 to balance the overhead of data exchanges for load balancing. No load balancing is needed if the preconditioner is very sparse and fast to construct. The default value when this parameter is not set is 0.

### 6.3.7

```
int HYPRE_ParaSailsSetReuse (HYPRE_Solver solver, int reuse)
```

Set the pattern reuse parameter for the ParaSails preconditioner.

**Parameters:**

`solver` — [IN] Preconditioner object for which to set the pattern reuse parameter.

`reuse` — [IN] Value of the pattern reuse parameter. A nonzero value indicates that the pattern of the preconditioner should be reused for subsequent constructions of the preconditioner. A zero value indicates that the preconditioner should be constructed from scratch. The default value when this parameter is not set is 0.

### 6.3.8

```
int HYPRE_ParaSailsSetLogging (HYPRE_Solver solver, int logging)
```

Set the logging parameter for the ParaSails preconditioner.

**Parameters:**

`solver` — [IN] Preconditioner object for which to set the logging parameter.

`logging` — [IN] Value of the logging parameter. A nonzero value sends statistics of the setup procedure to stdout. The default value when this parameter is not set is 0.

## 6.3.9

```
int
HYPRE_ParaSailsBuildIJMatrix (HYPRE_Solver solver, HYPRE_IJMatrix
*pij_A)
```

Build IJ Matrix of the sparse approximate inverse (factor). This function explicitly creates the IJ Matrix corresponding to the sparse approximate inverse or the inverse factor. Example: `HYPRE_IJMatrix ij_A; HYPRE_ParaSailsBuildIJMatrix(solver, &ij_A);`

**Parameters:**                    `solver` — [IN] Preconditioner object.  
                                      `pij_A` — [OUT] Pointer to the IJ Matrix.

## 6.4

## ParCSR Euclid Preconditioner

## Names

```
int
HYPRE_EuclidCreate (MPI_Comm comm, HYPRE_Solver *solver)
 Create a Euclid object

int
HYPRE_EuclidDestroy (HYPRE_Solver solver)
 Destroy a Euclid object

6.4.1 int
HYPRE_EuclidSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A,
 HYPRE_ParVector b, HYPRE_ParVector x)
 Set up the Euclid preconditioner 108

6.4.2 int
HYPRE_EuclidSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A,
 HYPRE_ParVector b, HYPRE_ParVector x)
 Apply the Euclid preconditioner 108

6.4.3 int
HYPRE_EuclidSetParams (HYPRE_Solver solver, int argc, char *argv[])
 Insert (name, value) pairs in Euclid's options database by passing Euclid
the command line (or an array of strings) 109

6.4.4 int
HYPRE_EuclidSetParamsFromFile (HYPRE_Solver solver, char *filename)
 Insert (name, value) pairs in Euclid's options database 109
```

MPI Parallel ILU preconditioner

Options summary:

| Option    | Default   | Synopsis                             |
|-----------|-----------|--------------------------------------|
| -level    | 1         | ILU( $k$ ) factorization level       |
| -bj       | 0 (false) | Use Block Jacobi ILU instead of PILU |
| -eu_stats | 0 (false) | Print internal timing and statistics |
| -eu_mem   | 0 (false) | Print internal memory usage          |

#### 6.4.1

```
int
HYPRE_EuclidSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)
```

Set up the Euclid preconditioner. This function should be passed to the iterative solver `SetPrecond` function.

**Parameters:**

- `solver` — [IN] Preconditioner object to set up.
- `A` — [IN] ParCSR matrix used to construct the preconditioner.
- `b` — Ignored by this function.
- `x` — Ignored by this function.

#### 6.4.2

```
int
HYPRE_EuclidSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)
```

Apply the Euclid preconditioner. This function should be passed to the iterative solver `SetPrecond` function.

**Parameters:**

- `solver` — [IN] Preconditioner object to apply.
- `A` — Ignored by this function.
- `b` — [IN] Vector to precondition.
- `x` — [OUT] Preconditioned vector.

## 6.4.3

```
int HYPRE_EuclidSetParams (HYPRE_Solver solver, int argc, char *argv[])
```

Insert (name, value) pairs in Euclid's options database by passing Euclid the command line (or an array of strings). All Euclid options (e.g, level, drop-tolerance) are stored in this database. If a (name, value) pair already exists, this call updates the value. See also: `HYPRE_EuclidSetParamsFromFile`.

**Parameters:**

- `argc` — [IN] Length of argv array
- `argv` — [IN] Array of strings

## 6.4.4

```
int HYPRE_EuclidSetParamsFromFile (HYPRE_Solver solver, char *filename)
```

Insert (name, value) pairs in Euclid's options database. Each line of the file should either begin with a "#," indicating a comment line, or contain a (name value) pair, e.g:

```
>cat optionsFile
#sample runtime parameter file
-blockJacobi 3
-matFile /home/hysom/myfile.euclid
-doSomething true
-xx_coeff -1.0
```

See also: `HYPRE_EuclidSetParams`.

**Parameters:**

- `filename`[IN] — Pathname/filename to read

## 6.5

## ParCSR Pilut Preconditioner

### Names

```
int HYPRE_ParCSRPilutCreate (MPL_Comm comm, HYPRE_Solver *solver)
 Create a preconditioner object
```

```
int
```

**HYPRE\_ParCSRPilotDestroy** (HYPRE\_Solver solver)

*Destroy a preconditioner object*

int

**HYPRE\_ParCSRPilotSetup** (HYPRE\_Solver solver,  
HYPRE\_ParCSRMatrix A,  
HYPRE\_ParVector b, HYPRE\_ParVector x)

int

**HYPRE\_ParCSRPilotSolve** (HYPRE\_Solver solver,  
HYPRE\_ParCSRMatrix A,  
HYPRE\_ParVector b, HYPRE\_ParVector x)

*Precondition the system*

int

**HYPRE\_ParCSRPilotSetMaxIter** (HYPRE\_Solver solver, int max\_iter)  
*(Optional) Set maximum number of iterations*

int

**HYPRE\_ParCSRPilotSetDropTolerance** (HYPRE\_Solver solver, double tol)  
*(Optional)*

int

**HYPRE\_ParCSRPilotSetFactorRowSize** (HYPRE\_Solver solver, int size)  
*(Optional)*

## 6.6

### ParCSR AMS Solver and Preconditioner

#### Names

int

**HYPRE\_AMSCreate** (HYPRE\_Solver \*solver)  
*Create an AMS solver object*

int

**HYPRE\_AMSDestroy** (HYPRE\_Solver solver)  
*Destroy an AMS solver object*

6.6.1

int

**HYPRE\_AMSSetup** (HYPRE\_Solver solver, HYPRE\_ParCSRMatrix A,  
HYPRE\_ParVector b, HYPRE\_ParVector x)  
*Set up the AMS solver or preconditioner* ..... 112

6.6.2

int

**HYPRE\_AMSSolve** (HYPRE\_Solver solver, HYPRE\_ParCSRMatrix A,  
HYPRE\_ParVector b, HYPRE\_ParVector x)  
*Solve the system or apply AMS as a preconditioner* ..... 113

6.6.3

int

**HYPRE\_AMSSetDimension** (HYPRE\_Solver solver, int dim)  
*(Optional) Sets the problem dimension (2 or 3)* ..... 113

6.6.4

int

|        |                  |                                                                                                                               |                                                                                                                                                                                                                                                                                                   |     |
|--------|------------------|-------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
|        | <b>HYPRE_AMS</b> | <b>SetDiscreteGradient</b> (HYPRE_Solver solver,<br>HYPRE_ParCSRMatrix G)<br><i>Sets the discrete gradient matrix G</i> ..... | 113                                                                                                                                                                                                                                                                                               |     |
| 6.6.5  | int              | <b>HYPRE_AMS</b>                                                                                                              | <b>SetCoordinateVectors</b> (HYPRE_Solver solver,<br>HYPRE_ParVector x,<br>HYPRE_ParVector y,<br>HYPRE_ParVector z)<br><i>Sets the x, y and z coordinates of the vertices in the mesh</i> .....                                                                                                   | 113 |
| 6.6.6  | int              | <b>HYPRE_AMS</b>                                                                                                              | <b>SetEdgeConstantVectors</b> (HYPRE_Solver solver,<br>HYPRE_ParVector Gx,<br>HYPRE_ParVector Gy,<br>HYPRE_ParVector Gz)<br><i>Sets the vectors Gx, Gy and Gz which give the representations of the constant vector fields (1, 0, 0), (0, 1, 0) and (0, 0, 1) in the edge element basis</i> ..... | 114 |
| 6.6.7  | int              | <b>HYPRE_AMS</b>                                                                                                              | <b>SetAlphaPoissonMatrix</b> (HYPRE_Solver solver,<br>HYPRE_ParCSRMatrix A_alpha)<br><i>(Optional) Sets the matrix <math>A_\alpha</math> corresponding to the Poisson problem with coefficient <math>\alpha</math> (the curl-curl term coefficient in the Maxwell problem)</i> .....              | 114 |
| 6.6.8  | int              | <b>HYPRE_AMS</b>                                                                                                              | <b>SetBetaPoissonMatrix</b> (HYPRE_Solver solver,<br>HYPRE_ParCSRMatrix A_beta)<br><i>(Optional) Sets the matrix <math>A_\beta</math> corresponding to the Poisson problem with coefficient <math>\beta</math> (the mass term coefficient in the Maxwell problem)</i> .....                       | 114 |
| 6.6.9  | int              | <b>HYPRE_AMS</b>                                                                                                              | <b>SetMaxIter</b> (HYPRE_Solver solver, int maxit)<br><i>(Optional) Sets maximum number of iterations, if AMS is used as a solver</i> .....                                                                                                                                                       | 115 |
| 6.6.10 | int              | <b>HYPRE_AMS</b>                                                                                                              | <b>SetTol</b> (HYPRE_Solver solver, double tol)<br><i>(Optional) Set the convergence tolerance, if AMS is used as a solver</i> ....                                                                                                                                                               | 115 |
| 6.6.11 | int              | <b>HYPRE_AMS</b>                                                                                                              | <b>SetCycleType</b> (HYPRE_Solver solver, int cycle_type)<br><i>(Optional) Choose which three-level solver to use</i> .....                                                                                                                                                                       | 115 |
| 6.6.12 | int              | <b>HYPRE_AMS</b>                                                                                                              | <b>SetPrintLevel</b> (HYPRE_Solver solver, int print_level)<br><i>(Optional) Control how much information is printed during the solution iterations</i> .....                                                                                                                                     | 116 |
| 6.6.13 | int              | <b>HYPRE_AMS</b>                                                                                                              | <b>SetSmoothingOptions</b> (HYPRE_Solver solver, int relax_type,<br>int relax_times, double relax_weight,<br>double omega)<br><i>(Optional) Sets relaxation parameters for A</i> .....                                                                                                            | 116 |
| 6.6.14 | int              |                                                                                                                               |                                                                                                                                                                                                                                                                                                   |     |

- HYPRE\_AMSSetAlphaAMGOptions** (HYPRE\_Solver solver,  
int alpha\_coarsen\_type,  
int alpha\_agg\_levels,  
int alpha\_relax\_type,  
double alpha\_strength\_threshold)  
*(Optional) Sets AMG parameters for  $B_{\Pi}$*  ..... 116
- 6.6.15 int  
**HYPRE\_AMSSetBetaAMGOptions** (HYPRE\_Solver solver,  
int beta\_coarsen\_type,  
int beta\_agg\_levels, int beta\_relax\_type,  
double beta\_strength\_threshold)  
*(Optional) Sets AMG parameters for  $B_G$*  ..... 116
- int  
**HYPRE\_AMSGetNumIterations** (HYPRE\_Solver solver,  
int \*num\_iterations)  
*Returns the number of iterations taken*
- int  
**HYPRE\_AMSGetFinalRelativeResidualNorm** (HYPRE\_Solver solver,  
double \*rel\_resid\_norm)  
*Returns the norm of the final relative residual*
- 6.6.16 int  
**HYPRE\_AMSConstructDiscreteGradient** (HYPRE\_ParCSRMatrix A,  
HYPRE\_ParVector x\_coord,  
int \*edge\_vertex,  
HYPRE\_ParCSRMatrix \*G)  
*Construct and return the discrete gradient matrix  $G$  using some edge and vertex information* ..... 117

Parallel auxiliary space Maxwell solver and preconditioner

### 6.6.1

```
int
HYPRE_AMSSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)
```

Set up the AMS solver or preconditioner. If used as a preconditioner, this function should be passed to the iterative solver `SetPrecond` function.

**Parameters:**

- `solver` — [IN] object to be set up.
- `A` — [IN] ParCSR matrix used to construct the solver/preconditioner.
- `b` — Ignored by this function.
- `x` — Ignored by this function.



**6.6.2**

```
int
HYPRE_AMSSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)
```

Solve the system or apply AMS as a preconditioner. If used as a preconditioner, this function should be passed to the iterative solver `SetPrecond` function.

**Parameters:**

- `solver` — [IN] solver or preconditioner object to be applied.
- `A` — [IN] ParCSR matrix, matrix of the linear system to be solved
- `b` — [IN] right hand side of the linear system to be solved
- `x` — [OUT] approximated solution of the linear system to be solved

**6.6.3**

```
int HYPRE_AMSSetDimension (HYPRE_Solver solver, int dim)
```

(Optional) Sets the problem dimension (2 or 3). The default is 3.

**6.6.4**

```
int
HYPRE_AMSSetDiscreteGradient (HYPRE_Solver solver,
HYPRE_ParCSRMatrix G)
```

Sets the discrete gradient matrix  $G$ . This function should be called before `HYPRE_AMSSetup()`!

**6.6.5**

```
int
HYPRE_AMSSetCoordinateVectors (HYPRE_Solver solver,
HYPRE_ParVector x, HYPRE_ParVector y, HYPRE_ParVector z)
```

Sets the  $x$ ,  $y$  and  $z$  coordinates of the vertices in the mesh.

Either `HYPRE_AMSSetCoordinateVectors()` or `HYPRE_AMSSetEdgeConstantVectors()` should be called before `HYPRE_AMSSetup()`!

#### 6.6.6

```
int
HYPRE_AMSSetEdgeConstantVectors (HYPRE_Solver solver,
HYPRE_ParVector Gx, HYPRE_ParVector Gy, HYPRE_ParVector Gz)
```

Sets the vectors  $Gx$ ,  $Gy$  and  $Gz$  which give the representations of the constant vector fields  $(1, 0, 0)$ ,  $(0, 1, 0)$  and  $(0, 0, 1)$  in the edge element basis.

Either `HYPRE_AMSSetCoordinateVectors()` or `HYPRE_AMSSetEdgeConstantVectors()` should be called before `HYPRE_AMSSetup()`!

#### 6.6.7

```
int
HYPRE_AMSSetAlphaPoissonMatrix (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A_alpha)
```

(Optional) Sets the matrix  $A_\alpha$  corresponding to the Poisson problem with coefficient  $\alpha$  (the curl-curl term coefficient in the Maxwell problem).

If this function is called, the coarse space solver on the range of  $\Pi^T$  is a block-diagonal version of  $A_\Pi$ . If this function is not called, the coarse space solver on the range of  $\Pi^T$  is constructed as  $\Pi^T A \Pi$  in `HYPRE_AMSSetup()`. See the user's manual for more details.

#### 6.6.8

```
int
HYPRE_AMSSetBetaPoissonMatrix (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A_beta)
```

(Optional) Sets the matrix  $A_\beta$  corresponding to the Poisson problem with coefficient  $\beta$  (the mass term coefficient in the Maxwell problem).

If not given, the Poisson matrix will be computed in `HYPRE_AMSSetup()`. If the given matrix is `NULL`, we assume that  $\beta$  is identically 0 and use two-level (instead of three-level) methods. See the user's manual for more details.

### 6.6.9

```
int HYPRE_AMSsetMaxIter (HYPRE_Solver solver, int maxit)
```

(Optional) Sets maximum number of iterations, if AMS is used as a solver. To use AMS as a preconditioner, set the maximum number of iterations to 1. The default is 20.

### 6.6.10

```
int HYPRE_AMSsetTol (HYPRE_Solver solver, double tol)
```

(Optional) Set the convergence tolerance, if AMS is used as a solver. When using AMS as a preconditioner, set the tolerance to 0.0. The default is  $10^{-6}$ .

### 6.6.11

```
int HYPRE_AMSsetCycleType (HYPRE_Solver solver, int cycle_type)
```

(Optional) Choose which three-level solver to use. Possible values are:

|   |                                         |
|---|-----------------------------------------|
| 1 | 3-level multiplicative solver (01210)   |
| 3 | 3-level multiplicative solver (02120)   |
| 5 | 3-level multiplicative solver (0102010) |
| 7 | 3-level multiplicative solver (0201020) |
| 2 | 3-level additive solver (0+1+2)         |
| 4 | 3-level additive solver (010+2)         |
| 6 | 3-level additive solver (1+020)         |
| 8 | 3-level additive solver (010+020)       |

The default is 1. See the user's manual for more details.

**6.6.12**

```
int HYPRE_AMSSetPrintLevel (HYPRE_Solver solver, int print_level)
```

(Optional) Control how much information is printed during the solution iterations. The default is 1 (print residual norm at each step).

**6.6.13**

```
int
HYPRE_AMSSetSmoothingOptions (HYPRE_Solver solver, int relax_type, int
relax_times, double relax_weight, double omega)
```

(Optional) Sets relaxation parameters for  $A$ . The defaults are 2, 1, 1.0, 1.0.

**6.6.14**

```
int
HYPRE_AMSSetAlphaAMGOptions (HYPRE_Solver solver, int
alpha_coarsen_type, int alpha_agg_levels, int alpha_relax_type, double
alpha_strength_threshold)
```

(Optional) Sets AMG parameters for  $B_{II}$ . The defaults are 10, 1, 3, 0.25. See the user's manual for more details.

**6.6.15**

```
int
HYPRE_AMSSetBetaAMGOptions (HYPRE_Solver solver, int
beta_coarsen_type, int beta_agg_levels, int beta_relax_type, double
beta_strength_threshold)
```

(Optional) Sets AMG parameters for  $B_G$ . The defaults are 10, 1, 3, 0.25. See the user's manual for more details.

## 6.6.16

```
int
HYPRE_AMSConstructDiscreteGradient (HYPRE_ParCSRMatrix A,
HYPRE_ParVector x_coord, int *edge_vertex, HYPRE_ParCSRMatrix *G)
```

Construct and return the discrete gradient matrix G using some edge and vertex information. We assume that edge\_vertex lists the edge vertices consecutively, and that the orientation of edge i depends only on the sign of edge\_vertex[2\*i+1] - edge\_vertex[2\*i].

## 6.7

## ParCSR Hybrid Solver

## Names

```
int
HYPRE_ParCSRHybridCreate (HYPRE_Solver *solver)
 Create solver object

int
HYPRE_ParCSRHybridDestroy (HYPRE_Solver solver)
 Destroy solver object

6.7.1 int
HYPRE_ParCSRHybridSetup (HYPRE_Solver solver,
 HYPRE_ParCSRMatrix A,
 HYPRE_ParVector b, HYPRE_ParVector x)
 Setup the hybrid solver 121

6.7.2 int
HYPRE_ParCSRHybridSolve (HYPRE_Solver solver,
 HYPRE_ParCSRMatrix A,
 HYPRE_ParVector b, HYPRE_ParVector x)
 Solve linear system 121

6.7.3 int
HYPRE_ParCSRHybridSetTol (HYPRE_Solver solver, double tol)
 Set the convergence tolerance for the Krylov solver 121

int
HYPRE_ParCSRHybridSetConvergenceTol (HYPRE_Solver solver,
 double cf_tol)
 Set the desired convergence factor

int
HYPRE_ParCSRHybridSetDSCGMaxIter (HYPRE_Solver solver,
 int dscg_max_its)
 Set the maximal number of iterations for the diagonally preconditioned solver

int
```

---

|       |                                                                                                                                                                                                                                                                                        |     |
|-------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
|       | <b>HYPRE_ParCSRHybridSetPCGMaxIter</b> (HYPRE_Solver solver,<br>int pcg_max_its)<br><i>Set the maximal number of iterations for the AMG preconditioned solver</i>                                                                                                                      |     |
| 6.7.4 | int<br><b>HYPRE_ParCSRHybridSetSolverType</b> (HYPRE_Solver solver,<br>int solver_type)<br><i>Set the desired solver type</i> .....                                                                                                                                                    | 121 |
| 6.7.5 | int<br><b>HYPRE_ParCSRHybridSetKDim</b> (HYPRE_Solver solver, int k_dim)<br><i>Set the Krylov dimension for restarted GMRES</i> .....                                                                                                                                                  | 122 |
|       | int<br><b>HYPRE_ParCSRHybridSetTwoNorm</b> (HYPRE_Solver solver, int two_norm)<br><i>Set the type of norm for PCG</i>                                                                                                                                                                  |     |
|       | int<br><b>HYPRE_ParCSRHybridSetStopCrit</b> (HYPRE_Solver solver, int stop_crit)<br><i>Set the choice of stopping criterion for PCG</i>                                                                                                                                                |     |
|       | int<br><b>HYPRE_ParCSRHybridSetPrecond</b> (HYPRE_Solver solver,<br>HYPRE_PtrToParSolverFcn precondition,<br>HYPRE_PtrToParSolverFcn<br>precond_setup,<br>HYPRE_Solver precondition_solver)<br><i>Set preconditioner if wanting to use one that is not set up by the hybrid solver</i> |     |
|       | int<br><b>HYPRE_ParCSRHybridSetLogging</b> (HYPRE_Solver solver, int logging)<br><i>Set logging parameter (default: 0, no logging)</i>                                                                                                                                                 |     |
|       | int<br><b>HYPRE_ParCSRHybridSetPrintLevel</b> (HYPRE_Solver solver,<br>int print_level)<br><i>Set print level (default: 0, no printing)</i>                                                                                                                                            |     |
| 6.7.6 | int<br><b>HYPRE_ParCSRHybridSetStrongThreshold</b> ( HYPRE_Solver solver,<br>double strong_threshold )<br><i>(Optional) Sets AMG strength threshold</i> .....                                                                                                                          | 122 |
| 6.7.7 | int<br><b>HYPRE_ParCSRHybridSetMaxRowSum</b> ( HYPRE_Solver solver,<br>double max_row_sum )<br><i>(Optional) Sets a parameter to modify the definition of strength for diagonal dominant portions of the matrix</i> .....                                                              | 122 |
| 6.7.8 | int<br><b>HYPRE_ParCSRHybridSetTruncFactor</b> ( HYPRE_Solver solver,<br>double trunc_factor )<br><i>(Optional) Defines a truncation factor for the interpolation</i> .....                                                                                                            | 122 |
| 6.7.9 | int<br><b>HYPRE_ParCSRHybridSetMaxLevels</b> ( HYPRE_Solver solver,<br>int max_levels )<br><i>(Optional) Defines the maximal number of levels used for AMG</i> .....                                                                                                                   | 123 |
|       | int                                                                                                                                                                                                                                                                                    |     |

---

|        |     |                                                                                                                                                                                                                     |     |
|--------|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
|        |     | <b>HYPRE_ParCSRHybridSetMeasureType</b> ( HYPRE_Solver solver,<br>int measure_type )<br><i>(Optional) Defines whether local or global measures are used</i>                                                         |     |
| 6.7.10 | int | <b>HYPRE_ParCSRHybridSetCoarsenType</b> ( HYPRE_Solver solver,<br>int coarsen_type )<br><i>(Optional) Defines which parallel coarsening algorithm is used</i>                                                       | 123 |
| 6.7.11 | int | <b>HYPRE_ParCSRHybridSetCycleType</b> ( HYPRE_Solver solver,<br>int cycle_type )<br><i>(Optional) Defines the type of cycle</i>                                                                                     | 123 |
| 6.7.12 | int | <b>HYPRE_ParCSRHybridSetNumSweeps</b> ( HYPRE_Solver solver,<br>int num_sweeps )<br><i>(Optional) Sets the number of sweeps</i>                                                                                     | 124 |
| 6.7.13 | int | <b>HYPRE_ParCSRHybridSetCycleNumSweeps</b> ( HYPRE_Solver solver,<br>int num_sweeps, int k )<br><i>(Optional) Sets the number of sweeps at a specified cycle</i>                                                    | 124 |
| 6.7.14 | int | <b>HYPRE_ParCSRHybridSetRelaxType</b> ( HYPRE_Solver solver,<br>int relax_type )<br><i>(Optional) Defines the smoother to be used</i>                                                                               | 124 |
| 6.7.15 | int | <b>HYPRE_ParCSRHybridSetCycleRelaxType</b> ( HYPRE_Solver solver,<br>int relax_type, int k )<br><i>(Optional) Defines the smoother at a given cycle</i>                                                             | 125 |
| 6.7.16 | int | <b>HYPRE_ParCSRHybridSetRelaxOrder</b> ( HYPRE_Solver solver,<br>int relax_order )<br><i>(Optional) Defines in which order the points are relaxed</i>                                                               | 125 |
| 6.7.17 | int | <b>HYPRE_ParCSRHybridSetRelaxWt</b> ( HYPRE_Solver solver,<br>double relax_wt )<br><i>(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid<br/>SOR on all levels</i>                             | 125 |
| 6.7.18 | int | <b>HYPRE_ParCSRHybridSetLevelRelaxWt</b> ( HYPRE_Solver solver,<br>double relax_wt, int level )<br><i>(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid<br/>SOR on the user defined level</i> | 126 |
| 6.7.19 | int | <b>HYPRE_ParCSRHybridSetOuterWt</b> ( HYPRE_Solver solver,<br>double outer_wt )<br><i>(Optional) Defines the outer relaxation weight for hybrid SOR and SSOR<br/>on all levels</i>                                  | 126 |
| 6.7.20 | int |                                                                                                                                                                                                                     |     |

---

|        |     |                                                                                                                                                                                                                     |     |
|--------|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
|        |     | <b>HYPRE_ParCSRHybridSetLevelOuterWt</b> ( HYPRE_Solver solver,<br>double outer_wt, int level )<br><i>(Optional) Defines the outer relaxation weight for hybrid SOR or SSOR on<br/>the user defined level</i> ..... | 126 |
| 6.7.21 | int | <b>HYPRE_ParCSRHybridSetAggNumLevels</b> ( HYPRE_Solver solver,<br>int agg_num_levels )<br><i>(Optional) Defines the number of levels of aggressive coarsening, starting<br/>with the finest level</i> .....        | 127 |
| 6.7.22 | int | <b>HYPRE_ParCSRHybridSetNumPaths</b> ( HYPRE_Solver solver,<br>int num_paths )<br><i>(Optional) Defines the degree of aggressive coarsening</i> .....                                                               | 127 |
| 6.7.23 | int | <b>HYPRE_ParCSRHybridSetNumFunctions</b> ( HYPRE_Solver solver,<br>int num_functions)<br><i>(Optional) Sets the size of the system of PDEs, if using the systems version</i><br>.....                               | 127 |
| 6.7.24 | int | <b>HYPRE_ParCSRHybridSetDofFunc</b> ( HYPRE_Solver solver, int *dof_func )<br><i>(Optional) Sets the mapping that assigns the function to each variable, if<br/>using the systems version</i> .....                 | 128 |
| 6.7.25 | int | <b>HYPRE_ParCSRHybridSetNodal</b> ( HYPRE_Solver solver, int nodal )<br><i>(Optional) Sets whether to use the nodal systems version</i> .....                                                                       | 128 |
|        | int | <b>HYPRE_ParCSRHybridGetNumIterations</b> (HYPRE_Solver solver,<br>int *num_its)<br><i>Retrieves the total number of iterations</i>                                                                                 |     |
|        | int | <b>HYPRE_ParCSRHybridGetDSCGNumIterations</b> (HYPRE_Solver solver,<br>int *dscg_num_its)<br><i>Retrieves the number of iterations used by the diagonally scaled solver</i>                                         |     |
|        | int | <b>HYPRE_ParCSRHybridGetPCGNumIterations</b> (HYPRE_Solver solver,<br>int *pcg_num_its)<br><i>Retrieves the number of iterations used by the AMG preconditioned solver</i>                                          |     |
|        | int | <b>HYPRE_ParCSRHybridGetFinalRelativeResidualNorm</b> (HYPRE_Solver<br>solver,<br>double *norm)<br><i>Retrieves the final relative residual norm</i>                                                                |     |



## 6.7.1

```
int
HYPRE_ParCSRHybridSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix
A, HYPRE_ParVector b, HYPRE_ParVector x)
```

**Parameters:**

- `solver` — [IN] object to be set up.
- `A` — [IN] ParCSR matrix used to construct the solver/preconditioner.
- `b` — Ignored by this function.
- `x` — Ignored by this function.

## 6.7.2

```
int
HYPRE_ParCSRHybridSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix
A, HYPRE_ParVector b, HYPRE_ParVector x)
```

**Parameters:**

- `solver` — [IN] solver or preconditioner object to be applied.
- `A` — [IN] ParCSR matrix, matrix of the linear system to be solved
- `b` — [IN] right hand side of the linear system to be solved
- `x` — [OUT] approximated solution of the linear system to be solved

## 6.7.3

```
int HYPRE_ParCSRHybridSetTol (HYPRE_Solver solver, double tol)
```

Set the convergence tolerance for the Krylov solver. The default is 1.e-7.

## 6.7.4

```
int
HYPRE_ParCSRHybridSetSolverType (HYPRE_Solver solver, int solver_type)
```

Set the desired solver type. There are the following options:

- 1 PCG (default)
- 2 GMRES
- 3 BiCGSTAB

**6.7.5**

```
int HYPRE_ParCSRHybridSetKDim (HYPRE_Solver solver, int k_dim)
```

Set the Krylov dimension for restarted GMRES. The default is 5.

**6.7.6**

```
int
HYPRE_ParCSRHybridSetStrongThreshold (HYPRE_Solver solver, double
strong_threshold)
```

(Optional) Sets AMG strength threshold. The default is 0.25. For 2d Laplace operators, 0.25 is a good value, for 3d Laplace operators, 0.5 or 0.6 is a better value. For elasticity problems, a large strength threshold, such as 0.9, is often better.

**6.7.7**

```
int
HYPRE_ParCSRHybridSetMaxRowSum (HYPRE_Solver solver, double
max_row_sum)
```

(Optional) Sets a parameter to modify the definition of strength for diagonal dominant portions of the matrix. The default is 0.9. If max\_row\_sum is 1, no checking for diagonally dominant rows is performed.

**6.7.8**

```
int
HYPRE_ParCSRHybridSetTruncFactor (HYPRE_Solver solver, double
trunc_factor)
```

(Optional) Defines a truncation factor for the interpolation. The default is 0.

### 6.7.9

```
int
HYPRE_ParCSRHybridSetMaxLevels (HYPRE_Solver solver, int max_levels)
```

(Optional) Defines the maximal number of levels used for AMG. The default is 25.

### 6.7.10

```
int
HYPRE_ParCSRHybridSetCoarsenType (HYPRE_Solver solver, int
coarsen_type)
```

(Optional) Defines which parallel coarsening algorithm is used. There are the following options for `coarsen_type`:

|    |                                                                                                                                                         |
|----|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | CLJP-coarsening (a parallel coarsening algorithm using independent sets).                                                                               |
| 1  | classical Ruge-Stueben coarsening on each processor, no boundary treatment                                                                              |
| 3  | classical Ruge-Stueben coarsening on each processor, followed by a third pass, which adds coarse points on the boundaries                               |
| 6  | Falgout coarsening (uses 1 first, followed by CLJP using the interior coarse points generated by 1 as its first independent set)                        |
| 7  | CLJP-coarsening (using a fixed random vector, for debugging purposes only)                                                                              |
| 8  | PMIS-coarsening (a parallel coarsening algorithm using independent sets with lower complexities than CLJP, might also lead to slower convergence)       |
| 9  | PMIS-coarsening (using a fixed random vector, for debugging purposes only)                                                                              |
| 10 | HMIS-coarsening (uses one pass Ruge-Stueben on each processor independently, followed by PMIS using the interior C-points as its first independent set) |
| 11 | one-pass Ruge-Stueben coarsening on each processor, no boundary treatment                                                                               |

The default is 6.

### 6.7.11

```
int
HYPRE_ParCSRHybridSetCycleType (HYPRE_Solver solver, int cycle_type)
```

(Optional) Defines the type of cycle. For a V-cycle, set `cycle_type` to 1, for a W-cycle set `cycle_type` to 2. The default is 1.

#### 6.7.12

```
int
HYPRE_ParCSRHybridSetNumSweeps (HYPRE_Solver solver, int
num_sweeps)
```

(Optional) Sets the number of sweeps. On the finest level, the up and the down cycle the number of sweeps are set to `num_sweeps` and on the coarsest level to 1. The default is 1.

#### 6.7.13

```
int
HYPRE_ParCSRHybridSetCycleNumSweeps (HYPRE_Solver solver, int
num_sweeps, int k)
```

(Optional) Sets the number of sweeps at a specified cycle. There are the following options for `k`:

|                    |                       |
|--------------------|-----------------------|
| the down cycle     | if <code>k=1</code>   |
| the up cycle       | if <code>k=2</code>   |
| the coarsest level | if <code>k=3</code> . |

#### 6.7.14

```
int
HYPRE_ParCSRHybridSetRelaxType (HYPRE_Solver solver, int relax_type)
```

(Optional) Defines the smoother to be used. It uses the given smoother on the fine grid, the up and the down cycle and sets the solver on the coarsest level to Gaussian elimination (9). The default is Gauss-Seidel (3).

There are the following options for `relax_type`:

|   |                                                                        |
|---|------------------------------------------------------------------------|
| 0 | Jacobi                                                                 |
| 1 | Gauss-Seidel, sequential (very slow!)                                  |
| 2 | Gauss-Seidel, interior points in parallel, boundary sequential (slow!) |
| 3 | hybrid Gauss-Seidel or SOR, forward solve                              |
| 4 | hybrid Gauss-Seidel or SOR, backward solve                             |
| 5 | hybrid chaotic Gauss-Seidel (works only with OpenMP)                   |
| 6 | hybrid symmetric Gauss-Seidel or SSOR                                  |
| 9 | Gaussian elimination (only on coarsest level)                          |

**6.7.15**

```
int
HYPRE_ParCSRHybridSetCycleRelaxType (HYPRE_Solver solver, int
relax_type, int k)
```

(Optional) Defines the smoother at a given cycle. For options of `relax_type` see description of `HYPRE_BoomerAMGSetRelaxType`). Options for `k` are

|                    |         |
|--------------------|---------|
| the down cycle     | if k=1  |
| the up cycle       | if k=2  |
| the coarsest level | if k=3. |

**6.7.16**

```
int
HYPRE_ParCSRHybridSetRelaxOrder (HYPRE_Solver solver, int
relax_order)
```

(Optional) Defines in which order the points are relaxed. There are the following options for `relax_order`:

|   |                                                                                                                                                                                                                                                                                                                  |
|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0 | the points are relaxed in natural or lexicographic order on each processor                                                                                                                                                                                                                                       |
| 1 | CF-relaxation is used, i.e on the fine grid and the down cycle the coarse points are relaxed first, followed by the fine points; on the up cycle the F-points are relaxed first, followed by the C-points. On the coarsest level, if an iterative scheme is used, the points are relaxed in lexicographic order. |

The default is 1 (CF-relaxation).

**6.7.17**

```
int
HYPRE_ParCSRHybridSetRelaxWt (HYPRE_Solver solver, double relax_wt)
```

(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR on all levels.

|                   |                                                                                                                                                                                                                                      |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| relax_weight > 0  | this assigns the given relaxation weight on all levels                                                                                                                                                                               |
| relax_weight = 0  | the weight is determined on each level with the estimate $\frac{3}{4\ D^{-1/2}AD^{-1/2}\ }$ ,                                                                                                                                        |
| relax_weight = -k | where $D$ is the diagonal matrix of $A$ (this should only be used with Jacobi)<br>the relaxation weight is determined with at most $k$ CG steps on each level<br>(this should only be used for symmetric positive definite problems) |

The default is 1.

#### 6.7.18

```
int
HYPRE_ParCSRHybridSetLevelRelaxWt (HYPRE_Solver solver, double
relax_wt, int level)
```

(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR on the user defined level. Note that the finest level is denoted 0, the next coarser level 1, etc. For nonpositive relax\_weight, the parameter is determined on the given level as described for HYPRE\_BoomerAMGSetRelaxWt. The default is 1.

#### 6.7.19

```
int
HYPRE_ParCSRHybridSetOuterWt (HYPRE_Solver solver, double outer_wt
)
```

(Optional) Defines the outer relaxation weight for hybrid SOR and SSOR on all levels.

|            |                                                                                                                                                                               |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| omega > 0  | this assigns the same outer relaxation weight omega on each level                                                                                                             |
| omega = -k | an outer relaxation weight is determined with at most $k$ CG steps on each level<br>(this only makes sense for symmetric positive definite problems and smoothers, e.g. SSOR) |

The default is 1.

#### 6.7.20

```
int
HYPRE_ParCSRHybridSetLevelOuterWt (HYPRE_Solver solver, double
outer_wt, int level)
```

(Optional) Defines the outer relaxation weight for hybrid SOR or SSOR on the user defined level. Note that the finest level is denoted 0, the next coarser level 1, etc. For nonpositive omega, the parameter is determined on the given level as described for HYPRE\_BoomerAMGSetOuterWt. The default is 1.

#### 6.7.21

```
int
HYPRE_ParCSRHybridSetAggNumLevels (HYPRE_Solver solver, int
agg_num_levels)
```

(Optional) Defines the number of levels of aggressive coarsening, starting with the finest level. The default is 0, i.e. no aggressive coarsening.

#### 6.7.22

```
int
HYPRE_ParCSRHybridSetNumPaths (HYPRE_Solver solver, int num_paths
)
```

(Optional) Defines the degree of aggressive coarsening. The default is 1, which leads to the most aggressive coarsening. Setting num\_paths to 2 will increase complexity somewhat, but can lead to better convergence.\*

#### 6.7.23

```
int
HYPRE_ParCSRHybridSetNumFunctions (HYPRE_Solver solver, int
num_functions)
```

(Optional) Sets the size of the system of PDEs, if using the systems version. The default is 1.

**6.7.24**

```
int
HYPRE_ParCSRHybridSetDofFunc (HYPRE_Solver solver, int *dof_func)
```

(Optional) Sets the mapping that assigns the function to each variable, if using the systems version. If no assignment is made and the number of functions is  $k > 1$ , the mapping generated is  $(0,1,\dots,k-1,0,1,\dots,k-1,\dots)$ .

**6.7.25**

```
int HYPRE_ParCSRHybridSetNodal (HYPRE_Solver solver, int nodal)
```

(Optional) Sets whether to use the nodal systems version. The default is 0 (the unknown based approach).

**6.8****ParCSR PCG Solver****Names**

```
int
HYPRE_ParCSRPCGCreate (MPI_Comm comm, HYPRE_Solver *solver)
 Create a solver object
```

```
int
HYPRE_ParCSRPCGDestroy (HYPRE_Solver solver)
 Destroy a solver object
```

```
int
HYPRE_ParCSRPCGSetup (HYPRE_Solver solver,
 HYPRE_ParCSRMatrix A,
 HYPRE_ParVector b, HYPRE_ParVector x)
```

```
int
HYPRE_ParCSRPCGSolve (HYPRE_Solver solver,
 HYPRE_ParCSRMatrix A,
 HYPRE_ParVector b, HYPRE_ParVector x)
 Solve the system
```

```
int
HYPRE_ParCSRPCGSetTol (HYPRE_Solver solver, double tol)
 (Optional) Set the convergence tolerance
```

```
int
```



---

**HYPRE\_ParCSRPCGSetMaxIter** (HYPRE\_Solver solver, int max\_iter)  
*(Optional) Set maximum number of iterations*

int

**HYPRE\_ParCSRPCGSetTwoNorm** (HYPRE\_Solver solver, int two\_norm)  
*(Optional) Use the two-norm in stopping criteria*

int

**HYPRE\_ParCSRPCGSetRelChange** (HYPRE\_Solver solver, int rel\_change)  
*(Optional) Additionally require that the relative difference in successive iterates be small*

int

**HYPRE\_ParCSRPCGSetPrecond** (HYPRE\_Solver solver,  
HYPRE\_PtrToParSolverFcn precondition,  
HYPRE\_PtrToParSolverFcn  
precond\_setup,  
HYPRE\_Solver precondition\_solver)  
*(Optional) Set the preconditioner to use*

int

**HYPRE\_ParCSRPCGGetPrecond** (HYPRE\_Solver solver,  
HYPRE\_Solver \*precond\_data)

int

**HYPRE\_ParCSRPCGSetLogging** (HYPRE\_Solver solver, int logging)  
*(Optional) Set the amount of logging to do*

int

**HYPRE\_ParCSRPCGSetPrintLevel** (HYPRE\_Solver solver, int print\_level)  
*(Optional) Set the print level*

int

**HYPRE\_ParCSRPCGGetNumIterations** (HYPRE\_Solver solver,  
int \*num\_iterations)  
*Return the number of iterations taken*

int

**HYPRE\_ParCSRPCGGetFinalRelativeResidualNorm** (HYPRE\_Solver  
solver,  
double \*norm)  
*Return the norm of the final relative residual*

int

**HYPRE\_ParCSRDiagScaleSetup** (HYPRE\_Solver solver,  
HYPRE\_ParCSRMatrix A,  
HYPRE\_ParVector y,  
HYPRE\_ParVector x)  
*Setup routine for diagonal preconditioning*

int

**HYPRE\_ParCSRDiagScale** (HYPRE\_Solver solver,  
HYPRE\_ParCSRMatrix HA,  
HYPRE\_ParVector Hy, HYPRE\_ParVector Hx)  
*Solve routine for diagonal preconditioning*

## 6.9

**ParCSR GMRES Solver****Names**

int  
**HYPRE\_ParCSRGMRESCreate** (MPI\_Comm comm,  
HYPRE\_Solver \*solver)  
*Create a solver object*

int  
**HYPRE\_ParCSRGMRESDestroy** (HYPRE\_Solver solver)  
*Destroy a solver object*

int  
**HYPRE\_ParCSRGMRESSetup** (HYPRE\_Solver solver,  
HYPRE\_ParCSRMatrix A,  
HYPRE\_ParVector b,  
HYPRE\_ParVector x)

int  
**HYPRE\_ParCSRGMRESSolve** (HYPRE\_Solver solver,  
HYPRE\_ParCSRMatrix A,  
HYPRE\_ParVector b, HYPRE\_ParVector x)  
*Solve the system*

int  
**HYPRE\_ParCSRGMRESSetKDim** (HYPRE\_Solver solver, int k\_dim)  
*(Optional) Set the maximum size of the Krylov space*

int  
**HYPRE\_ParCSRGMRESSetTol** (HYPRE\_Solver solver, double tol)  
*(Optional) Set the convergence tolerance*

int  
**HYPRE\_ParCSRGMRESSetMaxIter** (HYPRE\_Solver solver, int max\_iter)  
*(Optional) Set maximum number of iterations*

int  
**HYPRE\_ParCSRGMRESSetPrecond** (HYPRE\_Solver solver,  
HYPRE\_PtrToParSolverFcn precondition,  
HYPRE\_PtrToParSolverFcn  
precond\_setup,  
HYPRE\_Solver precondition\_solver)  
*(Optional) Set the preconditioner to use*

int  
**HYPRE\_ParCSRGMRESGetPrecond** (HYPRE\_Solver solver,  
HYPRE\_Solver \*precond\_data)

int  
**HYPRE\_ParCSRGMRESSetLogging** (HYPRE\_Solver solver, int logging)  
*(Optional) Set the amount of logging to do*

int

**HYPRE\_ParCSRGMRESSetPrintLevel** (HYPRE\_Solver solver,  
int print\_level)  
*(Optional) Set print level*

int  
**HYPRE\_ParCSRGMRESGetNumIterations** (HYPRE\_Solver solver,  
int \*num\_iterations)  
*Return the number of iterations taken*

int  
**HYPRE\_ParCSRGMRESGetFinalRelativeResidualNorm** (HYPRE\_Solver  
solver,  
double \*norm)  
*Return the norm of the final relative residual*

## 6.10

## ParCSR BiCGSTAB Solver

## Names

int  
**HYPRE\_ParCSRBiCGSTABCreate** (MPI\_Comm comm,  
HYPRE\_Solver \*solver)  
*Create a solver object*

int  
**HYPRE\_ParCSRBiCGSTABDestroy** (HYPRE\_Solver solver)  
*Destroy a solver object*

int  
**HYPRE\_ParCSRBiCGSTABSetup** (HYPRE\_Solver solver,  
HYPRE\_ParCSRMatrix A,  
HYPRE\_ParVector b,  
HYPRE\_ParVector x)  
*Set up BiCGSTAB solver*

int  
**HYPRE\_ParCSRBiCGSTABSolve** (HYPRE\_Solver solver,  
HYPRE\_ParCSRMatrix A,  
HYPRE\_ParVector b,  
HYPRE\_ParVector x)  
*Solve the linear system*

## 6.10.1

int  
**HYPRE\_ParCSRBiCGSTABSetTol** (HYPRE\_Solver solver, double tol)  
*(Optional) Set the convergence tolerance (default is 1 .....* 132

int  
**HYPRE\_ParCSRBiCGSTABSetMinIter** (HYPRE\_Solver solver,  
int min\_iter)  
*(Optional) Set the minimal number of iterations (default: 0)*

int

- HYPRE\_ParCSRBiCGSTABSetMaxIter** (HYPRE\_Solver solver,  
int max\_iter)  
*(Optional) Set the maximal number of iterations allowed (default: 1000)*
- 6.10.2 int  
**HYPRE\_ParCSRBiCGSTABSetStopCrit** (HYPRE\_Solver solver,  
int stop\_crit)  
*(Optional) If stop\_crit = 1, the absolute residual norm is used for the stopping criterion* ..... 133
- int  
**HYPRE\_ParCSRBiCGSTABSetPrecond** (HYPRE\_Solver solver,  
HYPRE\_PtrToParSolverFcn  
precond,  
HYPRE\_PtrToParSolverFcn  
precond\_setup,  
HYPRE\_Solver preconditioner)  
*(Optional) Set the preconditioner*
- int  
**HYPRE\_ParCSRBiCGSTABGetPrecond** (HYPRE\_Solver solver,  
HYPRE\_Solver \*precond\_data)  
*Get the preconditioner object*
- 6.10.3 int  
**HYPRE\_ParCSRBiCGSTABSetLogging** (HYPRE\_Solver solver,  
int logging)  
*(Optional) Set the amount of logging to be done* ..... 133
- 6.10.4 int  
**HYPRE\_ParCSRBiCGSTABSetPrintLevel** (HYPRE\_Solver solver,  
int print\_level)  
*(Optional) Set the desired print level* ..... 133
- int  
**HYPRE\_ParCSRBiCGSTABGetNumIterations** (HYPRE\_Solver solver,  
int \*num\_iterations)  
*Retrieve the number of iterations taken*
- int  
**HYPRE\_ParCSRBiCGSTABGetFinalRelativeResidualNorm** (HYPRE\_Solver  
solver,  
double  
\*norm)  
*Retrieve the final relative residual norm*

**6.10.1**

int **HYPRE\_ParCSRBiCGSTABSetTol** (HYPRE\_Solver solver, double tol)

(Optional) Set the convergence tolerance (default is 1.e-6).

**6.10.2**

```
int
HYPRE_ParCSRBiCGSTABSetStopCrit (HYPRE_Solver solver, int stop_crit)
```

(Optional) If `stop_crit = 1`, the absolute residual norm is used for the stopping criterion. The default is the relative residual norm (`stop_crit = 0`).

**6.10.3**

```
int
HYPRE_ParCSRBiCGSTABSetLogging (HYPRE_Solver solver, int logging)
```

(Optional) Set the amount of logging to be done. The default is 0, i.e. no logging.

**6.10.4**

```
int
HYPRE_ParCSRBiCGSTABSetPrintLevel (HYPRE_Solver solver, int
print_level)
```

(Optional) Set the desired print level. The default is 0, i.e. no printing.

7

extern **Krylov Solvers**

## Names

|     |                        |     |
|-----|------------------------|-----|
| 7.1 | <b>Krylov Solvers</b>  | 134 |
|     | .....                  |     |
| 7.2 | <b>PCG Solver</b>      | 134 |
|     | .....                  |     |
| 7.3 | <b>GMRES Solver</b>    | 136 |
|     | .....                  |     |
| 7.4 | <b>BiCGSTAB Solver</b> | 138 |
|     | .....                  |     |
| 7.5 | <b>CGNR Solver</b>     | 139 |
|     | .....                  |     |

These solvers support many of the matrix/vector storage schemes in hypre. They should be used in conjunction with the storage-specific interfaces, particularly the specific Create() and Destroy() functions.

7.1

**Krylov Solvers**

## Names

```
typedef struct hypre_Solver_struct* HYPRE_Solver
 The solver object

typedef struct hypre_Matrix_struct* HYPRE_Matrix
 The matrix object

typedef struct hypre_Vector_struct* HYPRE_Vector
 The vector object
```

7.2

**PCG Solver**

## Names

|       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 7.2.1 | <p>int<br/> <b>HYPRE_PCGSetup</b> (HYPRE_Solver solver, HYPRE_Matrix A,<br/> HYPRE_Vector b, HYPRE_Vector x)<br/> <i>Prepare to solve the system</i> ..... 136</p> <p>int<br/> <b>HYPRE_PCGSolve</b> (HYPRE_Solver solver, HYPRE_Matrix A,<br/> HYPRE_Vector b, HYPRE_Vector x)<br/> <i>Solve the system</i></p> <p>int<br/> <b>HYPRE_PCGSetTol</b> (HYPRE_Solver solver, double tol)<br/> <i>(Optional) Set the convergence tolerance</i></p> <p>int<br/> <b>HYPRE_PCGSetMaxIter</b> (HYPRE_Solver solver, int max_iter)<br/> <i>(Optional) Set maximum number of iterations</i></p> <p>int<br/> <b>HYPRE_PCGSetTwoNorm</b> (HYPRE_Solver solver, int two_norm)<br/> <i>(Optional) Use the two-norm in stopping criteria</i></p> <p>int<br/> <b>HYPRE_PCGSetRelChange</b> (HYPRE_Solver solver, int rel_change)<br/> <i>(Optional) Additionally require that the relative difference in successive it-<br/> erates be small</i></p> <p>int<br/> <b>HYPRE_PCGSetPrecond</b> (HYPRE_Solver solver,<br/> HYPRE_PtrToSolverFcn precond,<br/> HYPRE_PtrToSolverFcn precond_setup,<br/> HYPRE_Solver precond_solver)<br/> <i>(Optional) Set the preconditioner to use</i></p> <p>int<br/> <b>HYPRE_PCGSetLogging</b> (HYPRE_Solver solver, int logging)<br/> <i>(Optional) Set the amount of logging to do</i></p> <p>int<br/> <b>HYPRE_PCGSetPrintLevel</b> (HYPRE_Solver solver, int level)<br/> <i>(Optional) Set the amount of printing to do to the screen</i></p> <p>int<br/> <b>HYPRE_PCGGetNumIterations</b> (HYPRE_Solver solver, int *num_iterations)<br/> <i>Return the number of iterations taken</i></p> <p>int<br/> <b>HYPRE_PCGGetFinalRelativeResidualNorm</b> (HYPRE_Solver solver,<br/> double *norm)<br/> <i>Return the norm of the final relative residual</i></p> <p>int<br/> <b>HYPRE_PCGGetResidual</b> (HYPRE_Solver solver, void **residual)<br/> <i>Return the residual</i></p> <p>int<br/> <b>HYPRE_PCGGetTol</b> (HYPRE_Solver solver, double *tol)</p> <p>int<br/> <b>HYPRE_PCGGetMaxIter</b> (HYPRE_Solver solver, int *max_iter)</p> <p>int</p> |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```

HYPRE_PCGGetTwoNorm (HYPRE_Solver solver, int *two_norm)
int
HYPRE_PCGGetRelChange (HYPRE_Solver solver, int *rel_change)
int
HYPRE_PCGGetPrecond (HYPRE_Solver solver,
 HYPRE_Solver *precond_data_ptr)
int
HYPRE_PCGGetLogging (HYPRE_Solver solver, int *level)
int
HYPRE_PCGGetPrintLevel (HYPRE_Solver solver, int *level)
int
HYPRE_PCGGetConverged (HYPRE_Solver solver, int *converged)

```

### 7.2.1

```

int
HYPRE_PCGSetup (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector
b, HYPRE_Vector x)

```

Prepare to solve the system. The coefficient data in **b** and **x** is ignored here, but information about the layout of the data may be used.

### 7.3

## GMRES Solver

### Names

```

7.3.1 int
HYPRE_GMRESSetup (HYPRE_Solver solver, HYPRE_Matrix A,
 HYPRE_Vector b, HYPRE_Vector x)
 Prepare to solve the system 138
int
HYPRE_GMRESSolve (HYPRE_Solver solver, HYPRE_Matrix A,
 HYPRE_Vector b, HYPRE_Vector x)
 Solve the system
int
HYPRE_GMRESSetTol (HYPRE_Solver solver, double tol)
 (Optional) Set the convergence tolerance
int

```



---

**HYPRE\_GMRESSetMaxIter** (HYPRE\_Solver solver, int max\_iter)  
*(Optional) Set maximum number of iterations*

int

**HYPRE\_GMRESSetKDim** (HYPRE\_Solver solver, int k\_dim)  
*(Optional) Set the maximum size of the Krylov space*

int

**HYPRE\_GMRESSetRelChange** (HYPRE\_Solver solver, int rel\_change)  
*(Optional) Additionally require that the relative difference in successive iterates be small*

int

**HYPRE\_GMRESSetPrecond** (HYPRE\_Solver solver,  
HYPRE\_PtrToSolverFcn precond,  
HYPRE\_PtrToSolverFcn precond\_setup,  
HYPRE\_Solver precond\_solver)  
*(Optional) Set the preconditioner to use*

int

**HYPRE\_GMRESSetLogging** (HYPRE\_Solver solver, int logging)  
*(Optional) Set the amount of logging to do*

int

**HYPRE\_GMRESSetPrintLevel** (HYPRE\_Solver solver, int level)  
*(Optional) Set the amount of printing to do to the screen*

int

**HYPRE\_GMRESGetNumIterations** (HYPRE\_Solver solver,  
int \*num\_iterations)  
*Return the number of iterations taken*

int

**HYPRE\_GMRESGetFinalRelativeResidualNorm** (HYPRE\_Solver solver,  
double \*norm)  
*Return the norm of the final relative residual*

int

**HYPRE\_GMRESGetResidual** (HYPRE\_Solver solver, void \*\*residual)  
*Return the residual*

int

**HYPRE\_GMRESGetTol** (HYPRE\_Solver solver, double \*tol)

int

**HYPRE\_GMRESGetMaxIter** (HYPRE\_Solver solver, int \*max\_iter)

int

**HYPRE\_GMRESGetKDim** (HYPRE\_Solver solver, int \*k\_dim)

int

**HYPRE\_GMRESGetRelChange** (HYPRE\_Solver solver, int \*rel\_change)

int

**HYPRE\_GMRESGetPrecond** (HYPRE\_Solver solver,  
HYPRE\_Solver \*precond\_data\_ptr)

int

**HYPRE\_GMRESGetLogging** (HYPRE\_Solver solver, int \*level)

int

**HYPRE\_GMRESGetPrintLevel** (HYPRE\_Solver solver, int \*level)

int

**HYPRE\_GMRESGetConverged** (HYPRE\_Solver solver, int \*converged)

### 7.3.1

int

**HYPRE\_GMRESSetup** (HYPRE\_Solver solver, HYPRE\_Matrix A,  
HYPRE\_Vector b, HYPRE\_Vector x)

Prepare to solve the system. The coefficient data in **b** and **x** is ignored here, but information about the layout of the data may be used.

### 7.4

## BiCGSTAB Solver

### Names

#### 7.4.1

int

**HYPRE\_BiCGSTABSetup** (HYPRE\_Solver solver, HYPRE\_Matrix A,  
HYPRE\_Vector b, HYPRE\_Vector x)

*Prepare to solve the system* ..... 139

int

**HYPRE\_BiCGSTABSolve** (HYPRE\_Solver solver, HYPRE\_Matrix A,  
HYPRE\_Vector b, HYPRE\_Vector x)

*Solve the system*

int

**HYPRE\_BiCGSTABSetTol** (HYPRE\_Solver solver, double tol)

*(Optional) Set the convergence tolerance*

int

**HYPRE\_BiCGSTABSetMaxIter** (HYPRE\_Solver solver, int max\_iter)

*(Optional) Set maximum number of iterations*

int

**HYPRE\_BiCGSTABSetPrecond** (HYPRE\_Solver solver,  
HYPRE\_PtrToSolverFcn precond,  
HYPRE\_PtrToSolverFcn precond\_setup,  
HYPRE\_Solver precond\_solver)

*(Optional) Set the preconditioner to use*

int

**HYPRE\_BiCGSTABSetLogging** (HYPRE\_Solver solver, int logging)

*(Optional) Set the amount of logging to do*

int

**HYPRE\_BiCGSTABSetPrintLevel** (HYPRE\_Solver solver, int level)  
*(Optional) Set the amount of printing to do to the screen*

int

**HYPRE\_BiCGSTABGetNumIterations** (HYPRE\_Solver solver,  
 int \*num\_iterations)

*Return the number of iterations taken*

int

**HYPRE\_BiCGSTABGetFinalRelativeResidualNorm** (HYPRE\_Solver  
 solver,  
 double \*norm)

*Return the norm of the final relative residual*

int

**HYPRE\_BiCGSTABGetResidual** (HYPRE\_Solver solver, void \*\*residual)

*Return the residual*

int

**HYPRE\_BiCGSTABGetPrecond** (HYPRE\_Solver solver,  
 HYPRE\_Solver \*precond\_data\_ptr)

#### 7.4.1

int  
**HYPRE\_BiCGSTABSetup** (HYPRE\_Solver solver, HYPRE\_Matrix A,  
 HYPRE\_Vector b, HYPRE\_Vector x)

Prepare to solve the system. The coefficient data in **b** and **x** is ignored here, but information about the layout of the data may be used.

#### 7.5

### CGNR Solver

#### Names

7.5.1 int  
**HYPRE\_CGNRSetup** (HYPRE\_Solver solver, HYPRE\_Matrix A,  
 HYPRE\_Vector b, HYPRE\_Vector x)  
*Prepare to solve the system* ..... 140

int  
**HYPRE\_CGNRsolve** (HYPRE\_Solver solver, HYPRE\_Matrix A,  
 HYPRE\_Vector b, HYPRE\_Vector x)  
*Solve the system*

int

**HYPRE\_CGNRSetTol** (HYPRE\_Solver solver, double tol)  
*(Optional) Set the convergence tolerance*

int  
**HYPRE\_CGNRSetMaxIter** (HYPRE\_Solver solver, int max\_iter)  
*(Optional) Set maximum number of iterations*

int  
**HYPRE\_CGNRSetPrecond** (HYPRE\_Solver solver,  
 HYPRE\_PtrToSolverFcn precondition,  
 HYPRE\_PtrToSolverFcn preconditionT,  
 HYPRE\_PtrToSolverFcn precondition\_setup,  
 HYPRE\_Solver precondition\_solver)  
*(Optional) Set the preconditioner to use*

int  
**HYPRE\_CGNRSetLogging** (HYPRE\_Solver solver, int logging)  
*(Optional) Set the amount of logging to do*

int  
**HYPRE\_CGNRGetNumIterations** (HYPRE\_Solver solver,  
 int \*num\_iterations)  
*Return the number of iterations taken*

int  
**HYPRE\_CGNRGetFinalRelativeResidualNorm** (HYPRE\_Solver solver,  
 double \*norm)  
*Return the norm of the final relative residual*

int  
**HYPRE\_CGNRGetPrecond** (HYPRE\_Solver solver,  
 HYPRE\_Solver \*precond\_data\_ptr)

### 7.5.1

```
int
HYPRE_CGNRSetup (HYPRE_Solver solver, HYPRE_Matrix A,
HYPRE_Vector b, HYPRE_Vector x)
```

Prepare to solve the system. The coefficient data in **b** and **x** is ignored here, but information about the layout of the data may be used.

## HYPRE's Finite Element Interface

### Names

|     |                                   |     |
|-----|-----------------------------------|-----|
| 8.1 | <b>HYPRE FEI functions</b>        | 141 |
| 8.2 | <b>HYPRE FEI Matrix functions</b> | 149 |
| 8.3 | <b>HYPRE FEI Vector functions</b> | 150 |
| 8.4 | <b>Solver parameters</b>          | 152 |

## HYPRE FEI functions

### Names

|       |                                                                                                                                                                                                                                                                                                   |     |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 8.1.1 | int<br><b>HYPRE_FEMeshCreate</b> ( MPI_Comm comm, HYPRE_FEMesh *mesh )<br><i>Finite element interface constructor: this function creates an instantiation of the HYPRE fei class</i>                                                                                                              | 143 |
| 8.1.2 | int<br><b>HYPRE_FEMeshDestroy</b> ( HYPRE_FEMesh mesh )<br><i>Finite element interface destructor: this function destroys the object as well as its internal memory allocations</i>                                                                                                               | 143 |
| 8.1.3 | int<br><b>HYPRE_FEMeshSetFEObject</b> (HYPRE_FEMesh mesh, void *externFEI, void *linSys)<br><i>This function passes the externally-built FEI object (for example, Sandia's implementation of the FEI) as well as corresponding LinearSystemCore object</i>                                        | 144 |
| 8.1.4 | int<br><b>HYPRE_FEMeshParameters</b> (HYPRE_FEMesh mesh, int numParams, char **paramStrings)<br><i>The parameters function is the single most important function to pass solver information (which solver, which preconditioner, tolerance, other solver parameters) to the underlying solver</i> | 144 |
| 8.1.5 | int                                                                                                                                                                                                                                                                                               |     |

- 
- HYPRE\_FEMeshInitFields** ( HYPRE\_FEMesh mesh, int numFields,  
int \*fieldSizes, int \*fieldIDs )  
*Each node or element variable has one or more fields* ..... 144
- 8.1.6 int  
**HYPRE\_FEMeshInitElemBlock** ( HYPRE\_FEMesh mesh, int blockID,  
int nElements, int numNodesPerElement,  
int \*numFieldsPerNode,  
int \*\*nodalFieldIDs,  
int numElemDOFFieldsPerElement,  
int \*elemDOFFieldIDs,  
int interleaveStrategy )  
*The whole finite element mesh can be broken down into a number of element blocks* ..... 145
- 8.1.7 int  
**HYPRE\_FEMeshInitElem** ( HYPRE\_FEMesh mesh, int blockID, int elemID,  
int \*elemConn )  
*This function initializes element connectivity (that is, the node identifiers associated with the current element) given an element block identifier and the element identifier with the element block* ..... 145
- 8.1.8 int  
**HYPRE\_FEMeshInitSharedNodes** ( HYPRE\_FEMesh mesh, int nShared,  
int \*sharedIDs, int \*sharedLeng,  
int \*\*sharedProcs )  
*This function initializes the nodes that are shared between the current processor and its neighbors* ..... 145
- 8.1.9 int  
**HYPRE\_FEMeshInitComplete** ( HYPRE\_FEMesh mesh )  
*This function signals to the FEI that the initialization step has been completed* ..... 146
- 8.1.10 int  
**HYPRE\_FEMeshLoadNodeBCs** ( HYPRE\_FEMesh mesh, int numNodes,  
int \*nodeIDs, int fieldID, double \*\*alpha,  
double \*\*beta, double \*\*gamma )  
*This function loads the nodal boundary conditions* ..... 146
- 8.1.11 int  
**HYPRE\_FEMeshSumInElem** ( HYPRE\_FEMesh mesh, int blockID,  
int elemID, int\* elemConn,  
double\*\* elemStiffness, double \*elemLoad,  
int elemFormat )  
*This function adds the element contribution to the global stiffness matrix and also the element load to the right hand side vector* ..... 147
- 8.1.12 int  
**HYPRE\_FEMeshSumInElemMatrix** ( HYPRE\_FEMesh mesh, int blockID,  
int elemID, int\* elemConn,  
double\*\* elemStiffness,  
int elemFormat )  
*This function differs from the sumInElem function in that the right hand load vector is not passed* ..... 147
- 8.1.13 int

- 
- HYPRE\_FEMeshSumInElemRHS** ( HYPRE\_FEMesh mesh, int blockID,  
int elemID, int\* elemConn,  
double\* elemLoad )  
*This function adds the element load to the right hand side vector* ..... 147
- 8.1.14 int  
**HYPRE\_FEMeshLoadComplete** ( HYPRE\_FEMesh mesh )  
*This function signals to the FEI that the loading phase has been completed* 148
- 8.1.15 int  
**HYPRE\_FEMeshSolve** ( HYPRE\_FEMesh mesh )  
*This function tells the FEI to solve the linear system* ..... 148
- 8.1.16 int  
**HYPRE\_FEMeshGetBlockNodeIDList** ( HYPRE\_FEMesh mesh,  
int blockID, int numNodes,  
int \*nodeIDList )  
*This function returns the node identifiers given the element block* ..... 148
- 8.1.17 int  
**HYPRE\_FEMeshGetBlockNodeSolution** ( HYPRE\_FEMesh mesh,  
int blockID, int numNodes,  
int \*nodeIDList, int \*solnOffsets,  
double \*solnValues )  
*This function returns the nodal solutions given the element block number* 148

### 8.1.1

```
int HYPRE_FEMeshCreate (MPI_Comm comm, HYPRE_FEMesh *mesh)
```

**Parameters:**                    **comm** — - an MPI communicator  
                                  **mesh** — - upon return, contains a pointer to the finite element mesh

### 8.1.2

```
int HYPRE_FEMeshDestroy (HYPRE_FEMesh mesh)
```

**Parameters:**                    **mesh** — - a pointer to the finite element mesh

## 8.1.3

```
int
HYPRE_FEMeshSetFEObject (HYPRE_FEMesh mesh, void *externFEI, void
*linSys)
```

This function passes the externally-built FEI object (for example, Sandia's implementation of the FEI) as well as corresponding LinearSystemCore object.

**Parameters:**

- `mesh` — - a pointer to the finite element mesh
- `externFEI` — - a pointer to the externally built finite element object
- `linSys` — - a pointer to the HYPRE linear system solver object built using the `HYPRE_base_create` function.

## 8.1.4

```
int
HYPRE_FEMeshParameters (HYPRE_FEMesh mesh, int numParams, char
**paramStrings)
```

**Parameters:**

- `mesh` — - a pointer to the finite element mesh
- `numParams` — - number of command strings
- `paramStrings` — - the command strings

## 8.1.5

```
int
HYPRE_FEMeshInitFields (HYPRE_FEMesh mesh, int numFields, int
*fieldSizes, int *fieldIDs)
```

Each node or element variable has one or more fields. The field information can be set up using this function.

**Parameters:**

- `mesh` — - a pointer to the finite element mesh
- `numFields` — - total number of fields for all variable types
- `fieldSizes` — - degree of freedom for each field type
- `fieldIDs` — - a list of field identifiers



## 8.1.6

```
int
HYPRE_FEMeshInitElemBlock (HYPRE_FEMesh mesh, int blockID, int
nElements, int numNodesPerElement, int *numFieldsPerNode, int **nodalFieldIDs,
int numElemDOFFieldsPerElement, int *elemDOFFieldIDs, int interleaveStrategy)
```

The whole finite element mesh can be broken down into a number of element blocks. The attributes for each element block are: an identifier, number of elements, number of nodes per elements, the number of fields in each element node, etc.

**Parameters:**

**mesh** — - a pointer to the finite element mesh  
**blockID** — - element block identifier  
**nElements** — - number of element in this block  
**numNodesPerElement** — - number of nodes per element in this block  
**numFieldsPerNode** — - number of fields for each node  
**nodalFieldIDs** — - field identifiers for the nodal unknowns  
**numElemDOFFieldsPerElement** — - number of fields for the element  
**elemDOFFieldIDs** — - field identifier for the element unknowns  
**interleaveStratety** — - indicates how unknowns are ordered

## 8.1.7

```
int
HYPRE_FEMeshInitElem (HYPRE_FEMesh mesh, int blockID, int elemID, int
*elemConn)
```

**Parameters:**

**mesh** — - a pointer to the finite element mesh  
**blockID** — - element block identifier  
**elemID** — - element identifier  
**elemConn** — - a list of node identifiers for this element

## 8.1.8

```
int
HYPRE_FEMeshInitSharedNodes (HYPRE_FEMesh mesh, int nShared, int
*sharedIDs, int *sharedLeng, int **sharedProcs)
```

This function initializes the nodes that are shared between the current processor and its neighbors. The FEI will decide a unique processor each shared node will be assigned to.

**Parameters:**

- `mesh` — - a pointer to the finite element mesh
- `nShared` — - number of shared nodes
- `sharedIDs` — - shared node identifiers
- `sharedLengs` — - the number of processors each node shares with
- `sharedProcs` — - the processor identifiers each node shares with

### 8.1.9

```
int HYPRE_FEMeshInitComplete (HYPRE_FEMesh mesh)
```

This function signals to the FEI that the initialization step has been completed. The loading step will follow.

**Parameters:**

- `mesh` — - a pointer to the finite element mesh

### 8.1.10

```
int
HYPRE_FEMeshLoadNodeBCs (HYPRE_FEMesh mesh, int numNodes, int
*nodeIDs, int fieldID, double **alpha, double **beta, double **gamma)
```

This function loads the nodal boundary conditions. The boundary conditions allowed are of the robin type.

**Parameters:**

- `mesh` — - a pointer to the finite element mesh
- `nNodes` — - number of nodes boundary conditions are imposed
- `nodeIDs` — - nodal identifiers
- `fieldID` — - field identifier with nodes where BC are imposed
- `alpha` — - the multipliers for the field
- `beta` — - the multipliers for the normal derivative of the field
- `gamma` — - the boundary values on the right hand side of the equations

## 8.1.11

```
int
HYPRE_FEMeshSumInElem (HYPRE_FEMesh mesh, int blockID, int elemID,
int* elemConn, double** elemStiffness, double *elemLoad, int elemFormat)
```

**Parameters:**

- `mesh` — - a pointer to the finite element mesh
- `BlockID` — - element block identifier
- `elemID` — - element identifier
- `elemConn` — - a list of node identifiers for this element
- `elemStiff` — - element stiffness matrix
- `elemLoad` — - right hand side (load) for this element
- `elemFormat` — - the format the unknowns are passed in

## 8.1.12

```
int
HYPRE_FEMeshSumInElemMatrix (HYPRE_FEMesh mesh, int blockID, int
elemID, int* elemConn, double** elemStiffness, int elemFormat)
```

**Parameters:**

- `mesh` — - a pointer to the finite element mesh
- `blockID` — - element block identifier
- `elemID` — - element identifier
- `elemConn` — - a list of node identifiers for this element
- `elemStiff` — - element stiffness matrix
- `elemFormat` — - the format the unknowns are passed in

## 8.1.13

```
int
HYPRE_FEMeshSumInElemRHS (HYPRE_FEMesh mesh, int blockID, int
elemID, int* elemConn, double* elemLoad)
```

**Parameters:**

- `mesh` — - a pointer to the finite element mesh
- `blockID` — - element block identifier
- `elemID` — - element identifier
- `elemConn` — - a list of node identifiers for this element
- `elemLoad` — - right hand side (load) for this element

**8.1.14**

```
int HYPRE_FEMeshLoadComplete (HYPRE_FEMesh mesh)
```

**Parameters:**                    `mesh` — - a pointer to the finite element mesh

**8.1.15**

```
int HYPRE_FEMeshSolve (HYPRE_FEMesh mesh)
```

**Parameters:**                    `mesh` — - a pointer to the finite element mesh

**8.1.16**

```
int
HYPRE_FEMeshGetBlockNodeIDList (HYPRE_FEMesh mesh, int blockID,
int numNodes, int *nodeIDList)
```

**Parameters:**                    `mesh` — - a pointer to the finite element mesh  
                                  `blockID` — - element block identifier  
                                  `numNodes` — - the number of nodes  
                                  `nodeIDList` — - the node identifiers

**8.1.17**

```
int
HYPRE_FEMeshGetBlockNodeSolution (HYPRE_FEMesh mesh, int
blockID, int numNodes, int *nodeIDList, int *solnOffsets, double *solnValues)
```

**Parameters:**

- `mesh` — - a pointer to the finite element mesh
- `blockID` — - element block identifier
- `numNodes` — - the number of nodes
- `nodeIDList` — - the node identifiers
- `solnOffsets` — - the equation number for each nodal solution
- `solnValues` — - the nodal solution values

## 8.2

## HYPRE FEI Matrix functions

**Names**

|       |                                                                                                                                                                          |     |
|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 8.2.1 | int<br><b>HYPRE_FEMatrixCreate</b> ( MPI_Comm comm, HYPRE_FEMesh mesh,<br>HYPRE_FEMatrix *matrix )<br><i>Finite element matrix constructor</i> .....                     | 149 |
| 8.2.2 | int<br><b>HYPRE_FEMatrixDestroy</b> ( HYPRE_FEMatrix matrix )<br><i>Finite element matrix destructor</i> .....                                                           | 150 |
| 8.2.3 | int<br><b>HYPRE_FEMatrixGetObject</b> ( HYPRE_FEMatrix matrix, void **object )<br><i>This function gets the underlying HYPRE parser matrix from the FE mesh</i><br>..... | 150 |

## 8.2.1

```
int
HYPRE_FEMatrixCreate (MPI_Comm comm, HYPRE_FEMesh mesh,
 HYPRE_FEMatrix *matrix)
```

**Parameters:**

- `comm` — - an MPI communicator
- `mesh` — - a pointer to the finite element mesh
- `matrix` — - upon return, contains a pointer to the FE matrix

## 8.2.2

```
int HYPRE_FEMatrixDestroy (HYPRE_FEMatrix matrix)
```

**Parameters:** `matrix` — - a pointer to the FE matrix

## 8.2.3

```
int HYPRE_FEMatrixGetObject (HYPRE_FEMatrix matrix, void **object)
```

**Parameters:** `matrix` — - a pointer to the FE matrix  
`object` — - a pointer to the HYPRE parcsr matrix

## 8.3

### HYPRE FEI Vector functions

#### Names

|       |                                                                                                                                                           |     |
|-------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 8.3.1 | int<br><b>HYPRE_FEVectorCreate</b> ( MPI_Comm comm, HYPRE_FEMesh mesh,<br>HYPRE_FEVector *vector)<br><i>Finite element vector constructor</i> .....       | 151 |
| 8.3.2 | int<br><b>HYPRE_FEVectorDestroy</b> ( HYPRE_FEVector vector )<br><i>Finite element vector destructor</i> .....                                            | 151 |
| 8.3.3 | int<br><b>HYPRE_FEVectorGetRHS</b> ( HYPRE_FEVector vector, void **object )<br><i>This function gets the underlying RHS vector from the FE mesh</i> ..... | 151 |
| 8.3.4 | int<br><b>HYPRE_FEVectorSetSol</b> ( HYPRE_FEVector vector, void *object )<br><i>This function gives the solution vector to the FE mesh</i> .....         | 151 |

## 8.3.1

```
int
HYPRE_FEVectorCreate (MPI_Comm comm, HYPRE_FEMesh mesh,
HYPRE_FEVector *vector)
```

**Parameters:**

- `comm` — - an MPI communicator
- `mesh` — - a pointer to the finite element mesh
- `vector` — - upon return, contains a pointer to the FE vector

## 8.3.2

```
int HYPRE_FEVectorDestroy (HYPRE_FEVector vector)
```

**Parameters:**

- `vector` — - a pointer to the FE vector

## 8.3.3

```
int HYPRE_FEVectorGetRHS (HYPRE_FEVector vector, void **object)
```

**Parameters:**

- `vector` — - a pointer to the FE vector
- `object` — - upon return, points to the RHS vector

## 8.3.4

```
int HYPRE_FEVectorSetSol (HYPRE_FEVector vector, void *object)
```

**Parameters:**

- `vector` — - a pointer to the FE vector
- `object` — - points to the solution vector

## 8.4

## Solver parameters

### Names

|       |                                                                                                                         |     |
|-------|-------------------------------------------------------------------------------------------------------------------------|-----|
| 8.4.1 | <b>Preconditioners and Solvers</b><br><i>Here the various options for solvers and preconditioners are defined</i> ..... | 152 |
| 8.4.2 | <b>BoomerAMG</b><br><i>Parameter options for the algebraic multigrid preconditioner BoomerAMG</i>                       | 153 |
| 8.4.3 | <b>MLI</b><br><i>Parameter options for the smoothed aggregation preconditioner MLI</i> .....                            | 154 |
| 8.4.4 | <b>Various</b><br><i>Parameter options for ILUT, ParaSails and polynomial preconditioners are defined</i> .....         | 155 |
| 8.4.5 | <b>Matrix Reduction</b><br><i>Parameters which define different reduction modes</i> .....                               | 155 |
| 8.4.6 | <b>Performance Tuning and Diagnostics</b><br><i>Parameters control diagnostic information, memory use, etc</i> .....    | 156 |
| 8.4.7 | <b>Miscellaneous</b><br><i>Parameters that are helpful for finite element information</i> .....                         | 156 |

## 8.4.1

### Preconditioners and Solvers

Here the various options for solvers and preconditioners are defined.

**solver xxx** where xxx specifies one of `cg`, `gmres`, `fgmres`, `bicgs`, `bicgstab`, `tfqmr`, `symqmr`, `superlu`, or `superlux`. The default is `gmres`. The solver type can be followed by `override` to specify its priority when multiple solvers are declared at random order.

**preconditioner xxx** where xxx is one of `diagonal`, `pilut`, `euclid`, `parasails`, `boomeramg`, `poly`, or `mli`. The default is `diagonal`. Another option for xxx is `reuse` which allows the preconditioner to be reused (this should only be set after a preconditioner has been set up already). The preconditioner type can be followed by `override` to specify its priority when multiple preconditioners are declared at random order.

**maxIterations xxx** where xxx is an integer specifying the maximum number of iterations permitted for the iterative solvers. The default value is 1000.

**tolerance xxx** where xxx is a floating point number specifying the termination criterion for the iterative solvers. The default value is 1.0E-6.

**gmresDim xxx** where xxx is an integer specifying the value of m in restarted GMRES(m). The default value is 100.



**stopCrit xxx** where xxx is one of **absolute** or **relative** stopping criterion.

**superluOrdering xxx** - where xxx specifies one of **natural** or **mmd** (minimum degree ordering). This ordering is used to minimize the number of nonzeros generated in the LU decomposition. The default is natural ordering.

**superluScale xxx** where xxx specifies one of **y** (perform row and column scalings before decomposition) or **n**. The default is no scaling.

#### 8.4.2

### BoomerAMG

Parameter options for the algebraic multigrid preconditioner BoomerAMG.

**amgMaxLevels xxx** where xxx is an integer specifying the maximum number of levels to be used for the grid hierarchy.

**amgCoarsenType xxx** where xxx specifies one of **falgout** or **ruge**, or **default** (CLJP) coarsening for BOOMERAMG.

**amgMeasureType xxx** where xxx specifies one of **local** or **global**. This parameter affects how coarsening is performed in parallel.

**amgRelaxType xxx** where xxx is one of **jacobi** (Damped Jacobi), **gs-slow** (sequential Gauss-Seidel), **gs-fast** (Gauss-Seidel on interior nodes), or **hybrid**. The default is **hybrid**.

**amgNumSweeps xxx** where xxx is an integer specifying the number of pre- and post-smoothing at each level of BOOMERAMG. The default is two pre- and two post-smoothings.

**amgRelaxWeight xxx** where xxx is a floating point number between 0 and 1 specifying the damping factor for BOOMERAMG's damped Jacobi and GS smoothers. The default value is 1.0.

**amgRelaxOmega xxx** where xxx is a floating point number between 0 and 1 specifying the damping factor for BOOMERAMG's hybrid smoother for multiple processors. The default value is 1.0.

**amgStrongThreshold xxx** where xxx is a floating point number between 0 and 1 specifying the threshold used to determine strong coupling in BOOMERAMG's coarsening. The default value is 0.25.

**amgSystemSize xxx** where xxx is the degree of freedom per node.

**amgMaxLevels xxx** where xxx is an integer specifying the maximum number of iterations to be used during the solve phase.

**amgUseGSMG** - tells BOOMERAMG to use a different coarsening called GSMG.

**amgGSMGNumSamples** where xxx is the number of samples to generate to determine how to coarsen for GSMG.

## 8.4.3

## MLI

Parameter options for the smoothed aggregation preconditioner MLI.

**outputLevel xxx** where xxx is the output level for diagnostics.

**method xxx** where xxx is either AMGSA (default), AMGSAe, to indicate which MLI algorithm is to be used.

**numLevels xxx** where xxx is the maximum number of levels (default=30) used.

**maxIterations xxx** where xxx is the maximum number of iterations (default = 1 as preconditioner).

**cycleType xxx** where xxx is either 'V' or 'W' cycle (default = 'V').

**strengthThreshold xxx** strength threshold for coarsening (default = 0).

**smoother xxx** where xxx is either Jacobi, BJacobi, GS, SGS, HSGS (SSOR,default), BSGS, ParaSails, MLS, CGJacobi, CGBJacobi, or Chebyshev.

**numSweeps xxx** where xxx is the number of smoother sweeps (default = 2).

**coarseSolver xxx** where xxx is one of those in 'smoother' or SuperLU (default).

**minCoarseSize xxx** where xxx is the minimum coarse grid size to control the number of levels used (default = 3000).

**Pweight xxx** where xxx is the relaxation parameter for the prolongation smoother (default 0.0).

**nodeDOF xxx** where xxx is the degree of freedom for each node (default = 1).

**nullSpaceDim xxx** where xxx is the dimension of the null space for the coarse grid (default = 1).

**useNodalCoord xxx** where xxx is either 'on' or 'off' (default) to indicate whether the nodal coordinates are used to generate the initial null space.

**saAMGCalibrationSize xxx** where xxx is the additional null space vectors to be generated via calibration (default = 0).

**numSmoothVecs xxx** where xxx is the number of near null space vectors used to create the prolongation operator (default = 0).

**smoothVecSteps xxx** where xxx is the number of smoothing steps used to generate the smooth vectors (default = 0).

In addition, to use 'AMGSAe', the parameter 'haveSFEI' has to be sent into the FEI using the parameters function (this option is valid only for the Sandia FEI implementation).

## 8.4.4

## Various

Parameter options for ILUT, ParaSails and polynomial preconditioners are defined.

**euclidNlevels xxx** where xxx is a non-negative integer specifying the desired sparsity of the incomplete factors. The default value is 0.

**euclidThreshold xxx** where xxx is a floating point number specifying the threshold used to sparsify the incomplete factors. The default value is 0.0.

**parasailsThreshold xxx** where xxx is a floating point number between 0 and 1 specifying the threshold used to prune small entries in setting up the sparse approximate inverse. The default value is 0.0.

**parasailsNlevels xxx** where xxx is an integer larger than 0 specifying the desired sparsity of the approximate inverse. The default value is 1.

**parasailsFilter xxx** where xxx is a floating point number between 0 and 1 specifying the threshold used to prune small entries in  $A$ . The default value is 0.0.

**parasailsLoadbal xxx** where xxx is a floating point number between 0 and 1 specifying how load balancing has to be done (Edmond, explain please). The default value is 0.0.

**parasailsSymmetric** sets Parasails to take  $A$  as symmetric.

**parasailsUnSymmetric** sets Parasails to take  $A$  as nonsymmetric (default).

**parasailsReuse** sets Parasails to reuse the sparsity pattern of  $A$ .

**polyorder xxx** where xxx is the order of the least-squares polynomial preconditioner.

## 8.4.5

## Matrix Reduction

Parameters which define different reduction modes.

**schurReduction** turns on the Schur reduction mode.

**slideReduction** turns on the slide reduction mode.

**slideReduction2** turns on the slide reduction mode version 2 (see section 2).

**slideReduction3** turns on the slide reduction mode version 3 (see section 2).

## 8.4.6

Performance Tuning and **Diagnostics**

Parameters control diagnostic information, memory use, etc.

**outputLevel xxx** where xxx is an integer specifying the output level. An output level of 1 prints only the solver information such as number of iterations and timings. An output level of 2 prints debug information such as the functions visited and preconditioner information. An output level of 3 or higher prints more debug information such as the matrix and right hand side loaded via the LinearSystemCore functions to the standard output.

**setDebug xxx** where xxx is one of `slideReduction1`, `slideReduction2`, `slideReduction3` (level 1,2,3 diagnostics in the slide surface reduction code), `printMat` (print the original matrix into a file), `printReducedMat` (print the reduced matrix into a file), `printSol` (print the solution into a file), `ddilut` (output diagnostic information for DDilut preconditioner setup), and `amgDebug` (output diagnostic information for AMG).

**optimizeMemory** cleans up the matrix sparsity pattern after the matrix has been loaded. (It has been kept to allow matrix reuse.)

**imposeNoBC** turns off the boundary condition to allow diagnosing the matrix (for example, checking the null space.)

## 8.4.7

**Miscellaneous**

Parameters that are helpful for finite element information.

**AConjugateProjection xxx** where xxx specifies the number of previous solution vectors to keep for the A-conjugate projection. The default is 0 (the projection is off).

**minResProjection xxx** where xxx specifies the number of previous solution vectors to keep for projection. The default is 0 (the projection is off).

**haveFEData** indicates that additional finite element information are available to assist in building more efficient solvers.

**haveSFEI** indicates that the simplified finite element information are available to assist in building more efficient solvers.

# Class Graph