>>> voxel\_data

<vtkmodules.vtkCommonDataModel.vtkImageData(0x0000020EEA923890) at 0x0000020ED9B9BB20>

>>> help(voxel\_data)

Help on vtkImageData object:

class vtkImageData(vtkDataSet)

| vtkImageData - topologically and geometrically regular array of data

|

| Superclass: vtkDataSet

|

| vtkImageData is a data object that is a concrete implementation of

| vtkDataSet. vtkImageData represents a geometric structure that is a

| topological and geometrical regular array of points. Examples include

| volumes (voxel data) and pixmaps. This representation supports images

| up to three dimensions. The image may also be oriented (see the

| DirectionMatrices and related transformation methods). Note however,

| that not all filters support oriented images.

|

| @sa

| vtkImageTransform

|

| Method resolution order:

| vtkImageData

| vtkDataSet

| vtkDataObject

| vtkmodules.vtkCommonCore.vtkObject

| vtkmodules.vtkCommonCore.vtkObjectBase

| builtins.object

|

| Methods defined here:

|

| AllocateScalars(...)

| AllocateScalars(self, dataType:int, numComponents:int) -> None

| C++: virtual void AllocateScalars(int dataType, int numComponents)

| AllocateScalars(self, pipeline\_info:vtkInformation) -> None

| C++: virtual void AllocateScalars(vtkInformation \*pipeline\_info)

|

| Allocate the point scalars for this dataset. The data type

| determines the type of the array (VTK\_FLOAT, VTK\_INT etc.) where

| as numComponents determines its number of components.

|

| ApplyIndexToPhysicalMatrix(...)

| ApplyIndexToPhysicalMatrix(self, source:vtkMatrix4x4) -> None

| C++: void ApplyIndexToPhysicalMatrix(vtkMatrix4x4 \*source)

|

| Set the transformation matrix from the index space to the

| physical space coordinate system of the dataset. The transform is

| a 4 by 4 matrix. The supplied matrix pointer is not stored in the

| the image object but the matrix values are used for updating the

| Origin, Spacing, and DirectionMatrix.

| \sa SetOrigin

| \sa SetSpacing

| \sa SetDirectionMatrix

|

| ApplyPhysicalToIndexMatrix(...)

| ApplyPhysicalToIndexMatrix(self, source:vtkMatrix4x4) -> None

| C++: void ApplyPhysicalToIndexMatrix(vtkMatrix4x4 \*source)

|

| Get the transformation matrix from the physical space to the

| index space coordinate system of the dataset. The transform is a

| 4 by 4 matrix. The supplied matrix pointer is not stored in the

| the image object but the matrix values are used for updating the

| Origin, Spacing, and DirectionMatrix.

| \sa SetOrigin

| \sa SetSpacing

| \sa SetDirectionMatrix

|

| BlankCell(...)

| BlankCell(self, ptId:int) -> None

| C++: virtual void BlankCell(vtkIdType ptId)

| BlankCell(self, i:int, j:int, k:int) -> None

| C++: virtual void BlankCell(int i, int j, int k)

|

| Methods for supporting blanking of cells. Blanking turns on or

| off cells in the structured grid. These methods should be called

| only after the dimensions of the grid are set.

|

| BlankPoint(...)

| BlankPoint(self, ptId:int) -> None

| C++: virtual void BlankPoint(vtkIdType ptId)

| BlankPoint(self, i:int, j:int, k:int) -> None

| C++: virtual void BlankPoint(int i, int j, int k)

|

| Methods for supporting blanking of cells. Blanking turns on or

| off points in the structured grid, and hence the cells connected

| to them. These methods should be called only after the dimensions

| of the grid are set.

|

| ComputeBounds(...)

| ComputeBounds(self) -> None

| C++: void ComputeBounds() override;

|

| Compute the data bounding box from data points. THIS METHOD IS

| NOT THREAD SAFE.

|

| ComputeCellId(...)

| ComputeCellId(self, ijk:[int, int, int]) -> int

| C++: virtual vtkIdType ComputeCellId(int ijk[3])

|

| Given a location in structured coordinates (i-j-k), return the

| cell id.

|

| ComputeIndexToPhysicalMatrix(...)

| ComputeIndexToPhysicalMatrix(origin:(float, float, float),

| spacing:(float, float, float), direction:(float, float, float,

| float, float, float, float, float, float), result:[float,

| float, float, float, float, float, float, float, float, float,

| float, float, float, float, float, float]) -> None

| C++: static void ComputeIndexToPhysicalMatrix(

| double const origin[3], double const spacing[3],

| double const direction[9], double result[16])

|

| ComputeInternalExtent(...)

| ComputeInternalExtent(self, intExt:[int, ...], tgtExt:[int, ...],

| bnds:[int, ...]) -> None

| C++: void ComputeInternalExtent(int \*intExt, int \*tgtExt,

| int \*bnds)

|

| Given how many pixel are required on a side for boundary

| conditions (in bnds), the target extent to traverse, compute the

| internal extent (the extent for this ImageData that does not

| suffer from any boundary conditions) and place it in intExt

|

| ComputePointId(...)

| ComputePointId(self, ijk:[int, int, int]) -> int

| C++: virtual vtkIdType ComputePointId(int ijk[3])

|

| Given a location in structured coordinates (i-j-k), return the

| point id.

|

| ComputeStructuredCoordinates(...)

| ComputeStructuredCoordinates(self, x:(float, float, float),

| ijk:[int, int, int], pcoords:[float, float, float]) -> int

| C++: virtual int ComputeStructuredCoordinates(const double x[3],

| int ijk[3], double pcoords[3])

|

| Convenience function computes the structured coordinates for a

| point x[3]. The voxel is specified by the array ijk[3], and the

| parametric coordinates in the cell are specified with pcoords[3].

| The function returns a 0 if the point x is outside of the volume,

| and a 1 if inside the volume.

|

| CopyAndCastFrom(...)

| CopyAndCastFrom(self, inData:vtkImageData, extent:[int, int, int,

| int, int, int]) -> None

| C++: virtual void CopyAndCastFrom(vtkImageData \*inData,

| int extent[6])

| CopyAndCastFrom(self, inData:vtkImageData, x0:int, x1:int, y0:int,

| y1:int, z0:int, z1:int) -> None

| C++: virtual void CopyAndCastFrom(vtkImageData \*inData, int x0,

| int x1, int y0, int y1, int z0, int z1)

|

| This method is passed a input and output region, and executes the

| filter algorithm to fill the output from the input. It just

| executes a switch statement to call the correct function for the

| regions data types.

|

| CopyInformationFromPipeline(...)

| CopyInformationFromPipeline(self, information:vtkInformation)

| -> None

| C++: void CopyInformationFromPipeline(vtkInformation \*information)

| override;

|

| Override these to handle origin, spacing, scalar type, and scalar

| number of components. See vtkDataObject for details.

|

| CopyInformationToPipeline(...)

| CopyInformationToPipeline(self, information:vtkInformation)

| -> None

| C++: void CopyInformationToPipeline(vtkInformation \*information)

| override;

|

| Copy information from this data object to the pipeline

| information. This is used by the vtkTrivialProducer that is

| created when someone calls SetInputData() to connect the image to

| a pipeline.

|

| CopyStructure(...)

| CopyStructure(self, ds:vtkDataSet) -> None

| C++: void CopyStructure(vtkDataSet \*ds) override;

|

| Copy the geometric and topological structure of an input image

| data object.

|

| Crop(...)

| Crop(self, updateExtent:(int, ...)) -> None

| C++: void Crop(const int \*updateExtent) override;

|

| Reallocates and copies to set the Extent to updateExtent. This is

| used internally when the exact extent is requested, and the

| source generated more than the update extent.

|

| DeepCopy(...)

| DeepCopy(self, src:vtkDataObject) -> None

| C++: void DeepCopy(vtkDataObject \*src) override;

|

| The goal of the method is to copy the complete data from src into

| this object. The implementation is delegated to the differenent

| subclasses. If you want to copy the data up to the array pointers

| only, @see ShallowCopy.

|

| This method deep copy the field data and copy the internal

| structure.

|

| ExtendedNew(...)

| ExtendedNew() -> vtkImageData

| C++: static vtkImageData \*ExtendedNew()

|

| FindAndGetCell(...)

| FindAndGetCell(self, x:[float, float, float], cell:vtkCell,

| cellId:int, tol2:float, subId:int, pcoords:[float, float,

| float], weights:[float, ...]) -> vtkCell

| C++: vtkCell \*FindAndGetCell(double x[3], vtkCell \*cell,

| vtkIdType cellId, double tol2, int &subId, double pcoords[3],

| double \*weights) override;

|

| Locate the cell that contains a point and return the cell. Also

| returns the subcell id, parametric coordinates and weights for

| subsequent interpolation. This method combines the derived class

| methods int FindCell and vtkCell \*GetCell. Derived classes may

| provide a more efficient implementation. See for example

| vtkStructuredPoints. THIS METHOD IS NOT THREAD SAFE.

|

| FindCell(...)

| FindCell(self, x:[float, float, float], cell:vtkCell, cellId:int,

| tol2:float, subId:int, pcoords:[float, float, float],

| weights:[float, ...]) -> int

| C++: vtkIdType FindCell(double x[3], vtkCell \*cell,

| vtkIdType cellId, double tol2, int &subId, double pcoords[3],

| double \*weights) override;

| FindCell(self, x:[float, float, float], cell:vtkCell,

| gencell:vtkGenericCell, cellId:int, tol2:float, subId:int,

| pcoords:[float, float, float], weights:[float, ...]) -> int

| C++: vtkIdType FindCell(double x[3], vtkCell \*cell,

| vtkGenericCell \*gencell, vtkIdType cellId, double tol2,

| int &subId, double pcoords[3], double \*weights) override;

|

| Locate cell based on global coordinate x and tolerance squared.

| If cell and cellId is non-nullptr, then search starts from this

| cell and looks at immediate neighbors. Returns cellId >= 0 if

| inside, < 0 otherwise. The parametric coordinates are provided

| in pcoords[3]. The interpolation weights are returned in

| weights[]. (The number of weights is equal to the number of

| points in the found cell). Tolerance is used to control how close

| the point is to be considered "in" the cell. THIS METHOD IS NOT

| THREAD SAFE.

|

| FindPoint(...)

| FindPoint(self, x:[float, float, float]) -> int

| C++: vtkIdType FindPoint(double x[3]) override;

|

| GetActualMemorySize(...)

| GetActualMemorySize(self) -> int

| C++: unsigned long GetActualMemorySize() override;

|

| Return the actual size of the data in kibibytes (1024 bytes).

| This number is valid only after the pipeline has updated. The

| memory size returned is guaranteed to be greater than or equal to

| the memory required to represent the data (e.g., extra space in

| arrays, etc. are not included in the return value). THIS METHOD

| IS THREAD SAFE.

|

| GetArrayIncrements(...)

| GetArrayIncrements(self, array:vtkDataArray, increments:[int, int,

| int]) -> None

| C++: void GetArrayIncrements(vtkDataArray \*array,

| vtkIdType increments[3])

|

| Since various arrays have different number of components, the

| will have different increments.

|

| GetArrayPointer(...)

| GetArrayPointer(self, array:vtkDataArray, coordinates:[int, int,

| int]) -> Pointer

| C++: void \*GetArrayPointer(vtkDataArray \*array,

| int coordinates[3])

|

| GetArrayPointerForExtent(...)

| GetArrayPointerForExtent(self, array:vtkDataArray, extent:[int,

| int, int, int, int, int]) -> Pointer

| C++: void \*GetArrayPointerForExtent(vtkDataArray \*array,

| int extent[6])

|

| These are convenience methods for getting a pointer from any

| filed array. It is a start at expanding image filters to process

| any array (not just scalars).

|

| GetAxisUpdateExtent(...)

| GetAxisUpdateExtent(self, axis:int, min:int, max:int,

| updateExtent:(int, ...)) -> None

| C++: virtual void GetAxisUpdateExtent(int axis, int &min,

| int &max, const int \*updateExtent)

|

| GetCell(...)

| GetCell(self, cellId:int) -> vtkCell

| C++: vtkCell \*GetCell(vtkIdType cellId) override;

| GetCell(self, i:int, j:int, k:int) -> vtkCell

| C++: vtkCell \*GetCell(int i, int j, int k) override;

| GetCell(self, cellId:int, cell:vtkGenericCell) -> None

| C++: void GetCell(vtkIdType cellId, vtkGenericCell \*cell)

| override;

|

| Get cell with cellId such that: 0 <= cellId < NumberOfCells. The

| returned vtkCell is an object owned by this instance, hence the

| return value must not be deleted by the caller.

|

| @warning Repeat calls to this function for different face ids

| will change

| the data stored in the internal member object whose pointer is

| returned by this function.

|

| @warning THIS METHOD IS NOT THREAD SAFE. For a thread-safe

| version, please use

| void GetCell(vtkIdType cellId, vtkGenericCell\* cell).

|

| GetCellBounds(...)

| GetCellBounds(self, cellId:int, bounds:[float, float, float,

| float, float, float]) -> None

| C++: void GetCellBounds(vtkIdType cellId, double bounds[6])

| override;

|

| Get the bounds of the cell with cellId such that: 0 <= cellId <

| NumberOfCells. A subclass may be able to determine the bounds of

| cell without using an expensive GetCell() method. A default

| implementation is provided that actually uses a GetCell() call.

| This is to ensure the method is available to all datasets.

| Subclasses should override this method to provide an efficient

| implementation. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A

| SINGLE THREAD AND THE DATASET IS NOT MODIFIED

|

| GetCellDims(...)

| GetCellDims(self, cellDims:[int, int, int]) -> None

| C++: void GetCellDims(int cellDims[3])

|

| Given the node dimensions of this grid instance, this method

| computes the node dimensions. The value in each dimension can

| will have a lowest value of "1" such that computing the total

| number of cells can be achieved by simply by

| cellDims[0]\*cellDims[1]\*cellDims[2].

|

| GetCellNeighbors(...)

| GetCellNeighbors(self, cellId:int, ptIds:vtkIdList,

| cellIds:vtkIdList) -> None

| C++: void GetCellNeighbors(vtkIdType cellId, vtkIdList \*ptIds,

| vtkIdList \*cellIds) override;

| GetCellNeighbors(self, cellId:int, ptIds:vtkIdList,

| cellIds:vtkIdList, seedLoc:[int, ...]) -> None

| C++: void GetCellNeighbors(vtkIdType cellId, vtkIdList \*ptIds,

| vtkIdList \*cellIds, int \*seedLoc)

|

| Topological inquiry to get all cells using list of points

| exclusive of cell specified (e.g., cellId). Note that the list

| consists of only cells that use ALL the points provided. THIS

| METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND

| THE DATASET IS NOT MODIFIED

|

| GetCellPoints(...)

| GetCellPoints(self, cellId:int, npts:int, pts:(int, ...),

| ptIds:vtkIdList) -> None

| C++: void GetCellPoints(vtkIdType cellId, vtkIdType &npts,

| vtkIdType const \*&pts, vtkIdList \*ptIds) override;

| GetCellPoints(self, cellId:int, ptIds:vtkIdList) -> None

| C++: void GetCellPoints(vtkIdType cellId, vtkIdList \*ptIds)

| override;

|

| Topological inquiry to get points defining cell.

|

| This function MAY use ptIds, which is an object that is created

| by each thread, to guarantee thread safety.

|

| @warning Subsequent calls to this method may invalidate previous

| call

| results.

|

| THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD

| AND THE DATASET IS NOT MODIFIED

|

| GetCellSize(...)

| GetCellSize(self, cellId:int) -> int

| C++: vtkIdType GetCellSize(vtkIdType cellId) override;

|

| Get the size of cell with cellId such that: 0 <= cellId <

| NumberOfCells. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A

| SINGLE THREAD AND THE DATASET IS NOT MODIFIED

|

| @warning This method MUST be overridden for performance reasons.

| Default implementation is very inefficient.

|

| GetCellType(...)

| GetCellType(self, cellId:int) -> int

| C++: int GetCellType(vtkIdType cellId) override;

|

| Get type of cell with cellId such that: 0 <= cellId <

| NumberOfCells. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A

| SINGLE THREAD AND THE DATASET IS NOT MODIFIED

|

| GetCellTypesArray(...)

| GetCellTypesArray(self) -> vtkConstantArray\_IiE

| C++: vtkConstantArray<int> \*GetCellTypesArray()

|

| Get the array of all cell types in the image data. Each

| single-component integer value is the same. The array is of size

| GetNumberOfCells().

|

| NOTE: the returned object should not be modified.

|

| GetCells(...)

| GetCells(self) -> vtkStructuredCellArray

| C++: vtkStructuredCellArray \*GetCells()

|

| Return the image data connectivity array.

|

| NOTE: the returned object should not be modified.

|

| GetContinuousIncrements(...)

| GetContinuousIncrements(self, extent:[int, int, int, int, int,

| int], incX:int, incY:int, incZ:int) -> None

| C++: virtual void GetContinuousIncrements(int extent[6],

| vtkIdType &incX, vtkIdType &incY, vtkIdType &incZ)

| GetContinuousIncrements(self, scalars:vtkDataArray, extent:[int,

| int, int, int, int, int], incX:int, incY:int, incZ:int)

| -> None

| C++: virtual void GetContinuousIncrements(vtkDataArray \*scalars,

| int extent[6], vtkIdType &incX, vtkIdType &incY,

| vtkIdType &incZ)

|

| Different ways to get the increments for moving around the data.

| incX is always returned with 0. incY is returned with the

| increment needed to move from the end of one X scanline of data

| to the start of the next line. incZ is filled in with the

| increment needed to move from the end of one image to the start

| of the next. The proper way to use these values is to for a loop

| over Z, Y, X, C, incrementing the pointer by 1 after each

| component. When the end of the component is reached, the pointer

| is set to the beginning of the next pixel, thus incX is properly

| set to 0. The first form of GetContinuousIncrements uses the

| active scalar field while the second form allows the scalar array

| to be passed in.

|

| GetData(...)

| GetData(info:vtkInformation) -> vtkImageData

| C++: static vtkImageData \*GetData(vtkInformation \*info)

| GetData(v:vtkInformationVector, i:int=0) -> vtkImageData

| C++: static vtkImageData \*GetData(vtkInformationVector \*v,

| int i=0)

|

| Retrieve an instance of this class from an information object.

|

| GetDataDescription(...)

| GetDataDescription(self) -> int

| C++: virtual int GetDataDescription()

|

| Get the data description of the image data.

|

| GetDataDimension(...)

| GetDataDimension(self) -> int

| C++: virtual int GetDataDimension()

|

| Return the dimensionality of the data.

|

| GetDataObjectType(...)

| GetDataObjectType(self) -> int

| C++: int GetDataObjectType() override;

|

| Return what type of dataset this is.

|

| GetDimensions(...)

| GetDimensions(self) -> (int, int, int)

| C++: virtual int \*GetDimensions()

| GetDimensions(self, dims:[int, int, int]) -> None

| C++: virtual void GetDimensions(int dims[3])

| GetDimensions(self, dims:[int, int, int]) -> None

| C++: virtual void GetDimensions(vtkIdType dims[3])

|

| Get dimensions of this structured points dataset. It is the

| number of points on each axis. Dimensions are computed from

| Extents during this call.

| \warning Non thread-safe, use second signature if you want it to

| be.

|

| GetDirectionMatrix(...)

| GetDirectionMatrix(self) -> vtkMatrix3x3

| C++: virtual vtkMatrix3x3 \*GetDirectionMatrix()

|

| Set/Get the direction transform of the dataset. The direction

| matrix is a 3x3 transformation matrix supporting scaling and

| rotation.

|

| GetExtent(...)

| GetExtent(self) -> (int, int, int, int, int, int)

| C++: virtual int \*GetExtent()

|

| GetExtentType(...)

| GetExtentType(self) -> int

| C++: int GetExtentType() override;

|

| The extent type is a 3D extent

|

| GetIncrements(...)

| GetIncrements(self) -> (int, int, int)

| C++: virtual vtkIdType \*GetIncrements()

| GetIncrements(self, incX:int, incY:int, incZ:int) -> None

| C++: virtual void GetIncrements(vtkIdType &incX, vtkIdType &incY,

| vtkIdType &incZ)

| GetIncrements(self, inc:[int, int, int]) -> None

| C++: virtual void GetIncrements(vtkIdType inc[3])

| GetIncrements(self, scalars:vtkDataArray) -> (int, int, int)

| C++: virtual vtkIdType \*GetIncrements(vtkDataArray \*scalars)

| GetIncrements(self, scalars:vtkDataArray, incX:int, incY:int,

| incZ:int) -> None

| C++: virtual void GetIncrements(vtkDataArray \*scalars,

| vtkIdType &incX, vtkIdType &incY, vtkIdType &incZ)

| GetIncrements(self, scalars:vtkDataArray, inc:[int, int, int])

| -> None

| C++: virtual void GetIncrements(vtkDataArray \*scalars,

| vtkIdType inc[3])

|

| Different ways to get the increments for moving around the data.

| GetIncrements() calls ComputeIncrements() to ensure the

| increments are up to date. The first three methods compute the

| increments based on the active scalar field while the next three,

| the scalar field is passed in.

|

| Note that all methods which do not have the increments passed in

| are not thread-safe. When working on a given `vtkImageData`

| instance on multiple threads, each thread should use the `inc\*`

| overloads to compute the increments to avoid racing with other

| threads.

|

| GetIndexToPhysicalMatrix(...)

| GetIndexToPhysicalMatrix(self) -> vtkMatrix4x4

| C++: virtual vtkMatrix4x4 \*GetIndexToPhysicalMatrix()

|

| Get the transformation matrix from the index space to the

| physical space coordinate system of the dataset. The transform is

| a 4 by 4 matrix.

|

| GetMaxCellSize(...)

| GetMaxCellSize(self) -> int

| C++: int GetMaxCellSize() override;

|

| Convenience method returns largest cell size in dataset. This is

| generally used to allocate memory for supporting data structures.

| THIS METHOD IS THREAD SAFE

|

| GetMaxSpatialDimension(...)

| GetMaxSpatialDimension(self) -> int

| C++: int GetMaxSpatialDimension() override;

|

| Get the maximum spatial dimensionality of the data which is the

| maximum dimension of all cells.

|

| @warning This method MUST be overridden for performance reasons.

| Default implementation is very inefficient.

|

| GetNumberOfCells(...)

| GetNumberOfCells(self) -> int

| C++: vtkIdType GetNumberOfCells() override;

|

| Standard vtkDataSet API methods. See vtkDataSet for more

| information.

| \warning If GetCell(int,int,int) gets overridden in a subclass,

| it is

| necessary to override GetCell(vtkIdType) in that class as well

| since vtkImageData::GetCell(vtkIdType) will always call

| vkImageData::GetCell(int,int,int)

|

| GetNumberOfGenerationsFromBase(...)

| GetNumberOfGenerationsFromBase(self, type:str) -> int

| C++: vtkIdType GetNumberOfGenerationsFromBase(const char \*type)

| override;

|

| Given the name of a base class of this class type, return the

| distance of inheritance between this class type and the named

| class (how many generations of inheritance are there between this

| class and the named class). If the named class is not in this

| class's inheritance tree, return a negative value. Valid

| responses will always be nonnegative. This method works in

| combination with vtkTypeMacro found in vtkSetGet.h.

|

| GetNumberOfGenerationsFromBaseType(...)

| GetNumberOfGenerationsFromBaseType(type:str) -> int

| C++: static vtkIdType GetNumberOfGenerationsFromBaseType(

| const char \*type)

|

| Given a the name of a base class of this class type, return the

| distance of inheritance between this class type and the named

| class (how many generations of inheritance are there between this

| class and the named class). If the named class is not in this

| class's inheritance tree, return a negative value. Valid

| responses will always be nonnegative. This method works in

| combination with vtkTypeMacro found in vtkSetGet.h.

|

| GetNumberOfPoints(...)

| GetNumberOfPoints(self) -> int

| C++: vtkIdType GetNumberOfPoints() override;

|

| Determine the number of points composing the dataset. THIS METHOD

| IS THREAD SAFE

|

| GetNumberOfScalarComponents(...)

| GetNumberOfScalarComponents(meta\_data:vtkInformation) -> int

| C++: static int GetNumberOfScalarComponents(

| vtkInformation \*meta\_data)

| GetNumberOfScalarComponents(self) -> int

| C++: int GetNumberOfScalarComponents()

|

| GetOrigin(...)

| GetOrigin(self) -> (float, float, float)

| C++: virtual double \*GetOrigin()

|

| Set/Get the origin of the dataset. The origin is the position in

| world coordinates of the point of extent (0,0,0). This point does

| not have to be part of the dataset, in other words, the dataset

| extent does not have to start at (0,0,0) and the origin can be

| outside of the dataset bounding box. The origin plus spacing

| determine the position in space of the points.

|

| GetPhysicalToIndexMatrix(...)

| GetPhysicalToIndexMatrix(self) -> vtkMatrix4x4

| C++: virtual vtkMatrix4x4 \*GetPhysicalToIndexMatrix()

|

| Get the transformation matrix from the physical space to the

| index space coordinate system of the dataset. The transform is a

| 4 by 4 matrix.

|

| GetPoint(...)

| GetPoint(self, ptId:int) -> (float, float, float)

| C++: double \*GetPoint(vtkIdType ptId) override;

| GetPoint(self, id:int, x:[float, float, float]) -> None

| C++: void GetPoint(vtkIdType id, double x[3]) override;

|

| Get point coordinates with ptId such that: 0 <= ptId <

| NumberOfPoints. THIS METHOD IS NOT THREAD SAFE.

|

| GetPointCells(...)

| GetPointCells(self, ptId:int, cellIds:vtkIdList) -> None

| C++: void GetPointCells(vtkIdType ptId, vtkIdList \*cellIds)

| override;

|

| Topological inquiry to get cells using point. THIS METHOD IS

| THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET

| IS NOT MODIFIED

|

| GetPointGradient(...)

| GetPointGradient(self, i:int, j:int, k:int, s:vtkDataArray,

| g:[float, float, float]) -> None

| C++: virtual void GetPointGradient(int i, int j, int k,

| vtkDataArray \*s, double g[3])

|

| Given structured coordinates (i,j,k) for a point in a structured

| point dataset, compute the gradient vector from the scalar data

| at that point. The scalars s are the scalars from which the

| gradient is to be computed. This method will treat structured

| point datasets of any dimension.

|

| GetPoints(...)

| GetPoints(self) -> vtkPoints

| C++: vtkPoints \*GetPoints() override;

|

| If the subclass has (implicit/explicit) points, then return them.

| Otherwise, create a vtkPoints object and return that.

|

| DO NOT MODIFY THE RETURNED POINTS OBJECT.

|

| GetScalarComponentAsDouble(...)

| GetScalarComponentAsDouble(self, x:int, y:int, z:int,

| component:int) -> float

| C++: virtual double GetScalarComponentAsDouble(int x, int y,

| int z, int component)

|

| GetScalarComponentAsFloat(...)

| GetScalarComponentAsFloat(self, x:int, y:int, z:int,

| component:int) -> float

| C++: virtual float GetScalarComponentAsFloat(int x, int y, int z,

| int component)

|

| For access to data from wrappers

|

| GetScalarIndex(...)

| GetScalarIndex(self, coordinates:[int, int, int]) -> int

| C++: virtual vtkIdType GetScalarIndex(int coordinates[3])

| GetScalarIndex(self, x:int, y:int, z:int) -> int

| C++: virtual vtkIdType GetScalarIndex(int x, int y, int z)

|

| GetScalarIndexForExtent(...)

| GetScalarIndexForExtent(self, extent:[int, int, int, int, int,

| int]) -> int

| C++: virtual vtkIdType GetScalarIndexForExtent(int extent[6])

|

| Access the index for the scalar data

|

| GetScalarPointer(...)

| GetScalarPointer(self, coordinates:[int, int, int]) -> Pointer

| C++: virtual void \*GetScalarPointer(int coordinates[3])

| GetScalarPointer(self, x:int, y:int, z:int) -> Pointer

| C++: virtual void \*GetScalarPointer(int x, int y, int z)

| GetScalarPointer(self) -> Pointer

| C++: virtual void \*GetScalarPointer()

|

| GetScalarPointerForExtent(...)

| GetScalarPointerForExtent(self, extent:[int, int, int, int, int,

| int]) -> Pointer

| C++: virtual void \*GetScalarPointerForExtent(int extent[6])

|

| Access the native pointer for the scalar data

|

| GetScalarSize(...)

| GetScalarSize(self, meta\_data:vtkInformation) -> int

| C++: virtual int GetScalarSize(vtkInformation \*meta\_data)

| GetScalarSize(self) -> int

| C++: virtual int GetScalarSize()

|

| Get the size of the scalar type in bytes.

|

| GetScalarType(...)

| GetScalarType(meta\_data:vtkInformation) -> int

| C++: static int GetScalarType(vtkInformation \*meta\_data)

| GetScalarType(self) -> int

| C++: int GetScalarType()

|

| GetScalarTypeAsString(...)

| GetScalarTypeAsString(self) -> str

| C++: const char \*GetScalarTypeAsString()

|

| GetScalarTypeMax(...)

| GetScalarTypeMax(self, meta\_data:vtkInformation) -> float

| C++: virtual double GetScalarTypeMax(vtkInformation \*meta\_data)

| GetScalarTypeMax(self) -> float

| C++: virtual double GetScalarTypeMax()

|

| GetScalarTypeMin(...)

| GetScalarTypeMin(self, meta\_data:vtkInformation) -> float

| C++: virtual double GetScalarTypeMin(vtkInformation \*meta\_data)

| GetScalarTypeMin(self) -> float

| C++: virtual double GetScalarTypeMin()

|

| These returns the minimum and maximum values the ScalarType can

| hold without overflowing.

|

| GetSpacing(...)

| GetSpacing(self) -> (float, float, float)

| C++: virtual double \*GetSpacing()

|

| Set the spacing (width,height,length) of the cubical cells that

| compose the data set.

|

| GetTupleIndex(...)

| GetTupleIndex(self, array:vtkDataArray, coordinates:[int, int,

| int]) -> int

| C++: vtkIdType GetTupleIndex(vtkDataArray \*array,

| int coordinates[3])

|

| Given a data array and a coordinate, return the index of the

| tuple in the array corresponding to that coordinate.

|

| This method is analogous to GetArrayPointer(), but it conforms to

| the API of vtkGenericDataArray.

|

| GetVoxelGradient(...)

| GetVoxelGradient(self, i:int, j:int, k:int, s:vtkDataArray,

| g:vtkDataArray) -> None

| C++: virtual void GetVoxelGradient(int i, int j, int k,

| vtkDataArray \*s, vtkDataArray \*g)

|

| Given structured coordinates (i,j,k) for a voxel cell, compute

| the eight gradient values for the voxel corners. The order in

| which the gradient vectors are arranged corresponds to the

| ordering of the voxel points. Gradient vector is computed by

| central differences (except on edges of volume where forward

| difference is used). The scalars s are the scalars from which the

| gradient is to be computed. This method will treat only 3D

| structured point datasets (i.e., volumes).

|

| HasAnyBlankCells(...)

| HasAnyBlankCells(self) -> bool

| C++: bool HasAnyBlankCells() override;

|

| Returns 1 if there is any visibility constraint on the cells, 0

| otherwise.

|

| HasAnyBlankPoints(...)

| HasAnyBlankPoints(self) -> bool

| C++: bool HasAnyBlankPoints() override;

|

| Returns 1 if there is any visibility constraint on the points, 0

| otherwise.

|

| HasNumberOfScalarComponents(...)

| HasNumberOfScalarComponents(meta\_data:vtkInformation) -> bool

| C++: static bool HasNumberOfScalarComponents(

| vtkInformation \*meta\_data)

|

| HasScalarType(...)

| HasScalarType(meta\_data:vtkInformation) -> bool

| C++: static bool HasScalarType(vtkInformation \*meta\_data)

|

| Initialize(...)

| Initialize(self) -> None

| C++: void Initialize() override;

|

| Restore object to initial state. Release memory back to system.

|

| IsA(...)

| IsA(self, type:str) -> int

| C++: vtkTypeBool IsA(const char \*type) override;

|

| Return 1 if this class is the same type of (or a subclass of) the

| named class. Returns 0 otherwise. This method works in

| combination with vtkTypeMacro found in vtkSetGet.h.

|

| IsCellVisible(...)

| IsCellVisible(self, cellId:int) -> int

| C++: unsigned char IsCellVisible(vtkIdType cellId)

|

| Return non-zero value if specified point is visible. These

| methods should be called only after the dimensions of the grid

| are set.

|

| IsPointVisible(...)

| IsPointVisible(self, ptId:int) -> int

| C++: unsigned char IsPointVisible(vtkIdType ptId)

|

| Return non-zero value if specified point is visible. These

| methods should be called only after the dimensions of the grid

| are set.

|

| IsTypeOf(...)

| IsTypeOf(type:str) -> int

| C++: static vtkTypeBool IsTypeOf(const char \*type)

|

| Return 1 if this class type is the same type of (or a subclass

| of) the named class. Returns 0 otherwise. This method works in

| combination with vtkTypeMacro found in vtkSetGet.h.

|

| NewInstance(...)

| NewInstance(self) -> vtkImageData

| C++: vtkImageData \*NewInstance()

|

| PrepareForNewData(...)

| PrepareForNewData(self) -> None

| C++: void PrepareForNewData() override;

|

| make the output data ready for new data to be inserted. For most

| objects we just call Initialize. But for image data we leave the

| old data in case the memory can be reused.

|

| SafeDownCast(...)

| SafeDownCast(o:vtkObjectBase) -> vtkImageData

| C++: static vtkImageData \*SafeDownCast(vtkObjectBase \*o)

|

| SetAxisUpdateExtent(...)

| SetAxisUpdateExtent(self, axis:int, min:int, max:int,

| updateExtent:(int, ...), axisUpdateExtent:[int, ...]) -> None

| C++: virtual void SetAxisUpdateExtent(int axis, int min, int max,

| const int \*updateExtent, int \*axisUpdateExtent)

|

| Set / Get the extent on just one axis

|

| SetDimensions(...)

| SetDimensions(self, i:int, j:int, k:int) -> None

| C++: virtual void SetDimensions(int i, int j, int k)

| SetDimensions(self, dims:(int, int, int)) -> None

| C++: virtual void SetDimensions(const int dims[3])

|

| Same as SetExtent(0, i-1, 0, j-1, 0, k-1)

|

| SetDirectionMatrix(...)

| SetDirectionMatrix(self, m:vtkMatrix3x3) -> None

| C++: virtual void SetDirectionMatrix(vtkMatrix3x3 \*m)

| SetDirectionMatrix(self, elements:(float, float, float, float,

| float, float, float, float, float)) -> None

| C++: virtual void SetDirectionMatrix(const double elements[9])

| SetDirectionMatrix(self, e00:float, e01:float, e02:float,

| e10:float, e11:float, e12:float, e20:float, e21:float,

| e22:float) -> None

| C++: virtual void SetDirectionMatrix(double e00, double e01,

| double e02, double e10, double e11, double e12, double e20,

| double e21, double e22)

|

| SetExtent(...)

| SetExtent(self, extent:[int, int, int, int, int, int]) -> None

| C++: virtual void SetExtent(int extent[6])

| SetExtent(self, x1:int, x2:int, y1:int, y2:int, z1:int, z2:int)

| -> None

| C++: virtual void SetExtent(int x1, int x2, int y1, int y2,

| int z1, int z2)

|

| Set/Get the extent. On each axis, the extent is defined by the

| index of the first point and the index of the last point. The

| extent should be set before the "Scalars" are set or allocated.

| The Extent is stored in the order (X, Y, Z). The dataset extent

| does not have to start at (0,0,0). (0,0,0) is just the extent of

| the origin. The first point (the one with Id=0) is at extent

| (Extent[0],Extent[2],Extent[4]). As for any dataset, a data array

| on point data starts at Id=0.

|

| SetNumberOfScalarComponents(...)

| SetNumberOfScalarComponents(n:int, meta\_data:vtkInformation)

| -> None

| C++: static void SetNumberOfScalarComponents(int n,

| vtkInformation \*meta\_data)

|

| Set/Get the number of scalar components for points. As with the

| SetScalarType method this is setting pipeline info.

|

| SetOrigin(...)

| SetOrigin(self, i:float, j:float, k:float) -> None

| C++: virtual void SetOrigin(double i, double j, double k)

| SetOrigin(self, ijk:(float, float, float)) -> None

| C++: virtual void SetOrigin(const double ijk[3])

|

| SetScalarComponentFromDouble(...)

| SetScalarComponentFromDouble(self, x:int, y:int, z:int,

| component:int, v:float) -> None

| C++: virtual void SetScalarComponentFromDouble(int x, int y,

| int z, int component, double v)

|

| SetScalarComponentFromFloat(...)

| SetScalarComponentFromFloat(self, x:int, y:int, z:int,

| component:int, v:float) -> None

| C++: virtual void SetScalarComponentFromFloat(int x, int y, int z,

| int component, float v)

|

| SetScalarType(...)

| SetScalarType(\_\_a:int, meta\_data:vtkInformation) -> None

| C++: static void SetScalarType(int, vtkInformation \*meta\_data)

|

| SetSpacing(...)

| SetSpacing(self, i:float, j:float, k:float) -> None

| C++: virtual void SetSpacing(double i, double j, double k)

| SetSpacing(self, ijk:(float, float, float)) -> None

| C++: virtual void SetSpacing(const double ijk[3])

|

| ShallowCopy(...)

| ShallowCopy(self, src:vtkDataObject) -> None

| C++: void ShallowCopy(vtkDataObject \*src) override;

|

| Shallow and Deep copy.

|

| TransformContinuousIndexToPhysicalPoint(...)

| TransformContinuousIndexToPhysicalPoint(self, i:float, j:float,

| k:float, xyz:[float, float, float]) -> None

| C++: virtual void TransformContinuousIndexToPhysicalPoint(

| double i, double j, double k, double xyz[3])

| TransformContinuousIndexToPhysicalPoint(self, ijk:(float, float,

| float), xyz:[float, float, float]) -> None

| C++: virtual void TransformContinuousIndexToPhysicalPoint(

| const double ijk[3], double xyz[3])

| TransformContinuousIndexToPhysicalPoint(i:float, j:float, k:float,

| origin:(float, float, float), spacing:(float, float, float),

| direction:(float, float, float, float, float, float, float,

| float, float), xyz:[float, float, float]) -> None

| C++: static void TransformContinuousIndexToPhysicalPoint(double i,

| double j, double k, double const origin[3],

| double const spacing[3], double const direction[9],

| double xyz[3])

|

| Convert coordinates from index space (ijk) to physical space

| (xyz).

|

| TransformIndexToPhysicalPoint(...)

| TransformIndexToPhysicalPoint(self, i:int, j:int, k:int,

| xyz:[float, float, float]) -> None

| C++: virtual void TransformIndexToPhysicalPoint(int i, int j,

| int k, double xyz[3])

| TransformIndexToPhysicalPoint(self, ijk:(int, int, int),

| xyz:[float, float, float]) -> None

| C++: virtual void TransformIndexToPhysicalPoint(const int ijk[3],

| double xyz[3])

|

| TransformPhysicalNormalToContinuousIndex(...)

| TransformPhysicalNormalToContinuousIndex(self, xyz:(float, float,

| float), ijk:[float, float, float]) -> None

| C++: virtual void TransformPhysicalNormalToContinuousIndex(

| const double xyz[3], double ijk[3])

|

| Convert normal from physical space (xyz) to index space (ijk).

|

| TransformPhysicalPlaneToContinuousIndex(...)

| TransformPhysicalPlaneToContinuousIndex(self, pplane:(float,

| float, float, float), iplane:[float, float, float, float])

| -> None

| C++: virtual void TransformPhysicalPlaneToContinuousIndex(

| double const pplane[4], double iplane[4])

|

| Convert a plane from physical to a continuous index. The plane is

| represented as n(x-xo)=0; or using a four component normal:

| pplane=( nx,ny,nz,-(n(x0)) ).

|

| TransformPhysicalPointToContinuousIndex(...)

| TransformPhysicalPointToContinuousIndex(self, x:float, y:float,

| z:float, ijk:[float, float, float]) -> None

| C++: virtual void TransformPhysicalPointToContinuousIndex(

| double x, double y, double z, double ijk[3])

| TransformPhysicalPointToContinuousIndex(self, xyz:(float, float,

| float), ijk:[float, float, float]) -> None

| C++: virtual void TransformPhysicalPointToContinuousIndex(

| const double xyz[3], double ijk[3])

|

| Convert coordinates from physical space (xyz) to index space

| (ijk).

|

| UnBlankCell(...)

| UnBlankCell(self, ptId:int) -> None

| C++: virtual void UnBlankCell(vtkIdType ptId)

| UnBlankCell(self, i:int, j:int, k:int) -> None

| C++: virtual void UnBlankCell(int i, int j, int k)

|

| UnBlankPoint(...)

| UnBlankPoint(self, ptId:int) -> None

| C++: virtual void UnBlankPoint(vtkIdType ptId)

| UnBlankPoint(self, i:int, j:int, k:int) -> None

| C++: virtual void UnBlankPoint(int i, int j, int k)

|

| \_\_delattr\_\_(self, name, /)

| Implement delattr(self, name).

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_init\_\_(self, /, \*args, \*\*kwargs)

| Initialize self. See help(type(self)) for accurate signature.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_setattr\_\_(self, name, value, /)

| Implement setattr(self, name, value).

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| \_\_dict\_\_

| Dictionary of attributes set by user.

|

| \_\_this\_\_

| Pointer to the C++ object.

|

| actual\_memory\_size

| read-only, Calls GetActualMemorySize

|

| cell\_types\_array

| read-only, Calls GetCellTypesArray

|

| cells

| read-only, Calls GetCells

|

| data\_description

| read-only, Calls GetDataDescription

|

| data\_dimension

| read-only, Calls GetDataDimension

|

| data\_object\_type

| read-only, Calls GetDataObjectType

|

| dimensions

| read-write, Calls GetDimensions/SetDimensions

|

| direction\_matrix

| write-only, Calls SetDirectionMatrix

|

| extent

| read-write, Calls GetExtent/SetExtent

|

| extent\_type

| read-only, Calls GetExtentType

|

| increments

| read-only, Calls GetIncrements

|

| index\_to\_physical\_matrix

| read-only, Calls GetIndexToPhysicalMatrix

|

| max\_cell\_size

| read-only, Calls GetMaxCellSize

|

| max\_spatial\_dimension

| read-only, Calls GetMaxSpatialDimension

|

| number\_of\_cells

| read-only, Calls GetNumberOfCells

|

| number\_of\_points

| read-only, Calls GetNumberOfPoints

|

| number\_of\_scalar\_components

| read-only, Calls GetNumberOfScalarComponents

|

| origin

| read-write, Calls GetOrigin/SetOrigin

|

| physical\_to\_index\_matrix

| read-only, Calls GetPhysicalToIndexMatrix

|

| points

| read-only, Calls GetPoints

|

| scalar\_pointer

| read-only, Calls GetScalarPointer

|

| scalar\_size

| read-only, Calls GetScalarSize

|

| scalar\_type

| read-only, Calls GetScalarType

|

| scalar\_type\_max

| read-only, Calls GetScalarTypeMax

|

| scalar\_type\_min

| read-only, Calls GetScalarTypeMin

|

| spacing

| read-write, Calls GetSpacing/SetSpacing

|

| ----------------------------------------------------------------------

| Data and other attributes defined here:

|

| \_\_vtkname\_\_ = 'vtkImageData'

|

| ----------------------------------------------------------------------

| Methods inherited from vtkDataSet:

|

| AllocateCellGhostArray(...)

| AllocateCellGhostArray(self) -> vtkUnsignedCharArray

| C++: vtkUnsignedCharArray \*AllocateCellGhostArray()

|

| Allocate ghost array for cells.

|

| AllocatePointGhostArray(...)

| AllocatePointGhostArray(self) -> vtkUnsignedCharArray

| C++: vtkUnsignedCharArray \*AllocatePointGhostArray()

|

| Allocate ghost array for points.

|

| CheckAttributes(...)

| CheckAttributes(self) -> int

| C++: int CheckAttributes()

|

| This method checks to see if the cell and point attributes match

| the geometry. Many filters will crash if the number of tuples in

| an array is less than the number of points/cells. This method

| returns 1 if there is a mismatch, and 0 if everything is ok. It

| prints an error if an array is too short, and a warning if an

| array is too long.

|

| CopyAttributes(...)

| CopyAttributes(self, ds:vtkDataSet) -> None

| C++: virtual void CopyAttributes(vtkDataSet \*ds)

|

| Copy the attributes associated with the specified dataset to this

| instance of vtkDataSet. THIS METHOD IS NOT THREAD SAFE.

|

| GenerateGhostArray(...)

| GenerateGhostArray(self, zeroExt:[int, int, int, int, int, int])

| -> None

| C++: virtual void GenerateGhostArray(int zeroExt[6])

| GenerateGhostArray(self, zeroExt:[int, int, int, int, int, int],

| cellOnly:bool) -> None

| C++: virtual void GenerateGhostArray(int zeroExt[6],

| bool cellOnly)

|

| Normally called by pipeline executives or algorithms only. This

| method computes the ghost arrays for a given dataset. The zeroExt

| argument specifies the extent of the region which ghost type = 0.

|

| GetAttributesAsFieldData(...)

| GetAttributesAsFieldData(self, type:int) -> vtkFieldData

| C++: vtkFieldData \*GetAttributesAsFieldData(int type) override;

|

| Returns the attributes of the data object as a vtkFieldData. This

| returns non-null values in all the same cases as GetAttributes,

| in addition to the case of FIELD, which will return the field

| data for any vtkDataObject subclass.

|

| GetBounds(...)

| GetBounds(self) -> (float, float, float, float, float, float)

| C++: double \*GetBounds()

| GetBounds(self, bounds:[float, float, float, float, float, float])

| -> None

| C++: void GetBounds(double bounds[6])

|

| Return a pointer to the geometry bounding box in the form

| (xmin,xmax, ymin,ymax, zmin,zmax). THIS METHOD IS NOT THREAD

| SAFE.

|

| GetCellData(...)

| GetCellData(self) -> vtkCellData

| C++: vtkCellData \*GetCellData()

|

| Return a pointer to this dataset's cell data. THIS METHOD IS

| THREAD SAFE

|

| GetCellGhostArray(...)

| GetCellGhostArray(self) -> vtkUnsignedCharArray

| C++: vtkUnsignedCharArray \*GetCellGhostArray()

|

| Get the array that defines the ghost type of each cell. We cache

| the pointer to the array to save a lookup involving string

| comparisons

|

| GetCellNumberOfFaces(...)

| GetCellNumberOfFaces(self, cellId:int, cellType:int,

| cell:vtkGenericCell) -> int

| C++: int GetCellNumberOfFaces(vtkIdType cellId,

| unsigned char &cellType, vtkGenericCell \*cell)

|

| Get the number of faces of a cell.

|

| Most of the times extracting the number of faces requires only

| extracting the cell type. However, for some cell types, the

| number of faces is not constant. For example, a vtkPolyhedron

| cell can have a different number of faces for each cell. That's

| why this method requires the cell id and the dataset.

|

| GetCellTypes(...)

| GetCellTypes(self, types:vtkCellTypes) -> None

| C++: virtual void GetCellTypes(vtkCellTypes \*types)

|

| Get a list of types of cells in a dataset. The list consists of

| an array of types (not necessarily in any order), with a single

| entry per type. For example a dataset 5 triangles, 3 lines, and

| 100 hexahedra would result a list of three entries, corresponding

| to the types VTK\_TRIANGLE, VTK\_LINE, and VTK\_HEXAHEDRON. THIS

| METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND

| THE DATASET IS NOT MODIFIED

|

| GetCenter(...)

| GetCenter(self) -> (float, float, float)

| C++: double \*GetCenter()

| GetCenter(self, center:[float, float, float]) -> None

| C++: void GetCenter(double center[3])

|

| Get the center of the bounding box. THIS METHOD IS NOT THREAD

| SAFE.

|

| GetGhostArray(...)

| GetGhostArray(self, type:int) -> vtkUnsignedCharArray

| C++: vtkUnsignedCharArray \*GetGhostArray(int type) override;

|

| Returns the ghost array for the given type (point or cell). Takes

| advantage of the cache with the pointer to the array to save a

| string comparison.

|

| GetLength(...)

| GetLength(self) -> float

| C++: double GetLength()

|

| Return the length of the diagonal of the bounding box. THIS

| METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND

| THE DATASET IS NOT MODIFIED

|

| GetLength2(...)

| GetLength2(self) -> float

| C++: double GetLength2()

|

| Return the squared length of the diagonal of the bounding box.

| THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD

| AND THE DATASET IS NOT MODIFIED

|

| GetMTime(...)

| GetMTime(self) -> int

| C++: vtkMTimeType GetMTime() override;

|

| Datasets are composite objects and need to check each part for

| MTime THIS METHOD IS THREAD SAFE

|

| GetNumberOfElements(...)

| GetNumberOfElements(self, type:int) -> int

| C++: vtkIdType GetNumberOfElements(int type) override;

|

| Get the number of elements for a specific attribute type (POINT,

| CELL, etc.).

|

| GetPointData(...)

| GetPointData(self) -> vtkPointData

| C++: vtkPointData \*GetPointData()

|

| Return a pointer to this dataset's point data. THIS METHOD IS

| THREAD SAFE

|

| GetPointGhostArray(...)

| GetPointGhostArray(self) -> vtkUnsignedCharArray

| C++: vtkUnsignedCharArray \*GetPointGhostArray()

|

| Gets the array that defines the ghost type of each point. We

| cache the pointer to the array to save a lookup involving string

| comparisons

|

| GetScalarRange(...)

| GetScalarRange(self, range:[float, float]) -> None

| C++: virtual void GetScalarRange(double range[2])

| GetScalarRange(self) -> (float, float)

| C++: double \*GetScalarRange()

|

| Convenience method to get the range of the first component (and

| only the first component) of any scalars in the data set. If the

| data has both point data and cell data, it returns the (min/max)

| range of combined point and cell data. If there are no point or

| cell scalars the method will return (0,1). Note: It might be

| necessary to call Update to create or refresh the scalars before

| calling this method. THIS METHOD IS THREAD SAFE IF FIRST CALLED

| FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED

|

| HasAnyGhostCells(...)

| HasAnyGhostCells(self) -> bool

| C++: bool HasAnyGhostCells()

|

| Returns 1 if there are any ghost cells 0 otherwise.

|

| HasAnyGhostPoints(...)

| HasAnyGhostPoints(self) -> bool

| C++: bool HasAnyGhostPoints()

|

| Returns 1 if there are any ghost points 0 otherwise.

|

| NewCellIterator(...)

| NewCellIterator(self) -> vtkCellIterator

| C++: virtual vtkCellIterator \*NewCellIterator()

|

| Return an iterator that traverses the cells in this data set.

|

| SetCellOrderAndRationalWeights(...)

| SetCellOrderAndRationalWeights(self, cellId:int,

| cell:vtkGenericCell) -> None

| C++: void SetCellOrderAndRationalWeights(vtkIdType cellId,

| vtkGenericCell \*cell)

|

| Squeeze(...)

| Squeeze(self) -> None

| C++: virtual void Squeeze()

|

| Reclaim any extra memory used to store data. THIS METHOD IS NOT

| THREAD SAFE.

|

| SupportsGhostArray(...)

| SupportsGhostArray(self, type:int) -> bool

| C++: bool SupportsGhostArray(int type) override;

|

| Returns true for POINT or CELL, false otherwise

|

| UpdateCellGhostArrayCache(...)

| UpdateCellGhostArrayCache(self) -> None

| C++: void UpdateCellGhostArrayCache()

|

| Updates the pointer to the cell ghost array.

|

| UpdatePointGhostArrayCache(...)

| UpdatePointGhostArrayCache(self) -> None

| C++: void UpdatePointGhostArrayCache()

|

| Updates the pointer to the point ghost array.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from vtkDataSet:

|

| bounds

| read-only, Calls GetBounds

|

| cell\_data

| read-only, Calls GetCellData

|

| cell\_ghost\_array

| read-only, Calls GetCellGhostArray

|

| center

| read-only, Calls GetCenter

|

| length

| read-only, Calls GetLength

|

| length2

| read-only, Calls GetLength2

|

| m\_time

| read-only, Calls GetMTime

|

| point\_data

| read-only, Calls GetPointData

|

| point\_ghost\_array

| read-only, Calls GetPointGhostArray

|

| scalar\_range

| read-only, Calls GetScalarRange

|

| ----------------------------------------------------------------------

| Data and other attributes inherited from vtkDataSet:

|

| CELL\_DATA\_FIELD = 2

|

| DATA\_OBJECT\_FIELD = 0

|

| FieldDataType = <class 'vtkmodules.vtkCommonDataModel.vtkDataSet.Field...

|

| POINT\_DATA\_FIELD = 1

|

| ----------------------------------------------------------------------

| Methods inherited from vtkDataObject:

|

| ALL\_PIECES\_EXTENT(...)

| ALL\_PIECES\_EXTENT() -> vtkInformationIntegerVectorKey

| C++: static vtkInformationIntegerVectorKey \*ALL\_PIECES\_EXTENT()

|

| BOUNDING\_BOX(...)

| BOUNDING\_BOX() -> vtkInformationDoubleVectorKey

| C++: static vtkInformationDoubleVectorKey \*BOUNDING\_BOX()

|

| CELL\_DATA\_VECTOR(...)

| CELL\_DATA\_VECTOR() -> vtkInformationInformationVectorKey

| C++: static vtkInformationInformationVectorKey \*CELL\_DATA\_VECTOR()

|

| DATA\_EXTENT(...)

| DATA\_EXTENT() -> vtkInformationIntegerPointerKey

| C++: static vtkInformationIntegerPointerKey \*DATA\_EXTENT()

|

| DATA\_EXTENT\_TYPE(...)

| DATA\_EXTENT\_TYPE() -> vtkInformationIntegerKey

| C++: static vtkInformationIntegerKey \*DATA\_EXTENT\_TYPE()

|

| DATA\_NUMBER\_OF\_GHOST\_LEVELS(...)

| DATA\_NUMBER\_OF\_GHOST\_LEVELS() -> vtkInformationIntegerKey

| C++: static vtkInformationIntegerKey \*DATA\_NUMBER\_OF\_GHOST\_LEVELS(

| )

|

| DATA\_NUMBER\_OF\_PIECES(...)

| DATA\_NUMBER\_OF\_PIECES() -> vtkInformationIntegerKey

| C++: static vtkInformationIntegerKey \*DATA\_NUMBER\_OF\_PIECES()

|

| DATA\_OBJECT(...)

| DATA\_OBJECT() -> vtkInformationDataObjectKey

| C++: static vtkInformationDataObjectKey \*DATA\_OBJECT()

|

| DATA\_PIECE\_NUMBER(...)

| DATA\_PIECE\_NUMBER() -> vtkInformationIntegerKey

| C++: static vtkInformationIntegerKey \*DATA\_PIECE\_NUMBER()

|

| DATA\_TIME\_STEP(...)

| DATA\_TIME\_STEP() -> vtkInformationDoubleKey

| C++: static vtkInformationDoubleKey \*DATA\_TIME\_STEP()

|

| DATA\_TYPE\_NAME(...)

| DATA\_TYPE\_NAME() -> vtkInformationStringKey

| C++: static vtkInformationStringKey \*DATA\_TYPE\_NAME()

|

| DIRECTION(...)

| DIRECTION() -> vtkInformationDoubleVectorKey

| C++: static vtkInformationDoubleVectorKey \*DIRECTION()

|

| DataHasBeenGenerated(...)

| DataHasBeenGenerated(self) -> None

| C++: void DataHasBeenGenerated()

|

| This method is called by the source when it executes to generate

| data. It is sort of the opposite of ReleaseData. It sets the

| DataReleased flag to 0, and sets a new UpdateTime.

|

| EDGE\_DATA\_VECTOR(...)

| EDGE\_DATA\_VECTOR() -> vtkInformationInformationVectorKey

| C++: static vtkInformationInformationVectorKey \*EDGE\_DATA\_VECTOR()

|

| FIELD\_ACTIVE\_ATTRIBUTE(...)

| FIELD\_ACTIVE\_ATTRIBUTE() -> vtkInformationIntegerKey

| C++: static vtkInformationIntegerKey \*FIELD\_ACTIVE\_ATTRIBUTE()

|

| FIELD\_ARRAY\_TYPE(...)

| FIELD\_ARRAY\_TYPE() -> vtkInformationIntegerKey

| C++: static vtkInformationIntegerKey \*FIELD\_ARRAY\_TYPE()

|

| FIELD\_ASSOCIATION(...)

| FIELD\_ASSOCIATION() -> vtkInformationIntegerKey

| C++: static vtkInformationIntegerKey \*FIELD\_ASSOCIATION()

|

| FIELD\_ATTRIBUTE\_TYPE(...)

| FIELD\_ATTRIBUTE\_TYPE() -> vtkInformationIntegerKey

| C++: static vtkInformationIntegerKey \*FIELD\_ATTRIBUTE\_TYPE()

|

| FIELD\_NAME(...)

| FIELD\_NAME() -> vtkInformationStringKey

| C++: static vtkInformationStringKey \*FIELD\_NAME()

|

| FIELD\_NUMBER\_OF\_COMPONENTS(...)

| FIELD\_NUMBER\_OF\_COMPONENTS() -> vtkInformationIntegerKey

| C++: static vtkInformationIntegerKey \*FIELD\_NUMBER\_OF\_COMPONENTS()

|

| FIELD\_NUMBER\_OF\_TUPLES(...)

| FIELD\_NUMBER\_OF\_TUPLES() -> vtkInformationIntegerKey

| C++: static vtkInformationIntegerKey \*FIELD\_NUMBER\_OF\_TUPLES()

|

| FIELD\_OPERATION(...)

| FIELD\_OPERATION() -> vtkInformationIntegerKey

| C++: static vtkInformationIntegerKey \*FIELD\_OPERATION()

|

| FIELD\_RANGE(...)

| FIELD\_RANGE() -> vtkInformationDoubleVectorKey

| C++: static vtkInformationDoubleVectorKey \*FIELD\_RANGE()

|

| GetActiveFieldInformation(...)

| GetActiveFieldInformation(info:vtkInformation,

| fieldAssociation:int, attributeType:int) -> vtkInformation

| C++: static vtkInformation \*GetActiveFieldInformation(

| vtkInformation \*info, int fieldAssociation, int attributeType)

|

| Return the information object within the input information

| object's field data corresponding to the specified association

| (FIELD\_ASSOCIATION\_POINTS or FIELD\_ASSOCIATION\_CELLS) and

| attribute (SCALARS, VECTORS, NORMALS, TCOORDS, or TENSORS)

|

| GetAssociationTypeAsString(...)

| GetAssociationTypeAsString(associationType:int) -> str

| C++: static const char \*GetAssociationTypeAsString(

| int associationType)

|

| Given an integer association type, this static method returns a

| string type for the attribute (i.e. associationType = 0: returns

| "Points").

|

| GetAssociationTypeFromString(...)

| GetAssociationTypeFromString(associationName:str) -> int

| C++: static int GetAssociationTypeFromString(

| const char \*associationName)

|

| Given a string association name, this static method returns an

| integer association type for the attribute (i.e. associationName

| = "Points": returns 0).

|

| GetAttributeTypeForArray(...)

| GetAttributeTypeForArray(self, arr:vtkAbstractArray) -> int

| C++: virtual int GetAttributeTypeForArray(vtkAbstractArray \*arr)

|

| Retrieves the attribute type that an array came from. This is

| useful for obtaining which attribute type a input array to an

| algorithm came from (retrieved from

| GetInputAbstractArrayToProcesss).

|

| GetAttributes(...)

| GetAttributes(self, type:int) -> vtkDataSetAttributes

| C++: virtual vtkDataSetAttributes \*GetAttributes(int type)

|

| Returns the attributes of the data object of the specified

| attribute type. The type may be: POINT - Defined in vtkDataSet

| subclasses. CELL - Defined in vtkDataSet subclasses. VERTEX -

| Defined in vtkGraph subclasses. EDGE - Defined in vtkGraph

| subclasses. ROW - Defined in vtkTable. The other attribute

| type, FIELD, will return nullptr since field data is stored as a

| vtkFieldData instance, not a vtkDataSetAttributes instance. To

| retrieve field data, use GetAttributesAsFieldData.

|

| @warning This method NEEDS to be

| overridden in subclasses to work as documented. If not, it

| returns nullptr for any type but FIELD.

|

| GetDataReleased(...)

| GetDataReleased(self) -> int

| C++: virtual vtkTypeBool GetDataReleased()

|

| Get the flag indicating the data has been released.

|

| GetFieldData(...)

| GetFieldData(self) -> vtkFieldData

| C++: virtual vtkFieldData \*GetFieldData()

|

| GetGlobalReleaseDataFlag(...)

| GetGlobalReleaseDataFlag() -> int

| C++: static vtkTypeBool GetGlobalReleaseDataFlag()

|

| GetInformation(...)

| GetInformation(self) -> vtkInformation

| C++: virtual vtkInformation \*GetInformation()

|

| Set/Get the information object associated with this data object.

|

| GetNamedFieldInformation(...)

| GetNamedFieldInformation(info:vtkInformation,

| fieldAssociation:int, name:str) -> vtkInformation

| C++: static vtkInformation \*GetNamedFieldInformation(

| vtkInformation \*info, int fieldAssociation, const char \*name)

|

| Return the information object within the input information

| object's field data corresponding to the specified association

| (FIELD\_ASSOCIATION\_POINTS or FIELD\_ASSOCIATION\_CELLS) and name.

|

| GetUpdateTime(...)

| GetUpdateTime(self) -> int

| C++: vtkMTimeType GetUpdateTime()

|

| Used by Threaded ports to determine if they should initiate an

| asynchronous update (still in development).

|

| GlobalReleaseDataFlagOff(...)

| GlobalReleaseDataFlagOff(self) -> None

| C++: void GlobalReleaseDataFlagOff()

|

| GlobalReleaseDataFlagOn(...)

| GlobalReleaseDataFlagOn(self) -> None

| C++: void GlobalReleaseDataFlagOn()

|

| ORIGIN(...)

| ORIGIN() -> vtkInformationDoubleVectorKey

| C++: static vtkInformationDoubleVectorKey \*ORIGIN()

|

| PIECE\_EXTENT(...)

| PIECE\_EXTENT() -> vtkInformationIntegerVectorKey

| C++: static vtkInformationIntegerVectorKey \*PIECE\_EXTENT()

|

| POINT\_DATA\_VECTOR(...)

| POINT\_DATA\_VECTOR() -> vtkInformationInformationVectorKey

| C++: static vtkInformationInformationVectorKey \*POINT\_DATA\_VECTOR(

| )

|

| ReleaseData(...)

| ReleaseData(self) -> None

| C++: void ReleaseData()

|

| Release data back to system to conserve memory resource. Used

| during visualization network execution. Releasing this data does

| not make down-stream data invalid.

|

| RemoveNamedFieldInformation(...)

| RemoveNamedFieldInformation(info:vtkInformation,

| fieldAssociation:int, name:str) -> None

| C++: static void RemoveNamedFieldInformation(vtkInformation \*info,

| int fieldAssociation, const char \*name)

|

| Remove the info associated with an array

|

| SIL(...)

| SIL() -> vtkInformationDataObjectKey

| C++: static vtkInformationDataObjectKey \*SIL()

|

| SPACING(...)

| SPACING() -> vtkInformationDoubleVectorKey

| C++: static vtkInformationDoubleVectorKey \*SPACING()

|

| SetActiveAttribute(...)

| SetActiveAttribute(info:vtkInformation, fieldAssociation:int,

| attributeName:str, attributeType:int) -> vtkInformation

| C++: static vtkInformation \*SetActiveAttribute(

| vtkInformation \*info, int fieldAssociation,

| const char \*attributeName, int attributeType)

|

| Set the named array to be the active field for the specified type

| (SCALARS, VECTORS, NORMALS, TCOORDS, or TENSORS) and association

| (FIELD\_ASSOCIATION\_POINTS or FIELD\_ASSOCIATION\_CELLS). Returns

| the active field information object and creates on entry if one

| not found.

|

| SetActiveAttributeInfo(...)

| SetActiveAttributeInfo(info:vtkInformation, fieldAssociation:int,

| attributeType:int, name:str, arrayType:int, numComponents:int,

| numTuples:int) -> None

| C++: static void SetActiveAttributeInfo(vtkInformation \*info,

| int fieldAssociation, int attributeType, const char \*name,

| int arrayType, int numComponents, int numTuples)

|

| Set the name, array type, number of components, and number of

| tuples within the passed information object for the active

| attribute of type attributeType (in specified association,

| FIELD\_ASSOCIATION\_POINTS or FIELD\_ASSOCIATION\_CELLS). If there

| is not an active attribute of the specified type, an entry in the

| information object is created. If arrayType, numComponents, or

| numTuples equal to -1, or name=nullptr the value is not changed.

|

| SetFieldData(...)

| SetFieldData(self, \_\_a:vtkFieldData) -> None

| C++: virtual void SetFieldData(vtkFieldData \*)

|

| Assign or retrieve a general field data to this data object.

|

| SetGlobalReleaseDataFlag(...)

| SetGlobalReleaseDataFlag(val:int) -> None

| C++: static void SetGlobalReleaseDataFlag(vtkTypeBool val)

|

| Turn on/off flag to control whether every object releases its

| data after being used by a filter.

|

| SetInformation(...)

| SetInformation(self, \_\_a:vtkInformation) -> None

| C++: virtual void SetInformation(vtkInformation \*)

|

| SetPointDataActiveScalarInfo(...)

| SetPointDataActiveScalarInfo(info:vtkInformation, arrayType:int,

| numComponents:int) -> None

| C++: static void SetPointDataActiveScalarInfo(

| vtkInformation \*info, int arrayType, int numComponents)

|

| Convenience version of previous method for use (primarily) by the

| Imaging filters. If arrayType or numComponents == -1, the value

| is not changed.

|

| VERTEX\_DATA\_VECTOR(...)

| VERTEX\_DATA\_VECTOR() -> vtkInformationInformationVectorKey

| C++: static vtkInformationInformationVectorKey \*VERTEX\_DATA\_VECTOR(

| )

|

| \_\_rrshift\_\_(self, value, /)

| Return value>>self.

|

| \_\_rshift\_\_(self, value, /)

| Return self>>value.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from vtkDataObject:

|

| data\_released

| read-only, Calls GetDataReleased

|

| field\_data

| read-write, Calls GetFieldData/SetFieldData

|

| global\_release\_data\_flag

| read-write, Calls GetGlobalReleaseDataFlag/SetGlobalReleaseDataFlag

|

| information

| read-write, Calls GetInformation/SetInformation

|

| update\_time

| read-only, Calls GetUpdateTime

|

| ----------------------------------------------------------------------

| Data and other attributes inherited from vtkDataObject:

|

| AttributeTypes = <class 'vtkmodules.vtkCommonDataModel.vtkDataObject.A...

|

| CELL = 1

|

| EDGE = 5

|

| FIELD = 2

|

| FIELD\_ASSOCIATION\_CELLS = 1

|

| FIELD\_ASSOCIATION\_EDGES = 5

|

| FIELD\_ASSOCIATION\_NONE = 2

|

| FIELD\_ASSOCIATION\_POINTS = 0

|

| FIELD\_ASSOCIATION\_POINTS\_THEN\_CELLS = 3

|

| FIELD\_ASSOCIATION\_ROWS = 6

|

| FIELD\_ASSOCIATION\_VERTICES = 4

|

| FIELD\_OPERATION\_MODIFIED = 2

|

| FIELD\_OPERATION\_PRESERVED = 0

|

| FIELD\_OPERATION\_REINTERPOLATED = 1

|

| FIELD\_OPERATION\_REMOVED = 3

|

| FieldAssociations = <class 'vtkmodules.vtkCommonDataModel.vtkDataObjec...

|

| FieldOperations = <class 'vtkmodules.vtkCommonDataModel.vtkDataObject....

|

| NUMBER\_OF\_ASSOCIATIONS = 7

|

| NUMBER\_OF\_ATTRIBUTE\_TYPES = 7

|

| POINT = 0

|

| POINT\_THEN\_CELL = 3

|

| ROW = 6

|

| VERTEX = 4

|

| ----------------------------------------------------------------------

| Methods inherited from vtkmodules.vtkCommonCore.vtkObject:

|

| AddObserver(...)

| AddObserver(self, event:int, command:Callback, priority:float=0.0) -> int

| C++: unsigned long AddObserver(const char\* event,

| vtkCommand\* command, float priority=0.0f)

|

| Add an event callback command(o:vtkObject, event:int) for an event type.

| Returns a handle that can be used with RemoveEvent(event:int).

|

| BreakOnError(...)

| BreakOnError() -> None

| C++: static void BreakOnError()

|

| This method is called when vtkErrorMacro executes. It allows the

| debugger to break on error.

|

| DebugOff(...)

| DebugOff(self) -> None

| C++: virtual void DebugOff()

|

| Turn debugging output off.

|

| DebugOn(...)

| DebugOn(self) -> None

| C++: virtual void DebugOn()

|

| Turn debugging output on.

|

| GetCommand(...)

| GetCommand(self, tag:int) -> vtkCommand

| C++: vtkCommand \*GetCommand(unsigned long tag)

|

| GetDebug(...)

| GetDebug(self) -> bool

| C++: bool GetDebug()

|

| Get the value of the debug flag.

|

| GetGlobalWarningDisplay(...)

| GetGlobalWarningDisplay() -> int

| C++: static vtkTypeBool GetGlobalWarningDisplay()

|

| GetObjectDescription(...)

| GetObjectDescription(self) -> str

| C++: std::string GetObjectDescription() override;

|

| The object description printed in messages and PrintSelf output.

| To be used only for reporting purposes.

|

| GetObjectName(...)

| GetObjectName(self) -> str

| C++: virtual std::string GetObjectName()

|

| GlobalWarningDisplayOff(...)

| GlobalWarningDisplayOff() -> None

| C++: static void GlobalWarningDisplayOff()

|

| GlobalWarningDisplayOn(...)

| GlobalWarningDisplayOn() -> None

| C++: static void GlobalWarningDisplayOn()

|

| HasObserver(...)

| HasObserver(self, event:int, \_\_b:vtkCommand) -> int

| C++: vtkTypeBool HasObserver(unsigned long event, vtkCommand \*)

| HasObserver(self, event:str, \_\_b:vtkCommand) -> int

| C++: vtkTypeBool HasObserver(const char \*event, vtkCommand \*)

| HasObserver(self, event:int) -> int

| C++: vtkTypeBool HasObserver(unsigned long event)

| HasObserver(self, event:str) -> int

| C++: vtkTypeBool HasObserver(const char \*event)

|

| InvokeEvent(...)

| InvokeEvent(self, event:int, callData:Any) -> int

| C++: int InvokeEvent(unsigned long event, void\* callData)

| InvokeEvent(self, event:str, callData:Any) -> int

| C++: int InvokeEvent(const char\* event, void\* callData)

| InvokeEvent(self, event:int) -> int

| C++: int InvokeEvent(unsigned long event)

| InvokeEvent(self, event:str) -> int

| C++: int InvokeEvent(const char\* event)

|

| This method invokes an event and returns whether the event was

| aborted or not. If the event was aborted, the return value is 1,

| otherwise it is 0.

|

| Modified(...)

| Modified(self) -> None

| C++: virtual void Modified()

|

| Update the modification time for this object. Many filters rely

| on the modification time to determine if they need to recompute

| their data. The modification time is a unique monotonically

| increasing unsigned long integer.

|

| RemoveAllObservers(...)

| RemoveAllObservers(self) -> None

| C++: void RemoveAllObservers()

|

| RemoveObserver(...)

| RemoveObserver(self, \_\_a:vtkCommand) -> None

| C++: void RemoveObserver(vtkCommand \*)

| RemoveObserver(self, tag:int) -> None

| C++: void RemoveObserver(unsigned long tag)

|

| RemoveObservers(...)

| RemoveObservers(self, event:int, \_\_b:vtkCommand) -> None

| C++: void RemoveObservers(unsigned long event, vtkCommand \*)

| RemoveObservers(self, event:str, \_\_b:vtkCommand) -> None

| C++: void RemoveObservers(const char \*event, vtkCommand \*)

| RemoveObservers(self, event:int) -> None

| C++: void RemoveObservers(unsigned long event)

| RemoveObservers(self, event:str) -> None

| C++: void RemoveObservers(const char \*event)

|

| SetDebug(...)

| SetDebug(self, debugFlag:bool) -> None

| C++: void SetDebug(bool debugFlag)

|

| Set the value of the debug flag. A true value turns debugging on.

|

| SetGlobalWarningDisplay(...)

| SetGlobalWarningDisplay(val:int) -> None

| C++: static void SetGlobalWarningDisplay(vtkTypeBool val)

|

| This is a global flag that controls whether any debug, warning or

| error messages are displayed.

|

| SetObjectName(...)

| SetObjectName(self, objectName:str) -> None

| C++: virtual void SetObjectName(const std::string &objectName)

|

| Set/get the name of this object for reporting purposes. The name

| appears in warning and debug messages and in the Print output.

| Setting the object name does not change the MTime and does not

| invoke a ModifiedEvent. Derived classes implementing copying

| methods are expected not to copy the ObjectName.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from vtkmodules.vtkCommonCore.vtkObject:

|

| debug

| read-write, Calls GetDebug/SetDebug

|

| global\_warning\_display

| read-write, Calls GetGlobalWarningDisplay/SetGlobalWarningDisplay

|

| object\_description

| read-only, Calls GetObjectDescription

|

| object\_name

| read-write, Calls GetObjectName/SetObjectName

|

| ----------------------------------------------------------------------

| Methods inherited from vtkmodules.vtkCommonCore.vtkObjectBase:

|

| FastDelete(...)

| FastDelete(self) -> None

| C++: virtual void FastDelete()

|

| Delete a reference to this object. This version will not invoke

| garbage collection and can potentially leak the object if it is

| part of a reference loop. Use this method only when it is known

| that the object has another reference and would not be collected

| if a full garbage collection check were done.

|

| GetAddressAsString(...)

| GetAddressAsString(self, classname:str) -> str

|

| Get address of C++ object in format 'Addr=%p' after casting to

| the specified type. This method is obsolete, you can get the

| same information from o.\_\_this\_\_.

|

| GetClassName(...)

| GetClassName(self) -> str

| C++: const char \*GetClassName()

|

| Return the class name as a string.

|

| GetIsInMemkind(...)

| GetIsInMemkind(self) -> bool

| C++: bool GetIsInMemkind()

|

| A local state flag that remembers whether this object lives in

| the normal or extended memory space.

|

| GetReferenceCount(...)

| GetReferenceCount(self) -> int

| C++: int GetReferenceCount()

|

| Return the current reference count of this object.

|

| GetUsingMemkind(...)

| GetUsingMemkind() -> bool

| C++: static bool GetUsingMemkind()

|

| A global state flag that controls whether vtkObjects are

| constructed in the usual way (the default) or within the extended

| memory space.

|

| InitializeObjectBase(...)

| InitializeObjectBase(self) -> None

| C++: void InitializeObjectBase()

|

| Register(...)

| Register(self, o:vtkObjectBase)

| C++: virtual void Register(vtkObjectBase \*o)

|

| Increase the reference count by 1.

|

| SetMemkindDirectory(...)

| SetMemkindDirectory(directoryname:str) -> None

| C++: static void SetMemkindDirectory(const char \*directoryname)

|

| The name of a directory, ideally mounted -o dax, to memory map an

| extended memory space within. This must be called before any

| objects are constructed in the extended space. It can not be

| changed once setup.

|

| SetReferenceCount(...)

| SetReferenceCount(self, \_\_a:int) -> None

| C++: void SetReferenceCount(int)

|

| Sets the reference count. (This is very dangerous, use with

| care.)

|

| UnRegister(...)

| UnRegister(self, o:vtkObjectBase)

| C++: virtual void UnRegister(vtkObjectBase\* o)

|

| Decrease the reference count (release by another object). This

| has the same effect as invoking Delete() (i.e., it reduces the

| reference count by 1).

|

| UsesGarbageCollector(...)

| UsesGarbageCollector(self) -> bool

| C++: virtual bool UsesGarbageCollector()

|

| Indicate whether the class uses `vtkGarbageCollector` or not.

|

| Most classes will not need to do this, but if the class

| participates in a strongly-connected reference count cycle,

| participation can resolve these cycles.

|

| If overriding this method to return true, the `ReportReferences`

| method should be overridden to report references that may be in

| cycles.

|

| ----------------------------------------------------------------------

| Class methods inherited from vtkmodules.vtkCommonCore.vtkObjectBase:

|

| override(...) from builtins.type

| This method can be used to override a VTK class with a Python subclass.

| The class type passed to override will afterwards be instantiated

| instead of the type override is called on.

| For example,

|

| class foo(vtk.vtkPoints):

| pass

| vtk.vtkPoints.override(foo)

|

| will lead to foo being instantied every time vtkPoints() is called.

| The main objective of this functionality is to enable developers to

| extend VTK classes with more pythonic subclasses that contain

| convenience functionality.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from vtkmodules.vtkCommonCore.vtkObjectBase:

|

| class\_name

| read-only, Calls GetClassName

|

| is\_in\_memkind

| read-only, Calls GetIsInMemkind

|

| memkind\_directory

| write-only, Calls SetMemkindDirectory

|

| reference\_count

| read-write, Calls GetReferenceCount/SetReferenceCount

|

| using\_memkind

| read-only, Calls GetUsingMemkind

>>>

Help on vtkPointData object:

class vtkPointData(vtkDataSetAttributes)

| vtkPointData - represent and manipulate point attribute data

|

| Superclass: vtkDataSetAttributes

|

| vtkPointData is a class that is used to represent and manipulate

| point attribute data (e.g., scalars, vectors, normals, texture

| coordinates, etc.) Most of the functionality is handled by

| vtkDataSetAttributes.

|

| By default, `GhostTypesToSkip` is set to `DUPLICATEPOINT |

| HIDDENPOINT`. See `vtkDataSetAttributes` for the definition of those

| constants.

|

| Method resolution order:

| vtkPointData

| vtkDataSetAttributes

| vtkFieldData

| vtkmodules.vtkCommonCore.vtkObject

| vtkmodules.vtkCommonCore.vtkObjectBase

| builtins.object

|

| Methods defined here:

|

| ExtendedNew(...)

| ExtendedNew() -> vtkPointData

| C++: static vtkPointData \*ExtendedNew()

|

| GetNumberOfGenerationsFromBase(...)

| GetNumberOfGenerationsFromBase(self, type:str) -> int

| C++: vtkIdType GetNumberOfGenerationsFromBase(const char \*type)

| override;

|

| Given the name of a base class of this class type, return the

| distance of inheritance between this class type and the named

| class (how many generations of inheritance are there between this

| class and the named class). If the named class is not in this

| class's inheritance tree, return a negative value. Valid

| responses will always be nonnegative. This method works in

| combination with vtkTypeMacro found in vtkSetGet.h.

|

| GetNumberOfGenerationsFromBaseType(...)

| GetNumberOfGenerationsFromBaseType(type:str) -> int

| C++: static vtkIdType GetNumberOfGenerationsFromBaseType(

| const char \*type)

|

| Given a the name of a base class of this class type, return the

| distance of inheritance between this class type and the named

| class (how many generations of inheritance are there between this

| class and the named class). If the named class is not in this

| class's inheritance tree, return a negative value. Valid

| responses will always be nonnegative. This method works in

| combination with vtkTypeMacro found in vtkSetGet.h.

|

| IsA(...)

| IsA(self, type:str) -> int

| C++: vtkTypeBool IsA(const char \*type) override;

|

| Return 1 if this class is the same type of (or a subclass of) the

| named class. Returns 0 otherwise. This method works in

| combination with vtkTypeMacro found in vtkSetGet.h.

|

| IsTypeOf(...)

| IsTypeOf(type:str) -> int

| C++: static vtkTypeBool IsTypeOf(const char \*type)

|

| Return 1 if this class type is the same type of (or a subclass

| of) the named class. Returns 0 otherwise. This method works in

| combination with vtkTypeMacro found in vtkSetGet.h.

|

| NewInstance(...)

| NewInstance(self) -> vtkPointData

| C++: vtkPointData \*NewInstance()

|

| SafeDownCast(...)

| SafeDownCast(o:vtkObjectBase) -> vtkPointData

| C++: static vtkPointData \*SafeDownCast(vtkObjectBase \*o)

|

| \_\_delattr\_\_(self, name, /)

| Implement delattr(self, name).

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_init\_\_(self, /, \*args, \*\*kwargs)

| Initialize self. See help(type(self)) for accurate signature.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_setattr\_\_(self, name, value, /)

| Implement setattr(self, name, value).

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| ----------------------------------------------------------------------

| Data descriptors defined here:

|

| \_\_dict\_\_

| Dictionary of attributes set by user.

|

| \_\_this\_\_

| Pointer to the C++ object.

|

| ----------------------------------------------------------------------

| Data and other attributes defined here:

|

| \_\_vtkname\_\_ = 'vtkPointData'

|

| ----------------------------------------------------------------------

| Methods inherited from vtkDataSetAttributes:

|

| CopyAllOff(...)

| CopyAllOff(self, ctype:int=...) -> None

| C++: void CopyAllOff(int ctype=ALLCOPY) override;

|

| @copydoc vtkDataSetAttributes::SetCopyAttribute()

|

| CopyAllOn(...)

| CopyAllOn(self, ctype:int=...) -> None

| C++: void CopyAllOn(int ctype=ALLCOPY) override;

|

| @copydoc vtkDataSetAttributes::SetCopyAttribute()

|

| CopyAllocate(...)

| CopyAllocate(self, pd:vtkDataSetAttributes, sze:int=0,

| ext:int=1000) -> None

| C++: void CopyAllocate(vtkDataSetAttributes \*pd, vtkIdType sze=0,

| vtkIdType ext=1000)

| CopyAllocate(self, pd:vtkDataSetAttributes, sze:int, ext:int,

| shallowCopyArrays:int) -> None

| C++: void CopyAllocate(vtkDataSetAttributes \*pd, vtkIdType sze,

| vtkIdType ext, int shallowCopyArrays)

| CopyAllocate(self, list:vtkDataSetAttributesFieldList, sze:int=0,

| ext:int=1000) -> None

| C++: void CopyAllocate(vtkDataSetAttributes::FieldList &list,

| vtkIdType sze=0, vtkIdType ext=1000)

|

| Allocates point data for point-by-point (or cell-by-cell) copy

| operation. If sze=0, then use the input DataSetAttributes to

| create (i.e., find initial size of) new objects; otherwise use

| the sze variable. Note that pd HAS to be the vtkDataSetAttributes

| object which will later be used with CopyData. If this is not the

| case, consider using the alternative forms of CopyAllocate and

| CopyData. ext is no longer used. If shallowCopyArrays is true,

| input arrays are copied to the output instead of new ones being

| allocated.

|

| CopyData(...)

| CopyData(self, fromPd:vtkDataSetAttributes, fromId:int, toId:int)

| -> None

| C++: void CopyData(vtkDataSetAttributes \*fromPd, vtkIdType fromId,

| vtkIdType toId)

| CopyData(self, fromPd:vtkDataSetAttributes, fromIds:vtkIdList,

| toIds:vtkIdList) -> None

| C++: void CopyData(vtkDataSetAttributes \*fromPd,

| vtkIdList \*fromIds, vtkIdList \*toIds)

| CopyData(self, fromPd:vtkDataSetAttributes, fromIds:vtkIdList,

| destStartId:int=0) -> None

| C++: void CopyData(vtkDataSetAttributes \*fromPd,

| vtkIdList \*fromIds, vtkIdType destStartId=0)

| CopyData(self, fromPd:vtkDataSetAttributes, dstStart:int, n:int,

| srcStart:int) -> None

| C++: void CopyData(vtkDataSetAttributes \*fromPd,

| vtkIdType dstStart, vtkIdType n, vtkIdType srcStart)

| CopyData(self, list:vtkDataSetAttributesFieldList,

| dsa:vtkDataSetAttributes, idx:int, fromId:int, toId:int)

| -> None

| C++: void CopyData(vtkDataSetAttributes::FieldList &list,

| vtkDataSetAttributes \*dsa, int idx, vtkIdType fromId,

| vtkIdType toId)

| CopyData(self, list:vtkDataSetAttributesFieldList,

| dsa:vtkDataSetAttributes, idx:int, dstStart:int, n:int,

| srcStart:int) -> None

| C++: void CopyData(vtkDataSetAttributes::FieldList &list,

| vtkDataSetAttributes \*dsa, int idx, vtkIdType dstStart,

| vtkIdType n, vtkIdType srcStart)

|

| Copy the attribute data from one id to another. Make sure

| CopyAllocate() has been invoked before using this method. When

| copying a field, the following copying rules are followed: 1)

| Check if a field is an attribute, if yes and if there is a

| COPYTUPLE copy flag for that attribute (on or off), obey the flag

| for that attribute, ignore (2) and (3), 2) if there is a copy

| field for that field (on or off), obey the flag, ignore (3) 3)

| obey CopyAllOn/Off

|

| @warning This method is prone to compile-time ambiguity when

| called using `0` parameters.

| To fix the ambiguity, please replace `0` by `vtkIdType(0)`.

|

| CopyGlobalIdsOff(...)

| CopyGlobalIdsOff(self) -> None

| C++: virtual void CopyGlobalIdsOff()

|

| CopyGlobalIdsOn(...)

| CopyGlobalIdsOn(self) -> None

| C++: virtual void CopyGlobalIdsOn()

|

| CopyHigherOrderDegreesOff(...)

| CopyHigherOrderDegreesOff(self) -> None

| C++: virtual void CopyHigherOrderDegreesOff()

|

| CopyHigherOrderDegreesOn(...)

| CopyHigherOrderDegreesOn(self) -> None

| C++: virtual void CopyHigherOrderDegreesOn()

|

| CopyNormalsOff(...)

| CopyNormalsOff(self) -> None

| C++: virtual void CopyNormalsOff()

|

| CopyNormalsOn(...)

| CopyNormalsOn(self) -> None

| C++: virtual void CopyNormalsOn()

|

| CopyPedigreeIdsOff(...)

| CopyPedigreeIdsOff(self) -> None

| C++: virtual void CopyPedigreeIdsOff()

|

| CopyPedigreeIdsOn(...)

| CopyPedigreeIdsOn(self) -> None

| C++: virtual void CopyPedigreeIdsOn()

|

| CopyProcessIdsOff(...)

| CopyProcessIdsOff(self) -> None

| C++: virtual void CopyProcessIdsOff()

|

| CopyProcessIdsOn(...)

| CopyProcessIdsOn(self) -> None

| C++: virtual void CopyProcessIdsOn()

|

| CopyRationalWeightsOff(...)

| CopyRationalWeightsOff(self) -> None

| C++: virtual void CopyRationalWeightsOff()

|

| CopyRationalWeightsOn(...)

| CopyRationalWeightsOn(self) -> None

| C++: virtual void CopyRationalWeightsOn()

|

| CopyScalarsOff(...)

| CopyScalarsOff(self) -> None

| C++: virtual void CopyScalarsOff()

|

| CopyScalarsOn(...)

| CopyScalarsOn(self) -> None

| C++: virtual void CopyScalarsOn()

|

| CopyStructuredData(...)

| CopyStructuredData(self, inDsa:vtkDataSetAttributes, inExt:(int,

| ...), outExt:(int, ...), setSize:bool=True) -> None

| C++: void CopyStructuredData(vtkDataSetAttributes \*inDsa,

| const int \*inExt, const int \*outExt, bool setSize=true)

|

| This method is used to copy data arrays in images. You should

| call CopyAllocate or SetupForCopy before calling this method. If

| setSize is true, this method will set the size of the output

| arrays according to the output extent. This is required when

| CopyAllocate() was used to setup output arrays.

|

| CopyTCoordsOff(...)

| CopyTCoordsOff(self) -> None

| C++: virtual void CopyTCoordsOff()

|

| CopyTCoordsOn(...)

| CopyTCoordsOn(self) -> None

| C++: virtual void CopyTCoordsOn()

|

| CopyTangentsOff(...)

| CopyTangentsOff(self) -> None

| C++: virtual void CopyTangentsOff()

|

| CopyTangentsOn(...)

| CopyTangentsOn(self) -> None

| C++: virtual void CopyTangentsOn()

|

| CopyTensorsOff(...)

| CopyTensorsOff(self) -> None

| C++: virtual void CopyTensorsOff()

|

| CopyTensorsOn(...)

| CopyTensorsOn(self) -> None

| C++: virtual void CopyTensorsOn()

|

| CopyTuple(...)

| CopyTuple(self, fromData:vtkAbstractArray,

| toData:vtkAbstractArray, fromId:int, toId:int) -> None

| C++: void CopyTuple(vtkAbstractArray \*fromData,

| vtkAbstractArray \*toData, vtkIdType fromId, vtkIdType toId)

|

| Copy a tuple (or set of tuples) of data from one data array to

| another. This method assumes that the fromData and toData objects

| are of the same type, and have the same number of components.

| This is true if you invoke CopyAllocate() or

| InterpolateAllocate().

|

| CopyTuples(...)

| CopyTuples(self, fromData:vtkAbstractArray,

| toData:vtkAbstractArray, fromIds:vtkIdList, toIds:vtkIdList)

| -> None

| C++: void CopyTuples(vtkAbstractArray \*fromData,

| vtkAbstractArray \*toData, vtkIdList \*fromIds,

| vtkIdList \*toIds)

| CopyTuples(self, fromData:vtkAbstractArray,

| toData:vtkAbstractArray, dstStart:int, n:int, srcStart:int)

| -> None

| C++: void CopyTuples(vtkAbstractArray \*fromData,

| vtkAbstractArray \*toData, vtkIdType dstStart, vtkIdType n,

| vtkIdType srcStart)

|

| CopyVectorsOff(...)

| CopyVectorsOff(self) -> None

| C++: virtual void CopyVectorsOff()

|

| CopyVectorsOn(...)

| CopyVectorsOn(self) -> None

| C++: virtual void CopyVectorsOn()

|

| DeepCopy(...)

| DeepCopy(self, pd:vtkFieldData) -> None

| C++: void DeepCopy(vtkFieldData \*pd) override;

|

| Deep copy of data (i.e., create new data arrays and copy from

| input data). Ignores the copy flags but preserves them in the

| output.

|

| GetAbstractAttribute(...)

| GetAbstractAttribute(self, attributeType:int) -> vtkAbstractArray

| C++: vtkAbstractArray \*GetAbstractAttribute(int attributeType)

|

| Return an attribute given the attribute type (see

| vtkDataSetAttributes::AttributeTypes). This is the same as

| GetAttribute(), except that the returned array is a

| vtkAbstractArray instead of vtkDataArray. Some attributes (such

| as PEDIGREEIDS) may not be vtkDataArray subclass.

|

| GetAttribute(...)

| GetAttribute(self, attributeType:int) -> vtkDataArray

| C++: vtkDataArray \*GetAttribute(int attributeType)

|

| Return an attribute given the attribute type (see

| vtkDataSetAttributes::AttributeTypes). Some attributes (such as

| PEDIGREEIDS) may not be vtkDataArray subclass, so in that case

| use GetAbstractAttribute().

|

| GetAttributeIndices(...)

| GetAttributeIndices(self, indexArray:[int, ...]) -> None

| C++: void GetAttributeIndices(int \*indexArray)

|

| Get the field data array indices corresponding to scalars,

| vectors, tensors, etc. The given buffer must be at least

| NUM\_ATTRIBUTES elements big.

|

| GetAttributeTypeAsString(...)

| GetAttributeTypeAsString(attributeType:int) -> str

| C++: static const char \*GetAttributeTypeAsString(

| int attributeType)

|

| Given an integer attribute type, this static method returns a

| string type for the attribute (i.e. type = 0: returns "Scalars").

|

| GetCopyAttribute(...)

| GetCopyAttribute(self, index:int, ctype:int) -> int

| C++: int GetCopyAttribute(int index, int ctype)

|

| Get the attribute copy flag for copy operation ctype of

| attributeindex.

|

| GetCopyGlobalIds(...)

| GetCopyGlobalIds(self, ctype:int=...) -> int

| C++: vtkTypeBool GetCopyGlobalIds(int ctype=ALLCOPY)

|

| GetCopyHigherOrderDegrees(...)

| GetCopyHigherOrderDegrees(self, ctype:int=...) -> int

| C++: vtkTypeBool GetCopyHigherOrderDegrees(int ctype=ALLCOPY)

|

| GetCopyNormals(...)

| GetCopyNormals(self, ctype:int=...) -> int

| C++: vtkTypeBool GetCopyNormals(int ctype=ALLCOPY)

|

| GetCopyPedigreeIds(...)

| GetCopyPedigreeIds(self, ctype:int=...) -> int

| C++: vtkTypeBool GetCopyPedigreeIds(int ctype=ALLCOPY)

|

| GetCopyProcessIds(...)

| GetCopyProcessIds(self, ctype:int=...) -> int

| C++: vtkTypeBool GetCopyProcessIds(int ctype=ALLCOPY)

|

| GetCopyRationalWeights(...)

| GetCopyRationalWeights(self, ctype:int=...) -> int

| C++: vtkTypeBool GetCopyRationalWeights(int ctype=ALLCOPY)

|

| GetCopyScalars(...)

| GetCopyScalars(self, ctype:int=...) -> int

| C++: vtkTypeBool GetCopyScalars(int ctype=ALLCOPY)

|

| GetCopyTCoords(...)

| GetCopyTCoords(self, ctype:int=...) -> int

| C++: vtkTypeBool GetCopyTCoords(int ctype=ALLCOPY)

|

| GetCopyTangents(...)

| GetCopyTangents(self, ctype:int=...) -> int

| C++: vtkTypeBool GetCopyTangents(int ctype=ALLCOPY)

|

| GetCopyTensors(...)

| GetCopyTensors(self, ctype:int=...) -> int

| C++: vtkTypeBool GetCopyTensors(int ctype=ALLCOPY)

|

| GetCopyVectors(...)

| GetCopyVectors(self, ctype:int=...) -> int

| C++: vtkTypeBool GetCopyVectors(int ctype=ALLCOPY)

|

| GetGlobalIds(...)

| GetGlobalIds(self) -> vtkDataArray

| C++: vtkDataArray \*GetGlobalIds()

| GetGlobalIds(self, name:str) -> vtkDataArray

| C++: vtkDataArray \*GetGlobalIds(const char \*name)

|

| GetHigherOrderDegrees(...)

| GetHigherOrderDegrees(self) -> vtkDataArray

| C++: vtkDataArray \*GetHigherOrderDegrees()

| GetHigherOrderDegrees(self, name:str) -> vtkDataArray

| C++: vtkDataArray \*GetHigherOrderDegrees(const char \*name)

|

| GetLongAttributeTypeAsString(...)

| GetLongAttributeTypeAsString(attributeType:int) -> str

| C++: static const char \*GetLongAttributeTypeAsString(

| int attributeType)

|

| GetNormals(...)

| GetNormals(self) -> vtkDataArray

| C++: vtkDataArray \*GetNormals()

| GetNormals(self, name:str) -> vtkDataArray

| C++: vtkDataArray \*GetNormals(const char \*name)

|

| GetPedigreeIds(...)

| GetPedigreeIds(self) -> vtkAbstractArray

| C++: vtkAbstractArray \*GetPedigreeIds()

| GetPedigreeIds(self, name:str) -> vtkAbstractArray

| C++: vtkAbstractArray \*GetPedigreeIds(const char \*name)

|

| GetProcessIds(...)

| GetProcessIds(self) -> vtkDataArray

| C++: vtkDataArray \*GetProcessIds()

| GetProcessIds(self, name:str) -> vtkDataArray

| C++: vtkDataArray \*GetProcessIds(const char \*name)

|

| GetRationalWeights(...)

| GetRationalWeights(self) -> vtkDataArray

| C++: vtkDataArray \*GetRationalWeights()

| GetRationalWeights(self, name:str) -> vtkDataArray

| C++: vtkDataArray \*GetRationalWeights(const char \*name)

|

| GetScalars(...)

| GetScalars(self) -> vtkDataArray

| C++: vtkDataArray \*GetScalars()

| GetScalars(self, name:str) -> vtkDataArray

| C++: vtkDataArray \*GetScalars(const char \*name)

|

| GetTCoords(...)

| GetTCoords(self) -> vtkDataArray

| C++: vtkDataArray \*GetTCoords()

| GetTCoords(self, name:str) -> vtkDataArray

| C++: vtkDataArray \*GetTCoords(const char \*name)

|

| GetTangents(...)

| GetTangents(self) -> vtkDataArray

| C++: vtkDataArray \*GetTangents()

| GetTangents(self, name:str) -> vtkDataArray

| C++: vtkDataArray \*GetTangents(const char \*name)

|

| GetTensors(...)

| GetTensors(self) -> vtkDataArray

| C++: vtkDataArray \*GetTensors()

| GetTensors(self, name:str) -> vtkDataArray

| C++: vtkDataArray \*GetTensors(const char \*name)

|

| GetVectors(...)

| GetVectors(self) -> vtkDataArray

| C++: vtkDataArray \*GetVectors()

| GetVectors(self, name:str) -> vtkDataArray

| C++: vtkDataArray \*GetVectors(const char \*name)

|

| GhostArrayName(...)

| GhostArrayName() -> str

| C++: static const char \*GhostArrayName()

|

| Initialize(...)

| Initialize(self) -> None

| C++: void Initialize() override;

|

| Initialize all of the object's data to nullptr Also, clear the

| copy flags.

|

| InterpolateAllocate(...)

| InterpolateAllocate(self, pd:vtkDataSetAttributes, sze:int=0,

| ext:int=1000) -> None

| C++: void InterpolateAllocate(vtkDataSetAttributes \*pd,

| vtkIdType sze=0, vtkIdType ext=1000)

| InterpolateAllocate(self, pd:vtkDataSetAttributes, sze:int,

| ext:int, shallowCopyArrays:int) -> None

| C++: void InterpolateAllocate(vtkDataSetAttributes \*pd,

| vtkIdType sze, vtkIdType ext, int shallowCopyArrays)

| InterpolateAllocate(self, list:vtkDataSetAttributesFieldList,

| sze:int=0, ext:int=1000) -> None

| C++: void InterpolateAllocate(

| vtkDataSetAttributes::FieldList &list, vtkIdType sze=0,

| vtkIdType ext=1000)

|

| Initialize point interpolation method. Note that pd HAS to be the

| vtkDataSetAttributes object which will later be used with

| InterpolatePoint or InterpolateEdge. ext is no longer used. If

| shallowCopyArrays is true, input arrays are copied to the output

| instead of new ones being allocated.

|

| InterpolateEdge(...)

| InterpolateEdge(self, fromPd:vtkDataSetAttributes, toId:int,

| p1:int, p2:int, t:float) -> None

| C++: void InterpolateEdge(vtkDataSetAttributes \*fromPd,

| vtkIdType toId, vtkIdType p1, vtkIdType p2, double t)

|

| Interpolate data from the two points p1,p2 (forming an edge) and

| an interpolation factor, t, along the edge. The weight ranges

| from (0,1), with t=0 located at p1. Make sure that the method

| InterpolateAllocate() has been invoked before using this method.

| If the INTERPOLATION copy flag is set to 0 for an array,

| interpolation is prevented. If the flag is set to 1, weighted

| interpolation occurs. If the flag is set to 2, nearest neighbor

| interpolation is used.

|

| InterpolatePoint(...)

| InterpolatePoint(self, fromPd:vtkDataSetAttributes, toId:int,

| ids:vtkIdList, weights:[float, ...]) -> None

| C++: void InterpolatePoint(vtkDataSetAttributes \*fromPd,

| vtkIdType toId, vtkIdList \*ids, double \*weights)

| InterpolatePoint(self, list:vtkDataSetAttributesFieldList,

| fromPd:vtkDataSetAttributes, idx:int, toId:int, ids:vtkIdList,

| weights:[float, ...]) -> None

| C++: void InterpolatePoint(vtkDataSetAttributes::FieldList &list,

| vtkDataSetAttributes \*fromPd, int idx, vtkIdType toId,

| vtkIdList \*ids, double \*weights)

|

| Interpolate data set attributes from other data set attributes

| given cell or point ids and associated interpolation weights. If

| the INTERPOLATION copy flag is set to 0 for an array,

| interpolation is prevented. If the flag is set to 1, weighted

| interpolation occurs. If the flag is set to 2, nearest neighbor

| interpolation is used.

|

| InterpolateTime(...)

| InterpolateTime(self, from1:vtkDataSetAttributes,

| from2:vtkDataSetAttributes, id:int, t:float) -> None

| C++: void InterpolateTime(vtkDataSetAttributes \*from1,

| vtkDataSetAttributes \*from2, vtkIdType id, double t)

|

| Interpolate data from the same id (point or cell) at different

| points in time (parameter t). Two input data set attributes

| objects are input. The parameter t lies between (0<=t<=1).

| IMPORTANT: it is assumed that the number of attributes and number

| of components is the same for both from1 and from2, and the type

| of data for from1 and from2 are the same. Make sure that the

| method InterpolateAllocate() has been invoked before using this

| method. If the INTERPOLATION copy flag is set to 0 for an array,

| interpolation is prevented. If the flag is set to 1, weighted

| interpolation occurs. If the flag is set to 2, nearest neighbor

| interpolation is used.

|

| IsArrayAnAttribute(...)

| IsArrayAnAttribute(self, idx:int) -> int

| C++: int IsArrayAnAttribute(int idx)

|

| Determine whether a data array of index idx is considered a data

| set attribute (i.e., scalar, vector, tensor, etc). Return

| less-than zero if it is, otherwise an index 0<=idx<NUM\_ATTRIBUTES

| to indicate which attribute.

|

| PassData(...)

| PassData(self, fd:vtkFieldData) -> None

| C++: void PassData(vtkFieldData \*fd) override;

|

| Pass entire arrays of input data through to output. Obey the

| "copy" flags. When passing a field, the following copying rules

| are followed: 1) Check if a field is an attribute, if yes and if

| there is a PASSDATA copy flag for that attribute (on or off),

| obey the flag for that attribute, ignore (2) and (3), 2) if there

| is a copy field for that field (on or off), obey the flag, ignore

| (3) 3) obey CopyAllOn/Off

|

| RemoveArray(...)

| RemoveArray(self, index:int) -> None

| C++: void RemoveArray(int index) override;

| RemoveArray(self, name:str) -> None

| C++: virtual void RemoveArray(const char \*name)

|

| Remove an array (with the given index) from the list of arrays.

|

| SetActiveAttribute(...)

| SetActiveAttribute(self, name:str, attributeType:int) -> int

| C++: int SetActiveAttribute(const char \*name, int attributeType)

| SetActiveAttribute(self, index:int, attributeType:int) -> int

| C++: int SetActiveAttribute(int index, int attributeType)

|

| Make the array with the given name the active attribute.

| Attribute types are: vtkDataSetAttributes::SCALARS = 0

| vtkDataSetAttributes::VECTORS = 1 vtkDataSetAttributes::NORMALS =

| 2 vtkDataSetAttributes::TCOORDS = 3 vtkDataSetAttributes::TENSORS

| = 4 vtkDataSetAttributes::GLOBALIDS = 5

| vtkDataSetAttributes::PEDIGREEIDS = 6

| vtkDataSetAttributes::EDGEFLAG = 7 vtkDataSetAttributes::TANGENTS

| = 8 vtkDataSetAttributes::RATIONALWEIGHTS = 9

| vtkDataSetAttributes::HIGHERORDERDEGREES = 10

| vtkDataSetAttributes::PROCESSIDS = 11 Returns the index of the

| array if successful, -1 if the array is not in the list of

| arrays.

|

| SetActiveGlobalIds(...)

| SetActiveGlobalIds(self, name:str) -> int

| C++: int SetActiveGlobalIds(const char \*name)

|

| SetActiveHigherOrderDegrees(...)

| SetActiveHigherOrderDegrees(self, name:str) -> int

| C++: int SetActiveHigherOrderDegrees(const char \*name)

|

| SetActiveNormals(...)

| SetActiveNormals(self, name:str) -> int

| C++: int SetActiveNormals(const char \*name)

|

| SetActivePedigreeIds(...)

| SetActivePedigreeIds(self, name:str) -> int

| C++: int SetActivePedigreeIds(const char \*name)

|

| SetActiveProcessIds(...)

| SetActiveProcessIds(self, name:str) -> int

| C++: int SetActiveProcessIds(const char \*name)

|

| SetActiveRationalWeights(...)

| SetActiveRationalWeights(self, name:str) -> int

| C++: int SetActiveRationalWeights(const char \*name)

|

| SetActiveScalars(...)

| SetActiveScalars(self, name:str) -> int

| C++: int SetActiveScalars(const char \*name)

|

| SetActiveTCoords(...)

| SetActiveTCoords(self, name:str) -> int

| C++: int SetActiveTCoords(const char \*name)

|

| SetActiveTangents(...)

| SetActiveTangents(self, name:str) -> int

| C++: int SetActiveTangents(const char \*name)

|

| SetActiveTensors(...)

| SetActiveTensors(self, name:str) -> int

| C++: int SetActiveTensors(const char \*name)

|

| SetActiveVectors(...)

| SetActiveVectors(self, name:str) -> int

| C++: int SetActiveVectors(const char \*name)

|

| SetAttribute(...)

| SetAttribute(self, aa:vtkAbstractArray, attributeType:int) -> int

| C++: int SetAttribute(vtkAbstractArray \*aa, int attributeType)

|

| Set an array to use as the given attribute type (i.e.,

| vtkDataSetAttributes::SCALAR, vtkDataSetAttributes::VECTOR,

| vtkDataSetAttributes::TENSOR, etc.). If this attribute was

| previously set to another array, that array is removed from the

| vtkDataSetAttributes object and the array aa is used as the

| attribute.

|

| \* Returns the index of aa within the vtkDataSetAttributes object

| \* (i.e., the index to pass to the method GetArray(int) to obtain

| \* aa) if the attribute was set to aa successfully. If aa was

| \* already set as the given attributeType, returns the index of

| \* aa.

|

| \* Returns -1 in the following cases:

|

| \* - aa is nullptr (used to unset an attribute; not an error

| indicator)

| \* - aa is not a subclass of vtkDataArray, unless the

| attributeType

| \* is vtkDataSetAttributes::PEDIGREEIDS (error indicator)

| \* - aa has a number of components incompatible with the attribute

| type

| \* (error indicator)

|

| SetCopyAttribute(...)

| SetCopyAttribute(self, index:int, value:int, ctype:int=...)

| -> None

| C++: void SetCopyAttribute(int index, int value,

| int ctype=ALLCOPY)

|

| Turn on/off the copying of attribute data. ctype is one of the

| AttributeCopyOperations, and controls copy, interpolate and

| passdata behavior. For set, ctype=ALLCOPY means set all three

| flags to the same value. For get, ctype=ALLCOPY returns true only

| if all three flags are true.

|

| \* During copying, interpolation and passdata, the following rules

| are

| \* followed for each array:

| \* 1. If the copy/interpolate/pass flag for an attribute is set

| (on or off),

| \* it is applied. This overrides rules 2 and 3.

| \* 2. If the copy flag for an array is set (on or off), it is

| applied

| \* This overrides rule 3.

| \* 3. If CopyAllOn is set, copy the array.

| \* If CopyAllOff is set, do not copy the array

|

| \* For interpolation, the flag values can be as follows:

| \* 0: Do not interpolate.

| \* 1: Weighted interpolation.

| \* 2: Nearest neighbor interpolation.

|

| SetCopyGlobalIds(...)

| SetCopyGlobalIds(self, i:int, ctype:int=...) -> None

| C++: void SetCopyGlobalIds(vtkTypeBool i, int ctype=ALLCOPY)

|

| @copydoc vtkDataSetAttributes::SetCopyAttribute()

|

| SetCopyHigherOrderDegrees(...)

| SetCopyHigherOrderDegrees(self, i:int, ctype:int=...) -> None

| C++: void SetCopyHigherOrderDegrees(vtkTypeBool i,

| int ctype=ALLCOPY)

|

| @copydoc vtkDataSetAttributes::SetCopyAttribute()

|

| SetCopyNormals(...)

| SetCopyNormals(self, i:int, ctype:int=...) -> None

| C++: void SetCopyNormals(vtkTypeBool i, int ctype=ALLCOPY)

|

| @copydoc vtkDataSetAttributes::SetCopyAttribute()

|

| SetCopyPedigreeIds(...)

| SetCopyPedigreeIds(self, i:int, ctype:int=...) -> None

| C++: void SetCopyPedigreeIds(vtkTypeBool i, int ctype=ALLCOPY)

|

| @copydoc vtkDataSetAttributes::SetCopyAttribute()

|

| SetCopyProcessIds(...)

| SetCopyProcessIds(self, i:int, ctype:int=...) -> None

| C++: void SetCopyProcessIds(vtkTypeBool i, int ctype=ALLCOPY)

|

| @copydoc vtkDataSetAttributes::SetCopyAttribute()

|

| SetCopyRationalWeights(...)

| SetCopyRationalWeights(self, i:int, ctype:int=...) -> None

| C++: void SetCopyRationalWeights(vtkTypeBool i, int ctype=ALLCOPY)

|

| @copydoc vtkDataSetAttributes::SetCopyAttribute()

|

| SetCopyScalars(...)

| SetCopyScalars(self, i:int, ctype:int=...) -> None

| C++: void SetCopyScalars(vtkTypeBool i, int ctype=ALLCOPY)

|

| @copydoc vtkDataSetAttributes::SetCopyAttribute()

|

| SetCopyTCoords(...)

| SetCopyTCoords(self, i:int, ctype:int=...) -> None

| C++: void SetCopyTCoords(vtkTypeBool i, int ctype=ALLCOPY)

|

| @copydoc vtkDataSetAttributes::SetCopyAttribute()

|

| SetCopyTangents(...)

| SetCopyTangents(self, i:int, ctype:int=...) -> None

| C++: void SetCopyTangents(vtkTypeBool i, int ctype=ALLCOPY)

|

| @copydoc vtkDataSetAttributes::SetCopyAttribute()

|

| SetCopyTensors(...)

| SetCopyTensors(self, i:int, ctype:int=...) -> None

| C++: void SetCopyTensors(vtkTypeBool i, int ctype=ALLCOPY)

|

| @copydoc vtkDataSetAttributes::SetCopyAttribute()

|

| SetCopyVectors(...)

| SetCopyVectors(self, i:int, ctype:int=...) -> None

| C++: void SetCopyVectors(vtkTypeBool i, int ctype=ALLCOPY)

|

| @copydoc vtkDataSetAttributes::SetCopyAttribute()

|

| SetGlobalIds(...)

| SetGlobalIds(self, da:vtkDataArray) -> int

| C++: int SetGlobalIds(vtkDataArray \*da)

|

| Set/Get the global id data.

|

| SetHigherOrderDegrees(...)

| SetHigherOrderDegrees(self, da:vtkDataArray) -> int

| C++: int SetHigherOrderDegrees(vtkDataArray \*da)

|

| Set/Get the rational degrees data.

|

| SetNormals(...)

| SetNormals(self, da:vtkDataArray) -> int

| C++: int SetNormals(vtkDataArray \*da)

|

| Set/get the normal data.

|

| SetPedigreeIds(...)

| SetPedigreeIds(self, da:vtkAbstractArray) -> int

| C++: int SetPedigreeIds(vtkAbstractArray \*da)

|

| Set/Get the pedigree id data.

|

| SetProcessIds(...)

| SetProcessIds(self, da:vtkDataArray) -> int

| C++: int SetProcessIds(vtkDataArray \*da)

|

| Set/Get the process id data.

|

| SetRationalWeights(...)

| SetRationalWeights(self, da:vtkDataArray) -> int

| C++: int SetRationalWeights(vtkDataArray \*da)

|

| Set/Get the rational weights data.

|

| SetScalars(...)

| SetScalars(self, da:vtkDataArray) -> int

| C++: int SetScalars(vtkDataArray \*da)

|

| Set/Get the scalar data.

|

| SetTCoords(...)

| SetTCoords(self, da:vtkDataArray) -> int

| C++: int SetTCoords(vtkDataArray \*da)

|

| Set/Get the texture coordinate data.

|

| SetTangents(...)

| SetTangents(self, da:vtkDataArray) -> int

| C++: int SetTangents(vtkDataArray \*da)

|

| Set/get the tangent data.

|

| SetTensors(...)

| SetTensors(self, da:vtkDataArray) -> int

| C++: int SetTensors(vtkDataArray \*da)

|

| Set/Get the tensor data.

|

| SetVectors(...)

| SetVectors(self, da:vtkDataArray) -> int

| C++: int SetVectors(vtkDataArray \*da)

|

| Set/Get the vector data.

|

| SetupForCopy(...)

| SetupForCopy(self, pd:vtkDataSetAttributes) -> None

| C++: void SetupForCopy(vtkDataSetAttributes \*pd)

|

| Create a mapping between the input attributes and this object so

| that methods like CopyData() and CopyStructuredData() can be

| called. This method assumes that this object has the same arrays

| as the input and that they are ordered the same way (same array

| indices).

|

| ShallowCopy(...)

| ShallowCopy(self, pd:vtkFieldData) -> None

| C++: void ShallowCopy(vtkFieldData \*pd) override;

|

| Shallow copy of data (i.e., use reference counting). Ignores the

| copy flags but preserves them in the output.

|

| Update(...)

| Update(self) -> None

| C++: virtual void Update()

|

| Attributes have a chance to bring themselves up to date; right

| now this is ignored.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from vtkDataSetAttributes:

|

| copy\_attribute

| write-only, Calls SetCopyAttribute

|

| copy\_global\_ids

| write-only, Calls SetCopyGlobalIds

|

| copy\_higher\_order\_degrees

| write-only, Calls SetCopyHigherOrderDegrees

|

| copy\_normals

| write-only, Calls SetCopyNormals

|

| copy\_pedigree\_ids

| write-only, Calls SetCopyPedigreeIds

|

| copy\_process\_ids

| write-only, Calls SetCopyProcessIds

|

| copy\_rational\_weights

| write-only, Calls SetCopyRationalWeights

|

| copy\_scalars

| write-only, Calls SetCopyScalars

|

| copy\_t\_coords

| write-only, Calls SetCopyTCoords

|

| copy\_tangents

| write-only, Calls SetCopyTangents

|

| copy\_tensors

| write-only, Calls SetCopyTensors

|

| copy\_vectors

| write-only, Calls SetCopyVectors

|

| global\_ids

| read-only, Calls GetGlobalIds

|

| higher\_order\_degrees

| read-only, Calls GetHigherOrderDegrees

|

| normals

| read-only, Calls GetNormals

|

| pedigree\_ids

| read-only, Calls GetPedigreeIds

|

| process\_ids

| read-only, Calls GetProcessIds

|

| rational\_weights

| read-only, Calls GetRationalWeights

|

| scalars

| read-only, Calls GetScalars

|

| t\_coords

| read-only, Calls GetTCoords

|

| tangents

| read-only, Calls GetTangents

|

| tensors

| read-only, Calls GetTensors

|

| vectors

| read-only, Calls GetVectors

|

| ----------------------------------------------------------------------

| Data and other attributes inherited from vtkDataSetAttributes:

|

| ALLCOPY = 3

|

| AttributeCopyOperations = <class 'vtkmodules.vtkCommonDataModel.vtkDat...

|

| AttributeLimitTypes = <class 'vtkmodules.vtkCommonDataModel.vtkDataSet...

|

| AttributeTypes = <class 'vtkmodules.vtkCommonDataModel.vtkDataSetAttri...

|

| COPYTUPLE = 0

|

| CellGhostTypes = <class 'vtkmodules.vtkCommonDataModel.vtkDataSetAttri...

|

| DUPLICATECELL = 1

|

| DUPLICATEPOINT = 1

|

| EDGEFLAG = 7

|

| EXACT = 1

|

| EXTERIORCELL = 16

|

| GLOBALIDS = 5

|

| HIDDENCELL = 32

|

| HIDDENPOINT = 2

|

| HIGHCONNECTIVITYCELL = 2

|

| HIGHERORDERDEGREES = 10

|

| INTERPOLATE = 1

|

| LOWCONNECTIVITYCELL = 4

|

| MAX = 0

|

| NOLIMIT = 2

|

| NORMALS = 2

|

| NUM\_ATTRIBUTES = 12

|

| PASSDATA = 2

|

| PEDIGREEIDS = 6

|

| PROCESSIDS = 11

|

| PointGhostTypes = <class 'vtkmodules.vtkCommonDataModel.vtkDataSetAttr...

|

| RATIONALWEIGHTS = 9

|

| REFINEDCELL = 8

|

| SCALARS = 0

|

| TANGENTS = 8

|

| TCOORDS = 3

|

| TENSORS = 4

|

| VECTORS = 1

|

| ----------------------------------------------------------------------

| Methods inherited from vtkFieldData:

|

| AddArray(...)

| AddArray(self, array:vtkAbstractArray) -> int

| C++: int AddArray(vtkAbstractArray \*array)

|

| Add an array to the array list. If an array with the same name

| already exists - then the added array will replace it. Return the

| index of the added array. If the given array is nullptr, does

| nothing and returns -1.

|

| Allocate(...)

| Allocate(self, sz:int, ext:int=1000) -> int

| C++: vtkTypeBool Allocate(vtkIdType sz, vtkIdType ext=1000)

|

| Allocate data for each array. Note that ext is no longer used.

|

| AllocateArrays(...)

| AllocateArrays(self, num:int) -> None

| C++: void AllocateArrays(int num)

|

| AllocateArrays actually sets the number of vtkAbstractArray

| pointers in the vtkFieldData object, not the number of used

| pointers (arrays). Adding more arrays will cause the object to

| dynamically adjust the number of pointers if it needs to extend.

| Although AllocateArrays can be used if the number of arrays which

| will be added is known, it can be omitted with a small

| computation cost.

|

| CopyFieldOff(...)

| CopyFieldOff(self, name:str) -> None

| C++: void CopyFieldOff(const char \*name)

|

| CopyFieldOn(...)

| CopyFieldOn(self, name:str) -> None

| C++: void CopyFieldOn(const char \*name)

|

| Turn on/off the copying of the field specified by name. During

| the copying/passing, the following rules are followed for each

| array:

| 1. If the copy flag for an array is set (on or off), it is

| applied. This overrides rule 2.

| 2. If CopyAllOn is set, copy the array. If CopyAllOff is set, do

| not copy the array.

|

| CopyStructure(...)

| CopyStructure(self, \_\_a:vtkFieldData) -> None

| C++: void CopyStructure(vtkFieldData \*)

|

| Copy data array structure from a given field. The same arrays

| will exist with the same types, but will contain nothing in the

| copy.

|

| GetAbstractArray(...)

| GetAbstractArray(self, i:int) -> vtkAbstractArray

| C++: vtkAbstractArray \*GetAbstractArray(int i)

| GetAbstractArray(self, arrayName:str, index:int)

| -> vtkAbstractArray

| C++: vtkAbstractArray \*GetAbstractArray(const char \*arrayName,

| int &index)

| GetAbstractArray(self, arrayName:str) -> vtkAbstractArray

| C++: vtkAbstractArray \*GetAbstractArray(const char \*arrayName)

|

| Returns the ith array in the field. Unlike GetArray(), this

| method returns a vtkAbstractArray and can be used to access any

| array type. A nullptr is returned only if the index i is out of

| range.

|

| GetActualMemorySize(...)

| GetActualMemorySize(self) -> int

| C++: virtual unsigned long GetActualMemorySize()

|

| Return the memory in kibibytes (1024 bytes) consumed by this

| field data. Used to support streaming and reading/writing data.

| The value returned is guaranteed to be greater than or equal to

| the memory required to actually represent the data represented by

| this object.

|

| GetArray(...)

| GetArray(self, i:int) -> vtkDataArray

| C++: vtkDataArray \*GetArray(int i)

| GetArray(self, arrayName:str, index:int) -> vtkDataArray

| C++: vtkDataArray \*GetArray(const char \*arrayName, int &index)

| GetArray(self, arrayName:str) -> vtkDataArray

| C++: vtkDataArray \*GetArray(const char \*arrayName)

|

| Not recommended for use. Use GetAbstractArray(int i) instead.

|

| Return the ith array in the field. A nullptr is returned if the

| index i is out of range, or if the array at the given index is

| not a vtkDataArray. To access vtkStringArray, or vtkVariantArray,

| use GetAbstractArray(int i).

|

| GetArrayContainingComponent(...)

| GetArrayContainingComponent(self, i:int, arrayComp:int) -> int

| C++: int GetArrayContainingComponent(int i, int &arrayComp)

|

| Return the array containing the ith component of the field. The

| return value is an integer number n 0<=n<this->NumberOfArrays.

| Also, an integer value is returned indicating the component in

| the array is returned. Method returns -1 if specified component

| is not in the field.

|

| GetArrayName(...)

| GetArrayName(self, i:int) -> str

| C++: const char \*GetArrayName(int i)

|

| Get the name of ith array. Note that this is equivalent to:

| GetAbstractArray(i)->GetName() if ith array pointer is not

| nullptr

|

| GetField(...)

| GetField(self, ptId:vtkIdList, f:vtkFieldData) -> None

| C++: void GetField(vtkIdList \*ptId, vtkFieldData \*f)

|

| Get a field from a list of ids. Supplied field f should have same

| types and number of data arrays as this one (i.e., like

| CopyStructure() creates). This method should not be used if the

| instance is from a subclass of vtkFieldData (vtkPointData or

| vtkCellData). This is because in those cases, the attribute data

| is stored with the other fields and will cause the method to

| behave in an unexpected way.

|

| GetFiniteRange(...)

| GetFiniteRange(self, name:str, range:[float, float], comp:int=0)

| -> bool

| C++: bool GetFiniteRange(const char \*name, double range[2],

| int comp=0)

| GetFiniteRange(self, index:int, range:[float, float], comp:int=0)

| -> bool

| C++: bool GetFiniteRange(int index, double range[2], int comp=0)

|

| GetGhostArray(...)

| GetGhostArray(self) -> vtkUnsignedCharArray

| C++: virtual vtkUnsignedCharArray \*GetGhostArray()

|

| Get the ghost array, if present in this field data. If no ghost

| array is set, returns `nullptr`. A ghost array is a

| `vtkUnsignedCharArray` called `vtkGhostType`. See

| `vtkDataSetAttributes` for more context on ghost types.

|

| @sa

| vtkDataSetAttributes

|

| GetGhostsToSkip(...)

| GetGhostsToSkip(self) -> int

| C++: virtual unsigned char GetGhostsToSkip()

|

| Set / Get the binary mask filtering out certain types of ghosts

| when calling `GetRange`. By default, it is set to 0xff for pure

| `vtkFieldData`. In `vtkCellData`, it is set to `HIDDENCELL` and

| in `vtkPointData`, it is set to `HIDDENPOINT` by default. See

| `vtkDataSetAttributes` for more context on ghost types

| definitions.

|

| @sa

| vtkDataSetAttributes vtkPointData vtkCellData

|

| GetMTime(...)

| GetMTime(self) -> int

| C++: vtkMTimeType GetMTime() override;

|

| Check object's components for modified times.

|

| GetNumberOfArrays(...)

| GetNumberOfArrays(self) -> int

| C++: int GetNumberOfArrays()

|

| Get the number of arrays of data available. This does not include

| nullptr array pointers therefore after fd->AllocateArray(n);

| nArrays = GetNumberOfArrays(); nArrays is not necessarily equal

| to n.

|

| GetNumberOfComponents(...)

| GetNumberOfComponents(self) -> int

| C++: int GetNumberOfComponents()

|

| Get the number of components in the field. This is determined by

| adding up the components in each non-nullptr array. This method

| should not be used if the instance is from a subclass of

| vtkFieldData (vtkPointData or vtkCellData). This is because in

| those cases, the attribute data is stored with the other fields

| and will cause the method to behave in an unexpected way.

|

| GetNumberOfTuples(...)

| GetNumberOfTuples(self) -> int

| C++: vtkIdType GetNumberOfTuples()

|

| Get the number of tuples in the field. Note: some fields have

| arrays with different numbers of tuples; this method returns the

| number of tuples in the first array. Mixed-length arrays may have

| to be treated specially. This method should not be used if the

| instance is from a subclass of vtkFieldData (vtkPointData or

| vtkCellData). This is because in those cases, the attribute data

| is stored with the other fields and will cause the method to

| behave in an unexpected way.

|

| GetRange(...)

| GetRange(self, name:str, range:[float, float], comp:int=0) -> bool

| C++: bool GetRange(const char \*name, double range[2], int comp=0)

| GetRange(self, index:int, range:[float, float], comp:int=0)

| -> bool

| C++: bool GetRange(int index, double range[2], int comp=0)

|

| Computes the range of the input data array (specified through its

| `name` or the `index` in this field data). If the targeted array

| is not polymorphic with a `vtkDataArray`, or if no array match

| the input `name` or `index`, or if `comp` is out of bounds, then

| the returned range is `[NaN, NaN]`.

|

| The computed range is cached to avoid recomputing it. The range

| is recomputed if the held array has been modified, if

| `GhostsToSkip` has been changed, or if the ghost array has been

| changed / modified.

|

| If a ghost array is present in the field data, then the binary

| mask `GhostsToSkip` is used to skip values associated with a

| ghost that intersects this mask.

|

| `comp` targets which component of the array the range is to be

| computed on. Setting it to -1 results in computing the range of

| the magnitude of the array.

|

| The `Finite` version of this method skips infinite values in the

| array in addition to ghosts matching with `GhostsToSkip`.

|

| HasAnyGhostBitSet(...)

| HasAnyGhostBitSet(self, bitFlag:int) -> bool

| C++: bool HasAnyGhostBitSet(int bitFlag)

|

| Helper function that tests if any of the values in ghost array

| has been set. The test performed is (value & bitFlag).

|

| HasArray(...)

| HasArray(self, name:str) -> int

| C++: vtkTypeBool HasArray(const char \*name)

|

| Return 1 if an array with the given name could be found. 0

| otherwise.

|

| InsertNextTuple(...)

| InsertNextTuple(self, j:int, source:vtkFieldData) -> int

| C++: vtkIdType InsertNextTuple(vtkIdType j, vtkFieldData \*source)

|

| Insert the jth tuple in source field data at the end of the tuple

| matrix. Range checking is performed and memory is allocated as

| necessary.

|

| InsertTuple(...)

| InsertTuple(self, i:int, j:int, source:vtkFieldData) -> None

| C++: void InsertTuple(vtkIdType i, vtkIdType j,

| vtkFieldData \*source)

|

| Insert the jth tuple in source field data at the ith location.

| Range checking is performed and memory allocates as necessary.

|

| NullData(...)

| NullData(self, id:int) -> None

| C++: void NullData(vtkIdType id)

|

| Sets every vtkDataArray at index id to a null tuple.

|

| Reset(...)

| Reset(self) -> None

| C++: void Reset()

|

| Resets each data array in the field (Reset() does not release

| memory but it makes the arrays look like they are empty.)

|

| SetGhostsToSkip(...)

| SetGhostsToSkip(self, \_\_a:int) -> None

| C++: virtual void SetGhostsToSkip(unsigned char)

|

| SetNumberOfTuples(...)

| SetNumberOfTuples(self, number:int) -> None

| C++: void SetNumberOfTuples(vtkIdType number)

|

| Set the number of tuples for each data array in the field. This

| method should not be used if the instance is from a subclass of

| vtkFieldData (vtkPointData or vtkCellData). This is because in

| those cases, the attribute data is stored with the other fields

| and will cause the method to behave in an unexpected way.

|

| SetTuple(...)

| SetTuple(self, i:int, j:int, source:vtkFieldData) -> None

| C++: void SetTuple(vtkIdType i, vtkIdType j, vtkFieldData \*source)

|

| Set the jth tuple in source field data at the ith location. Set

| operations mean that no range checking is performed, so they're

| faster.

|

| Squeeze(...)

| Squeeze(self) -> None

| C++: void Squeeze()

|

| Squeezes each data array in the field (Squeeze() reclaims unused

| memory.)

|

| ----------------------------------------------------------------------

| Data descriptors inherited from vtkFieldData:

|

| actual\_memory\_size

| read-only, Calls GetActualMemorySize

|

| ghost\_array

| read-only, Calls GetGhostArray

|

| ghosts\_to\_skip

| read-write, Calls GetGhostsToSkip/SetGhostsToSkip

|

| m\_time

| read-only, Calls GetMTime

|

| number\_of\_components

| read-only, Calls GetNumberOfComponents

|

| number\_of\_tuples

| read-write, Calls GetNumberOfTuples/SetNumberOfTuples

|

| ----------------------------------------------------------------------

| Methods inherited from vtkmodules.vtkCommonCore.vtkObject:

|

| AddObserver(...)

| AddObserver(self, event:int, command:Callback, priority:float=0.0) -> int

| C++: unsigned long AddObserver(const char\* event,

| vtkCommand\* command, float priority=0.0f)

|

| Add an event callback command(o:vtkObject, event:int) for an event type.

| Returns a handle that can be used with RemoveEvent(event:int).

|

| BreakOnError(...)

| BreakOnError() -> None

| C++: static void BreakOnError()

|

| This method is called when vtkErrorMacro executes. It allows the

| debugger to break on error.

|

| DebugOff(...)

| DebugOff(self) -> None

| C++: virtual void DebugOff()

|

| Turn debugging output off.

|

| DebugOn(...)

| DebugOn(self) -> None

| C++: virtual void DebugOn()

|

| Turn debugging output on.

|

| GetCommand(...)

| GetCommand(self, tag:int) -> vtkCommand

| C++: vtkCommand \*GetCommand(unsigned long tag)

|

| GetDebug(...)

| GetDebug(self) -> bool

| C++: bool GetDebug()

|

| Get the value of the debug flag.

|

| GetGlobalWarningDisplay(...)

| GetGlobalWarningDisplay() -> int

| C++: static vtkTypeBool GetGlobalWarningDisplay()

|

| GetObjectDescription(...)

| GetObjectDescription(self) -> str

| C++: std::string GetObjectDescription() override;

|

| The object description printed in messages and PrintSelf output.

| To be used only for reporting purposes.

|

| GetObjectName(...)

| GetObjectName(self) -> str

| C++: virtual std::string GetObjectName()

|

| GlobalWarningDisplayOff(...)

| GlobalWarningDisplayOff() -> None

| C++: static void GlobalWarningDisplayOff()

|

| GlobalWarningDisplayOn(...)

| GlobalWarningDisplayOn() -> None

| C++: static void GlobalWarningDisplayOn()

|

| HasObserver(...)

| HasObserver(self, event:int, \_\_b:vtkCommand) -> int

| C++: vtkTypeBool HasObserver(unsigned long event, vtkCommand \*)

| HasObserver(self, event:str, \_\_b:vtkCommand) -> int

| C++: vtkTypeBool HasObserver(const char \*event, vtkCommand \*)

| HasObserver(self, event:int) -> int

| C++: vtkTypeBool HasObserver(unsigned long event)

| HasObserver(self, event:str) -> int

| C++: vtkTypeBool HasObserver(const char \*event)

|

| InvokeEvent(...)

| InvokeEvent(self, event:int, callData:Any) -> int

| C++: int InvokeEvent(unsigned long event, void\* callData)

| InvokeEvent(self, event:str, callData:Any) -> int

| C++: int InvokeEvent(const char\* event, void\* callData)

| InvokeEvent(self, event:int) -> int

| C++: int InvokeEvent(unsigned long event)

| InvokeEvent(self, event:str) -> int

| C++: int InvokeEvent(const char\* event)

|

| This method invokes an event and returns whether the event was

| aborted or not. If the event was aborted, the return value is 1,

| otherwise it is 0.

|

| Modified(...)

| Modified(self) -> None

| C++: virtual void Modified()

|

| Update the modification time for this object. Many filters rely

| on the modification time to determine if they need to recompute

| their data. The modification time is a unique monotonically

| increasing unsigned long integer.

|

| RemoveAllObservers(...)

| RemoveAllObservers(self) -> None

| C++: void RemoveAllObservers()

|

| RemoveObserver(...)

| RemoveObserver(self, \_\_a:vtkCommand) -> None

| C++: void RemoveObserver(vtkCommand \*)

| RemoveObserver(self, tag:int) -> None

| C++: void RemoveObserver(unsigned long tag)

|

| RemoveObservers(...)

| RemoveObservers(self, event:int, \_\_b:vtkCommand) -> None

| C++: void RemoveObservers(unsigned long event, vtkCommand \*)

| RemoveObservers(self, event:str, \_\_b:vtkCommand) -> None

| C++: void RemoveObservers(const char \*event, vtkCommand \*)

| RemoveObservers(self, event:int) -> None

| C++: void RemoveObservers(unsigned long event)

| RemoveObservers(self, event:str) -> None

| C++: void RemoveObservers(const char \*event)

|

| SetDebug(...)

| SetDebug(self, debugFlag:bool) -> None

| C++: void SetDebug(bool debugFlag)

|

| Set the value of the debug flag. A true value turns debugging on.

|

| SetGlobalWarningDisplay(...)

| SetGlobalWarningDisplay(val:int) -> None

| C++: static void SetGlobalWarningDisplay(vtkTypeBool val)

|

| This is a global flag that controls whether any debug, warning or

| error messages are displayed.

|

| SetObjectName(...)

| SetObjectName(self, objectName:str) -> None

| C++: virtual void SetObjectName(const std::string &objectName)

|

| Set/get the name of this object for reporting purposes. The name

| appears in warning and debug messages and in the Print output.

| Setting the object name does not change the MTime and does not

| invoke a ModifiedEvent. Derived classes implementing copying

| methods are expected not to copy the ObjectName.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from vtkmodules.vtkCommonCore.vtkObject:

|

| debug

| read-write, Calls GetDebug/SetDebug

|

| global\_warning\_display

| read-write, Calls GetGlobalWarningDisplay/SetGlobalWarningDisplay

|

| object\_description

| read-only, Calls GetObjectDescription

|

| object\_name

| read-write, Calls GetObjectName/SetObjectName

|

| ----------------------------------------------------------------------

| Methods inherited from vtkmodules.vtkCommonCore.vtkObjectBase:

|

| FastDelete(...)

| FastDelete(self) -> None

| C++: virtual void FastDelete()

|

| Delete a reference to this object. This version will not invoke

| garbage collection and can potentially leak the object if it is

| part of a reference loop. Use this method only when it is known

| that the object has another reference and would not be collected

| if a full garbage collection check were done.

|

| GetAddressAsString(...)

| GetAddressAsString(self, classname:str) -> str

|

| Get address of C++ object in format 'Addr=%p' after casting to

| the specified type. This method is obsolete, you can get the

| same information from o.\_\_this\_\_.

|

| GetClassName(...)

| GetClassName(self) -> str

| C++: const char \*GetClassName()

|

| Return the class name as a string.

|

| GetIsInMemkind(...)

| GetIsInMemkind(self) -> bool

| C++: bool GetIsInMemkind()

|

| A local state flag that remembers whether this object lives in

| the normal or extended memory space.

|

| GetReferenceCount(...)

| GetReferenceCount(self) -> int

| C++: int GetReferenceCount()

|

| Return the current reference count of this object.

|

| GetUsingMemkind(...)

| GetUsingMemkind() -> bool

| C++: static bool GetUsingMemkind()

|

| A global state flag that controls whether vtkObjects are

| constructed in the usual way (the default) or within the extended

| memory space.

|

| InitializeObjectBase(...)

| InitializeObjectBase(self) -> None

| C++: void InitializeObjectBase()

|

| Register(...)

| Register(self, o:vtkObjectBase)

| C++: virtual void Register(vtkObjectBase \*o)

|

| Increase the reference count by 1.

|

| SetMemkindDirectory(...)

| SetMemkindDirectory(directoryname:str) -> None

| C++: static void SetMemkindDirectory(const char \*directoryname)

|

| The name of a directory, ideally mounted -o dax, to memory map an

| extended memory space within. This must be called before any

| objects are constructed in the extended space. It can not be

| changed once setup.

|

| SetReferenceCount(...)

| SetReferenceCount(self, \_\_a:int) -> None

| C++: void SetReferenceCount(int)

|

| Sets the reference count. (This is very dangerous, use with

| care.)

|

| UnRegister(...)

| UnRegister(self, o:vtkObjectBase)

| C++: virtual void UnRegister(vtkObjectBase\* o)

|

| Decrease the reference count (release by another object). This

| has the same effect as invoking Delete() (i.e., it reduces the

| reference count by 1).

|

| UsesGarbageCollector(...)

| UsesGarbageCollector(self) -> bool

| C++: virtual bool UsesGarbageCollector()

|

| Indicate whether the class uses `vtkGarbageCollector` or not.

|

| Most classes will not need to do this, but if the class

| participates in a strongly-connected reference count cycle,

| participation can resolve these cycles.

|

| If overriding this method to return true, the `ReportReferences`

| method should be overridden to report references that may be in

| cycles.

|

| ----------------------------------------------------------------------

| Class methods inherited from vtkmodules.vtkCommonCore.vtkObjectBase:

|

| override(...) from builtins.type

| This method can be used to override a VTK class with a Python subclass.

| The class type passed to override will afterwards be instantiated

| instead of the type override is called on.

| For example,

|

| class foo(vtk.vtkPoints):

| pass

| vtk.vtkPoints.override(foo)

|

| will lead to foo being instantied every time vtkPoints() is called.

| The main objective of this functionality is to enable developers to

| extend VTK classes with more pythonic subclasses that contain

| convenience functionality.

|

| ----------------------------------------------------------------------

| Data descriptors inherited from vtkmodules.vtkCommonCore.vtkObjectBase:

|

| class\_name

| read-only, Calls GetClassName

|

| is\_in\_memkind

| read-only, Calls GetIsInMemkind

|

| memkind\_directory

| write-only, Calls SetMemkindDirectory

|

| reference\_count

| read-write, Calls GetReferenceCount/SetReferenceCount

|

| using\_memkind

| read-only, Calls GetUsingMemkind