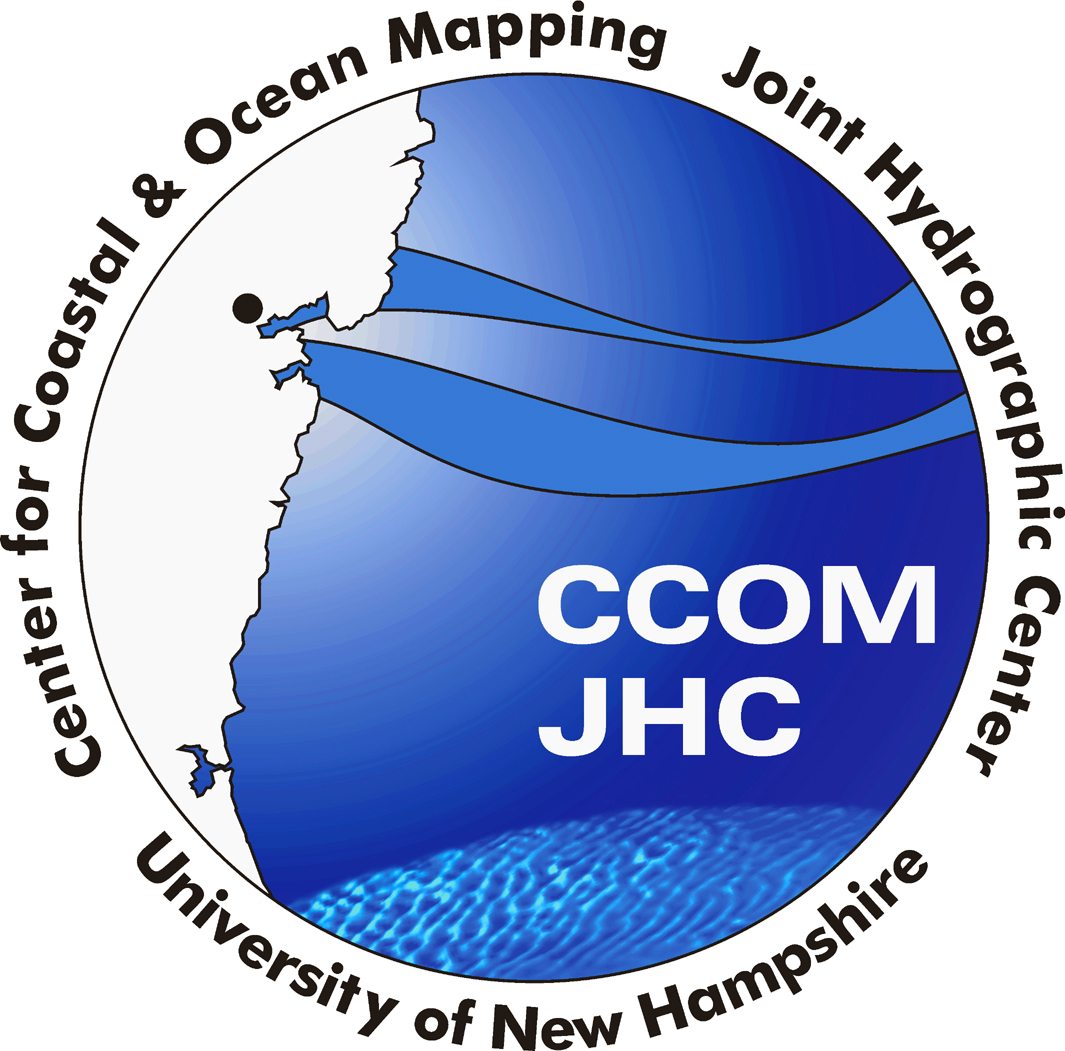
A Variable Resolution Grid Extension for BAG Files

Brian Calder

Center for Coastal and Ocean Mapping and NOAA/UNH Joint Hydrographic Center

Wade Ladner

Naval Oceanographic Office, Hydrographic Department

# A Variable Resolution Grid Extension for BAG Files

Version: 1.0

Date: 2014-07-15

Revision History:

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Description** | **Date** | **Author** |
| 1.0 | Initial revision for comments | 2014-07-15 | brc |
|  |  |  |  |
|  |  |  |  |

# Abstract

This document proposes an extension structure to be built into Bathymetric Attributed Grids (BAG) developed by the Open Navigation Surface Working Group (ONSWG), starting with version 2.0 of the support library. The intent is to provide a mechanism allowing for the storage of variable-resolution grids within a BAG file based on the model of a regular low resolution surface grid model for a given survey area that allows for piece-wise refinements of the cells to a higher resolution where necessary. That is, the current BAG structures for representation of bathymetry and uncertainty remain valid, but are taken to represent a best estimate for the depth at a relatively low resolution, with each cell at the lower resolution being (potentially) refined with a higher resolution regular grid that is also stored in the BAG file. Each cell can have a different resolution of refinement, allowing for piece-wise variable resolution reconstruction of a surface.

This document outlines the extensions required to the BAG file to support this mechanism, and the changes that would be required to the current (V 1.5.2) BAG API to support them. It also suggests a mechanism for internal tiling of the BAG file refinements that is required for the variable resolution extension, but might be of more general use in BAG files, and in particular for combining multiple fixed resolution BAG files into a single variable-resolution BAG file.

# Rationale

Bathymetric survey areas are rarely flat. At the same time, most techniques for measuring the depth of water have a density of measurement that is inversely proportional to (some function of) depth. Consequently, the rate at which stable estimates of depth can be constructed is also inversely proportional to (some function of) depth. When estimating depths, therefore, any regular grid – by far the most common method in current practice – is necessarily compromised when a resolution must be chosen. Most practitioners will choose a compromise grid resolution that is too coarse for the shallowest data, but too fine for the deepest data, which can lead to missing information in the shallower areas, and gaps in coverage (or interpolation) in deeper areas. More sophisticated methods might partition the area into regions of more or less homogeneous depth and establish regular grids of different resolutions in each region, but this quickly becomes unwieldy in practice.

New methods for bathymetric data processing have attempted to resolve this issue by establishing grids that adapt to the achievable resolution of the data as a function of position. These methods are typically driven by a measure of data density, or adequacy of representation in a model of the seafloor, and may use one or more passes over the data to determine the appropriate resolution at which to work before computing final estimates of depth. The result is a grid, often piece-wise regular, where the resolution is a complex function of spatial coordinates and data.

Unfortunately, however, tools to manipulate and archive these sorts of grids are only sparsely available. In particular, there is no standard form in which to preserve this type of grid for archive, or for exchange between different software packages providing the various stages of the processing pipeline. The BAG file format, however, has become a *de facto* standard for archival of high resolution bathymetric grids with associated uncertainty and metadata, and, through its IHO S-102 version, a *de jure* standard for the same. It is therefore logical to attempt to extend the base BAG file format to allow for the representation of a (subset of) variable resolution grid models. This document provides a template for these extensions.

# Variable Resolution Grid Data Model

This document focuses on a particular variant of variable resolution grids, and specifically on piece-wise regular grids. The fundamental model, Figure 1, is of a low resolution “base” grid which represents a best estimate of the depth within a significant area (along with an associated uncertainty and other metrics), each cell of which may be optionally refined with a higher-resolution regular grid containing estimates of depth and associated uncertainty (and possibly other metrics). This structure ensures that all of the components of the grid are regular grids, but still allows the resolution of the grid to change (at the rate of the low resolution grid) to adapt to changes in depth within the area of interest. Note that this proposal is neutral as to how the estimates in the grid are constructed.

The refined grids, by design, do not overlap. Specifically, the outer bounds of the refined grid (i.e., the location of the outer-most depth estimate on all corners) are constrained to be strictly interior to the cell that the grid is refining. In particular, this means that no nodes exist on the edges of the cells of the low resolution grid, and therefore that there cannot be multiple estimates at the same location in space. The refined grid is also centered on the geospatial location of the cell that it is refining. Depending on the resolution of the refinements, this can mean that there are an even number of nodes in the grid, and therefore no depth estimate that corresponds to the location of the low resolution depth estimate that is being replaced.

The abstract model is completed by the same features that are present in current BAG files: a metadata object containing ISO-standard metadata, a set of “hydrographer’s over-rides” to track changes made to the data by a hydrographer in order to preserve hydrographically significant features, and a digital signature to confirm authenticity and correctness of the data.

The data model is readily translated into a number of encodings. In this document, the focus is on HDF5 encoding since the target is a BAG file. The hierarchical structure of the HDF5 representation of a BAG file makes it easy to encapsulate the information for the extra metrics and refinements required to represent the variable resolution objects in such a way that they do not significantly change the rest of the BAG file.

|  |
| --- |
|  |
| **Figure 1:** General structure of the variable resolution data surface. The low resolution surface is georeferenced on the southwest corner, and is point referenced. Refined grids are centred in their cell, and do not touch the boundary of the cell. Note here that *NH*(*i*, *j*) = 8, since refined depths are computed at the intersections of the grid lines, and *H*(*i*,*j*) > 0 is required to avoid duplication of points on the edges of the low resolution cells. |

# Extension Layers

## Design

The data model shares many features of the current BAG structure. For example, the low resolution model, metadata, and signature are already present, and only the refinements need to be added as a new component, along with their associated tracking list. (The current tracking list cannot be used since it assumes a strictly regular grid and therefore could not be used to indicate a change in a refined grid.) In order to minimise the changes to the BAG data file, the depth and uncertainty components of the low resolution grid will be mapped into the existing depth and uncertainty layers, and the metadata set to indicate the appropriate geospatial location. As for current BAGs, the georeferencing point is the southwest corner, and refers to the location where the depth estimate is valid (i.e., not the corner of the surrounding grid cell).

The uncertainty rendering in the metadata may be any of the valid enumerates. A new metadata entry is required to indicate that refined depths are available, using a Boolean element set to 1 when refinements are available, and 0 otherwise; lack of the appropriate element will indicate that no refinements are available (for backwards compatibility).

There is some question of whether the refinements should be added as a single data structure, or as multiple datasets with a specific name, Figure 2. The former is simpler to generate and would lead to fewer computations during read in the sense that the code would not need to compute names for refinement datasets and then open then with the HDF5 API. The latter is simpler to understand logically, and may be simpler to update if required. On balance, however, it seems likely that saving the refinements in a single data structure, with index values for where each refinement starts within the structure, is likely to give better performance on read. The tiling structure introduced later provides for a segmented compromise between these two alternatives.

An implication of this design is that the data structure needs to ensure that there are sufficient indices available to serve the needs of all refinements. This could either be done by specifying 64-bit indices, or by breaking the total area into smaller tiles and then indexing within a tile. The former is simpler in spirit, but potentially more inefficient in storage; the latter is more complex in design but far more extensible and flexible. The latter solution is recommended here.

## Implementation

### Overall Depth and Uncertainty

The depth and uncertainty information for the low resolution grid will be stored in the HDF file at /BAG\_root/elevation and /BAG\_root/uncertainty, respectively. Georeferencing and row-order shall be maintained from the current BAG format. The metadata element gmd:resolution shall refer to the resolution of the low resolution grid; the metadata interpretation of depth (bag:BAG\_DepthCorrect Code) and uncertainty (bag:BAG\_VertUncertCode) may be any valid enumerate currently defined in the BAG format. This interpretation shall be understood to be consistent for all of the data, including the refinements.

### Metadata Extensions & Interpretation

The Boolean element to indicate whether refined depths are available or not shall be bag:BAG\_RefinementsAvailable, as part of the gmd:spatialRepresentation Info element. The element shall contain a single element of type gco:Boolean with a value of 0 to indicate no refinements are available and 1 to indicate that refinements are available. Lack of a bag:BAG\_RefinementsAvailable element shall be taken as indication that no refinements are available.

|  |
| --- |
|  |
| **Figure 2:** Alternative methods of storing the refined grids: as individual datasets stored as two dimensional grids (left), or as a single one dimensional grid (right) for all of the refined grids. |

### Tiling Data

In order to segment the information in the grid to a reasonable size, the data is split into logical tiles containing a number of low resolution cells. By design, tiles are square and of fixed size, with the exception of the last row and column of tiles, which have to be variably sized to accommodate arbitrarily sized input data grids. Not all tiles within the bounding box of the low resolution grid will necessarily be active (i.e., have at least one refined low resolution cell). In order to indicate which tiles are active, a two-dimensional binary grid is stored with the refinements, using value 0 to indicate that the tile is inactive, and 1 to indicate that it is active. The grid shall be ordered row-major from the south (i.e., west to east within a row, and south to north within the grid). Thus, tile (row = 0, column = 0) is the most southwest of all tiles, and contains the low resolution cell refinements in the southwest corner of the area of interest; tile(0,1) is immediately to its east; while tile (1,0) is immediately to its north.

The grid shall also have an attribute, TileSize, stored as a 16-bit unsigned integer, to indicate the size, in low resolution cells, of the tile. The excess between the smallest integer multiple of this size just less than the total width or height of the low resolution grid shall be understood to be the width and height (respectively) of the last column and row (respectively) tile. Where possible, generators may attempt to generate tiles with a size that divides the total size of the output, but readers must be tolerant of any tile size at the last row/column of the grid.

### Refinement Layer Structure

The refinements shall be stored in a new layer at /BAG\_Root/Refinements, Figure 3. This layer shall contain a TileMask dataset at /BAG\_Root/Refinements/Tile Mask, consisting of a binary bitmask to indicate whether each tile in the active area contains active refinements, as defined above. The TileMask coverage is the same as the coverage of the low resolution surface. Each tile that is active (i.e., has at least one active refinement grid) shall be represented within a sub-group of the main refinements group, as /BAG\_Root/Refinements/Tile\_R\_C where R and C are the row and column address of the tile within the active area bitmask defined above. This sub-group shall have as attribute TotalRefinements, set to be the total number of refinement grid nodes in all cells of the tile.

The refinements shall be stored in a sub-group of the tile, using a one-dimensional array at /BAG\_Root/Refinements/Tile\_R\_C/RefinedElevation for the elevation, and another at /BAG\_Root/Refinements/Tile\_R\_C/Refined-Uncertainty for the uncertainty. Both arrays shall be represented as 32-bit floating point values, and shall use the BAG-standard values for “no data” (i.e., as defined in bag.h through the NULL\_ELEVATION and NULL\_UNCERTAINTY macros). The coordinate reference frame for the depth shall be consistent with the BAG file format (i.e., positive values indicate height above datum, and negative values depth).

Indices for where the refinement grid for each cell of the tile are stored within these arrays shall be stored at /BAG\_Root/Refinements/Tile\_R\_C/Indices in a two-dimensional grid. The grid shall be ordered row-major from the south (i.e., west to east within a row, and south to north within the grid). The indices shall be 32-bit numbers, with 0xFFFFFFFF used to indicate “no refinements specified”; indices shall start at zero.

Although redundant, the number of nodes in each dimension of the refinement grid shall be stored at /BAG\_Root/Refinements/Tile\_R\_C/Dimensions as a two-dimensional regular grid. The grid shall be ordered row-major from the south as before, be represented as a 16-bit integer, and use value 0 to indicate “no refinements specified”. All refinement grids, by design, are square, and therefore a reader should expect to read N2 values if the corresponding dimensions array entry is set to N. Note that a value of 1 is impossible, and should be considered a format error.

|  |
| --- |
| Macintosh HD:Users:brc:Projects:openns:docs:VariableResolution:VR_BAG_Fig3.png |
| **Figure 3:** Proposed structure for the HDF5 implementation of the variable resolution grid extensions for BAG files. |

To complete the information required to reconstruct the refinement, the sample spacing for each refinement grid shall be stored as a two-dimensional regular grid, at /BAG\_Root/Refinements/Tile\_R\_C/SampleSpacing. The grid shall be ordered row-major from the south as before, be represented as a 32-bit floating point number, with value 0.0 indicating “no refinements specified”. As before, sampling in the cardinal dimensions is identical so that refinement grids are square. (Note that it might appear to Dimensions and SampleSpacing are mutually redundant. It is possible, however, that a particular algorithm for laying out nodes in the high resolution refinements might not be entirely predictable – for example, it might not make the refinement as big as possible so as to just fit inside the low resolution cell – preserving both ensures that the results of the algorithm can be retained in the extended BAG file.)

Hydrographer over-rides shall be stored as an HDF5 composite object, Figure 4, containing both cell and refinement grid row and column locations for where the correction was applied, along with the original depth and uncertainty that were replaced by the correction. All location specifications shall be stored as 16-bit unsigned values, while the depth and uncertainty shall be stored as 32-bit floating point values. The composite object shall also contain a track code, stored as an 8-bit unsigned enumerated type using one of the enumerates defined in the current BAG specification (i.e., the bagTrackCode enumerate in bag.h), and an identification number, stored as a 16-bit unsigned integer, to tie it into an element of the metadata structure. Note that this is very similar to the current BAG tracking list (and fits into the same storage space), but supplies the extra information required to find the location specified within the refined grid. The over-ride list shall be stored within the tile sub-group at /BAG\_Root/Refinements/Tile\_R\_C/TrackingList as a one-dimensional array of the composite data type. Over-rides to the low resolution surface may be specified through the BAG’s current tracking list, at /BAG\_Root/tracking\_list.

Optional layers, as defined for the /BAG\_Root group, are allowed in a tile, and shall have the same format as if they appeared in the /BAG\_Root group.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | CellRow | U16 | | CellCol | U16 | | GridRow | U16 | | GridCol | U16 | | Depth | F32 | | Uncertainty | F32 | | ReasonCode | U8 | | GroupID | U16 | |
| **Figure 4:** Structure of the compound type used to store the refined grid tracking list data. An array of these is stored with the tile refinement. |

# API Additions

The current BAG API will be maintained for backwards compatibility with files containing only a single resolution grid. In particular, the same routines for opening and creating BAG files will be used, along with the functionality for reading, manipulating, and writing metadata in XML encoding, and for reading and writing the current set of optional layers.

The current API for access to the data will be maintained, but will be interpreted as reading the low resolution grid’s elevation and uncertainty values.

A set of new API calls for access to the refined data shall be added, reflecting the same structure as the current set. The signatures for the calls are given in Table 1. Since it is difficult to specify a read of a row or sub-region when the regions are at different resolutions (except as a geographic location), the API is constrained to reads with a quantum of a low resolution cell, and returns the full refined grid at each cell as a C-style structure, Figure 5, or an array of these if appropriate. If an array is returned, it shall be returned as the one-dimensional equivalent of the two-dimensional area of interest, ordered row-major from the south as before.

|  |
| --- |
| struct {  f32 depth; // Depth at low resolution cell (m)  f32 uncertainty; // Uncertainty at low resolution cell  u16 refined\_node\_dim; // Width/height of refined grid (nodes)  f32 refined\_spacing; // Node spacing in refined grid (m)  f32 \*refined\_elevation; // Pointer to refined elevations (m)  f32 \*refined\_uncertainty; // Pointer to refined uncertainties  } RefinedCell; |
| **Figure 5:** Form of the RefinedCell structure used to pass information about the refined grid to and from the library. |

# Possible Extension for Composite Files

Although not a part of the current proposal, the introduction of tiles to the BAG file structure opens up possibilities for constructing composite files, given some small modifications. The HDF5 data structure allows for ‘mounting’ of files within a HDF5 file by a process analogous to mounting shares or drives in a Unix-style operating system. Using this mechanism, a separate HDF5 file can be logically linked to a particular sub-group object within an open HDF5 file so that references to the sub-group used as a mount point are automatically translated into references to the root group of the mounted HDF5 file.

In the context of this proposal, it would therefore be possible to have a single HDF5 file for each tile, and mount them on the /BAG\_Root/Refinements/ Tile\_R\_C sub-group as required for access to the data; a specially structured ‘root’ HDF5 file containing a sub-group that (a) indicates that mounts are required, and (b) provides the locations for the component tile grids, would also be required. The /BAG\_Root/elevation and /BAG\_Root/uncertainty sub-groups of the mounted BAG file would be interpreted as the elevation and associated uncertainty sub-groups for the tile, while the /BAG\_Root/Refinements sub-group would provide refinements for the tile as before. It is possible that the /BAG\_Root/Refinements/ TileMask object could be eliminated in this mode, but it could be retained as a 1×1 grid for compatibility (which would be definition be set to 1).

**Table 1:** List of API calls used to access the refined grid data as a series of RefinedCell objects.

|  |
| --- |
| bagError bagReadRefinedNodeLL(bagHandle bagHandle, f64 x, f64 y,  RefinedCell \*data); |
| bagError bagReadRefinedNode(bagHandle bagHandle,  u32 cellRow, u32 cellCol, RefinedCell \*data); |
| bagError bagReadRefinedNodePos(bagHandle bagHandle,  u32 cellRow, u32 cellCol, RefinedCell \*data,  f64 \*x, f64 \*y); |
| bagError bagWriteRefinedNodeLL(bagHandle bagHandle, f64 x, f64 y,  RefinedCell \*data); |
| bagError bagWriteRefinedNode(bagHandle bagHandle,  u32 cellRow, u32 cellCol, RefinedCell \*data); |
| bagError bagReadRefinedRow(bagHandle bagHandle, u32 cellRow,  u32 startCellCol, u32 endCellCol, RefinedCell \*data); |
| bagError bagReadRefinedRowPos(bagHandle, u32 cellRow,  u32 startCellCol, u32 endCellCol, RefinedCell \*data,  f64 \*\*x, f64 \*\*y); |
| bagError bagWriteRefinedRow(bagHandle bagHandle, u32 cellRow,  u32 startCellCol, u32 endCellCol, RefinedCell \*data); |
| bagError bagReadRefinedRegion(bagHandle bagHandle,  u32 startCellRow, u32 endCellRow,  u32 startCellCol, u32 endCellCol,  Refinedcell \*data); |
| bagError bagReadRefinedRegionPos(bagHandle bagHandle,  u32 startCellRow, u32 endCellRow,  u32 startCellCol, u32 endCellCol,  Refinedcell \*data, f64 \*\*x, f64 \*\*y); |
| bagError bagWriteRefinedRegion(bagHandle bagHandle,  u32 startCellRow, u32 endCellRow,  u32 startCellCol, u32 endCellCol,  Refinedcell \*data); |
| bagError bagReadRefinedDataset(bagHandle bagHandle); |
| bagError bagReadRefinedDatasetPos(bagHandle bagHandle,  f64 \*\*x, f64 \*\*y); |
| bagError bagWriteRefinedDataset(bagHandle bagHandle); |
| bagError bagReadCorrectedRefinedDataset(bagHandle bagHandle,  u32 corrIndex, RefinedCell \*data); |
| bagError bagReadCorrectedRefinedRegion(bagHandle bagHandle,  u32 startCellRow, u32 endCellRow, u32 startCellCol,  u32 endCellCol, u32 corrIndex, RefinedCell \*data); |
| bagError bagReadCorrectedRefinedRow(bagHandle bagHandle, u32 cellRow,  u32 corrIndex, RefinedCell \*data); |
| bagError bagReadCorrectedRefinedNode(bagHandle bagHandle,  u32 cellRow, u32 cellCol, u32 corrIndex,  RefinedCell \*data); |

A major advantage of this scheme is efficiency: if the user is only interested in a subsection of the file, only the segment of the file actually required is opened, which may have advantages in memory usage and performance. An auxiliary advantage, however, is that this also allows for a mechanism to convert collections of current single resolution grids into a variable-resolution grid without having to reformat all of the data. That is, if a particular survey was broken up into multiple single resolution grids in order to manage maximum grid sizes or to achieve a compromise resolution matching data density, a ‘root’ BAG could be constructed which would associate them all together as separate ‘tiles’ of a variable resolution grid, which could then be distributed as a single entity (e.g., as a ZIP, or other, archive file). Some consideration would have to be given to overlapping areas between the component files (if any), but this should be possible to resolve through a well-understood (and well documented) set of rules, implemented automatically in the API. Since reformatting of data is typically error prone, difficult, and costly, being able to assemble a composite grid without changing the underlying components might be enough of an advantage that it would be worth the changes to the proposed structure and API. (Specifically, this would require that a variable resolution BAG had two copies of the low resolution depth and uncertainty arrays, one as the /BAG\_Root/elevation and /BAG\_Root/uncertainty pair, and another, broken up by tiles, as the /BAG\_Root/Refinements/Tile\_R\_C/elevation and /BAG\_Root/Refinements /Tile\_R\_C/uncertainty arrays.)

# Summary

The current proposal provides for a new optional layer for the BAG file format to specify high resolution refinement grids associated with a low resolution ‘overview’ elevation/uncertainty grid pair. The addition of the optional layer requires a small addition to the metadata that would be transparent to an older reader (and ignorable for single resolution BAG files), making the addition backwards compatible with the current BAG library. Variable resolution-aware versions of the library, however, would be able to recognise that refinements exist through examination of the metadata for the file, and users of the library could check to see whether this was an option by examination of the version information for the library. Single-resolution BAG files created with the variable resolution-aware version of the library would be readable by prior versions since BAG readers are required to ignore any layers that they do not understand, and the additions do not change the interpretation of any components to preclude a current single-resolution file being valid.

The current API would be maintained without changes, although all reads using the current API would refer only to the low resolution representation of the data, and no refinements could be extracted by these methods. A new API compatible with extraction of high resolution refinement grids has been proposed, mimicking the current API except that the refinement grids are passed as arrays of objects rather than as raw 32-bit floating point values.

A tiling scheme has been proposed to actively manage the size of the data structures associated with the refined grids. Although proposed in order to manage the required index variable size for most grids, this addition also allows, potentially, a mechanism for splitting grids into a group of files to manage maximum file size, or to allow a composite grid to be constructed from a set of extant, related single resolution BAG files.