BRL-CAD Database Format

Version 5 (DRAFT)

Lee A Butler
Michael John Muuss
Paul J Tanenbaum
John R Anderson
Robert G Parker
Ronald A Bowers
Christopher T Johnson
Eric W Edwards

1. Background and Terminology

BRL-CAD is a constructive solid geometry (CSG) modeling system. Primitive solid shapes are combined using boolean operations to form regions of homogeneous material.

The database is organized as a *directed acyclic graph* (DAG), which comprises

- primitive *solids* - the minimal elements of the DAG.
- *combinations* - the non-minimal elements of the DAG, some of which are specially marked as regions. The maximal elements of the DAG are called tops.
- *arcs* - contain boolean operators and 4x4 homogeneous transformation matrices.

In a slight abuse of terminology, the DAG is often spoken of as a tree or collection of trees. In this context, the solids are also called *leaves*.

1.1. Format of Data Elements/Database External Format

The external format has several important properties, especially with regard to the *Object_Body*:

- Numbers are stored in binary for storage efficiency, for speed of reading and writing, and for preventing errors from creeping in due to repetitive conversion between binary and an ASCII string representation. This eliminates the need to use the old g2asc and asc2g to move databases between machines of different architectures.
- All data in the object wrapper are stored in a machine-independent format, as follows:
  - All floating point numbers are stored as IEEE double-precision, in big-endian order, where byte 0 is on the left end of the word.

- All integers are stored as either *unsigned* or *twos-complement signed* binary numbers in either 8, 16, 32, or 64 bits, in big-endian order.
- All character strings are stored in the ASCII 8-bit character set. A string is stored as an integer followed by an array of 8-bit characters. The last character in the array is always a null byte. The integer indicates the number of bytes in the array including the terminating null.

2. Definition of a Single, Generic Database Object

The database access library stores *objects* as a collection of data with a globally unique name and places no interpretation on the content of those data. The *object* is the smallest granularity of an item in the database; objects must be read from and written to the database in a single atomic operation.

In the case of librt, each database object will contain exactly one combination node or leaf (solid) node.

2.1. Object Structure

All objects share certain common properties, which are stored in a standardized *object wrapper* consisting of an Object Header and an Object Footer.

The Object Header consists of:

- An 8-bit Magic1 element that holds a specific magic number value used for database integrity checking.
- A 16-bit Flags element consisting of three 8-bit fields: HFlags, AFlags, and BFlags, described later.
- A 16-bit Object_Type element organized into two 8-bit-wide fields: the Major_Type and the Minor_Type.
- An Object_Length element that indicates the total number of bytes required for this object, including the magic numbers.
- An Object_Name element that is a string holding a name unique to that object and drawn from a name space that is global to the database. Like other strings, it consists of two fields, Length and Data. In the case of the Object_Name element, these are referred to as Name_Length and Name_Data, respectively. Note: The Object_Name element is mandatory for all allocated storage in the database. Database free-space management objects are the only objects for which the Object_Name element is optional.

The Object Footer consists of:

- Any padding bytes necessary to bring the total size of the object in bytes to an integral multiple of 8.

- An 8-bit Magic2 element that holds a specific magic number value used for database integrity checking.

Objects may store application-specific information in an Object Interior.

- An object may optionally have an Object_Attributes element consisting of a pair of fields: Attribute_Length and Attribute_Data. From the point of view of the database interface specification, the names and values of these attributes are opaque (but a standardized import and export encoding API is provided).

- An object may optionally have an Object_Body element consisting of a pair of fields, Body_Length and Body_Data. From the point of view of the database interface specification, the format of the data is opaque.

Note that an object can now have (1) either an attribute or a body, (2) both an attribute and a body, or (3) neither an attribute nor a body.

The on-disk version of each object consists of three distinct parts: Object Header, Object Interior, and Object Footer. This is called the external format of the object.

Table 1. On-Disk BRL-CAD Object Structure

Part	Element	Comments
Object Header: (not compressible)	Magic1	Required
	HFlags, AFlags, BFlags	
	Object_Type (Major_Type, Minor_Type)	
	Object_Length	Required
	Object Name: Name_Length, Name_Data	Conditional on flag bit N Required for Application Data
Object Interior: (individually compressible)	Object Attributes: Attribute_Length, Attribute_Data	Conditional on flag bit A (ZZZ compression)
	Object Body: Body_Length, Body_Data	Conditional on flag bit B (ZZZ compression)
Object Footer: (not compressible)	Padding	As required to maintain 8-byte object boundaries
	Magic2	Required

The routines rt_db_get_internal() and rt_db_put_internal() are used to move objects between their format in the database disk file and their internal format in memory. The routines are defined in `librt`.)

2.2. Flags

The Flags element consists of three 8-bit fields: HFlags, AFlags, and BFlags. The HFlags field is 1 byte containing flag bits that pertain to the noncompressible basic header and the database object as a whole. The AFlags and BFlags fields are each single bytes containing flag bits that pertain to the (potentially compressed) attributes and body, respectively, in the object interior.

Table 2. BRL-CAD Flags Structure

HFlags									AFlags									BFlags							
7	6	5	4	3	2	1	0		7	6	5	4	3	2	1	0		7	6	5	4	3	2	1	0
OWid		NP	NWid		r	DLI			AWid		AP	r	r	AZ				BWid		BP	r	r	BZ		

2.2.1. Wid Flags

The length of an object or sub-element in the database is recorded using an unsigned integer. These are variable-width fields based on the magnitude of the maximum number needed. The Wid bits specify the size of the unsigned integer employed in each instance. There are four 2-bit width (Wid) flags: Object_Wid (OWid) and Name_Wid (NWid) (stored in HFlags), Attribute_Wid (AWid) (stored in AFlags), and Body_Wid (BWid) (stored in BFlags). The Wid fields are interpreted in this manner:

Table 3. Wid Flag Definitions

Wid Bits	Width (in bits) of Associated Length Fields
00	8
01	16
10	32
11	64

The OWid flag, at the high end of HFlags, encodes the width of the Object_Length field. The NWid flag, in bits 3 and 4 of HFlags, encodes the width of the Name_Length field (when the name element is present; see the N bit, shown later). AWid (or BWid, as the case may be) encodes the width of the Attribute_Length field (when the Object_Attributes or Object_Body element is present; see the AP and BP bits below).

(See the original draft at *[http://ftp.arl.mil/~mike/papers/brlcad5.0/newdb.html]*.)

The rationale for allowing the width of the Object_Length field to be specified independently of the other widths is to save space on objects in which the values in many of the length fields nearly overflow the specified field width, so that their sum requires a wider field. For example, for four 255-byte interior fields, the corresponding length fields need be no more than 8 bits wide, so the choice Interior_Wid=00 suffices, but their combined length of 1020 bytes would require Object_Wid=01. Because all of the length fields besides Object_Length must have the same width (FIXME: is that true?), the largest of the values stored in these length fields determines

the value of Interior_Wid required. Both Object_Wid and Interior_Wid may vary from object to object. It is expected that the routines that write an object to the disk will use the narrowest width possible for each object.

2.2.2. "r" Bits

The bits labeled as "r" in all three flags are reserved for future design work assigning additional optional fields in the object.

2.2.3. HFlags - the DLI Flag

The DLI flag is a 2-bit flag that indicates whether the object is an Application Data Object or a Database Layer Internal Object. The bits are interpreted as follows:

Table 4. DLI Flag Structure

DLI Bits	Meaning
00	Application Data Object The object contains application-specific data. N must be 1. A and B are determined by what the application presents for storage in the object; both may be 0 (empty Object_Interior).
01	Database Layer Internal, Header Object A Header Object must be the first object encountered in the database. In order to support direct concatenation of two existing databases into one new database, additional header objects may appear elsewhere in the database The header object has no object name, object attributes, or object body (e.g., NP=0, AP=0, BP=0). Major_Type=RESERVED, Minor_Type=0.
10	Database Layer Internal, Free Storage. Unused space in the database is kept using a special Free DB Storage object that has no object name or object attributes. The object body is null-filled and of the proper size for the storage to be represented. Like all other objects, the total length of the object will be a multiple of 8 bytes. NP=0, AP=0, BP=1. Major_Type=RESERVED, Minor_Type=0.
11	Database Layer Internal, Reserved This value is reserved for future use.

The DLI flag is not available to the higher database access layers.

Note

Implementation note: Before writing a new object into the database in a free area, the library should read the object header from the database and confirm that the space is indeed free. Similarly, additions to the end should be checked by ensuring that the file hasn't been extended. In case the check fails, the database write should fail, the user

should be notified, and the internal library mode (not the operating system file access permissions) should be changed over to read-only access so that no further attempts to write will be issued. These checks will provide protection against two or more users trying to modify the same database simultaneously and accidentally stepping on each other. In the NFS world, file locking isn't a strong enough assurance.

2.2.4. HFlags - the NP Bit

The "NP" bit indicates whether the Name element (consisting of Name_Length and Object_Name fields) is present (1) or absent (0) in the noncompressible basic header immediately following the Object_Length field. The width of the Name_Length field is specified by the Name_Wid field.

2.2.5. AFlags/BFlags - the AP/BP Bit

The ``(A|B)P'' bit indicates whether the Attributes (or, alternatively, Body) element consisting of Attribute_Length and Attribute_Data (or Body_Length and Body_Data) fields, is present (1) or absent (0) in the Object_Interior.

2.2.6. AFlags/BFlags - the AZ/BZ Flag

The 3-bit ``(A|B)Z'' flag indicates the compression, if any, of the object Attributes (or Body):

Table 5. AZ/BZ Flag Definitions

AZ/BZ Bits	Compression Algorithm
000	None
001	GNU GZIP
010	Burroughs-Wheeler
011	Reserved
100	Reserved
101	Reserved
110	Reserved
111	Reserved

2.3. Object Type

The Object_Type element is always 16 bits wide, organized into two 8-bit-wide fields: the Major_Type and the Minor_Type.

Table 6. Object_Type Element Structure

Object_Type	
Major Type	Minor Type

Object_Type															
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Each different Major_Type value is assigned to a different class of database objects. The following values are defined in this specification:

Table 7. Major_Type Values and Meanings

Value	Object Class
0	Reserved
1	BRL-CAD Nongeometry Objects
2	BRL-CAD Geometry Objects
3	Attribute-Only Objects
8	Experimental Binary Objects (Unrecorded Structure) (Minor Type Unspecified)
9	Uniform Array Binary Objects, (Type Described in Minor Type)
10	MIME_Typed Binary Objects (Attribute "mime_type" Describes Format)
16-31	Registered-Type Binary Objects
128	First Non-ARL Type Begins Here

The remainder are available for extending the types of objects that may be stored in the database, allowing BRL-CAD users to extend the database for their own particular purposes far beyond what the "attribute" method permits.

2.3.1. Major_Type = 0: Reserved

Major Type 0 is illegal. The rationale is to provide the library an opportunity to detect incompletely filled in data structures.

2.3.2. Major_Type = 1: BRL-CAD Nongeometry Objects

This class of objects is private to librt and concerns all nongeometric objects needed by the library. For this Major_Type, the following Minor_Type values are defined:

Table 8. Major_Type = 1: Minor_Type Values and Meanings

Minor_Type Value	Object Type
0	Reserved for sanity check
1	Combination
2	Grip (Nongeometric)
3	Joint (Nongeometric)

All other values reserved for future expansion.

?????Should "Grip" and "Joint" objects be of this type, or Major_Type = 2?

2.3.3. Major_Type = 2: BRL-CAD Geometry Objects

This class of objects is private to librt and concerns all geometric objects needed by the library. Typically, there will be one xxx/xxx.c module in librt for each minor type. For this Major_Type, the following Minor_Type values are defined:

Table 9. Major_Type = 2: Minor_Type Values and Meanings

Minor_Type Value	Object Type
0	Reserved for sanity check
1	Torus (TOR)
2	Truncated General Cone (TGC)
3	Ellipsoid (ELL)
4	Generalized ARB. V + 7 vectors
5	ARS
6	Half-Space (HALF)
7	Right Elliptical Cylinder (REC) (TGC special case)
8	Polygonal faceted object (Polysolid)
9	B-Spline Solid
10	Sphere (ELL Special Case)
11	n-Manifold Geometry (NMG) solid
12	Extruded bitmap solid
13	Volume (VOL)
14	ARB with N faces (ARBN)
15	Pipe (wire) solid (PIPE)
16	Particle system solid (PART)
17	Right Parabolic Cylinder (RPC)
18	Right Hyperbolic Cylinder (RHC)
19	Elliptical Paraboloid (EPA)
20	Elliptical Hyperboloid (EHY)
21	Elliptical Torus (ETO)
22	Grip Nongeometric
23	Joint Nongeometric
24	Height Field (HF)

Minor_Type Value	Object Type
25	Displacement Map (DSP)
26	2D Sketch (SKETCH)
27	Solid of extrusion (EXTRUDE)
28	Instanced submodel
29	FASTGEN4 CLINE solid
30	Bag o' triangles (BOT)
31	Combination Record
32	Experimental binary
33	Uniform-array binary
34	MIME-typed binary
35	Superquadratic ellipsoid
36	Metaball
37	Brep object
38	Hyperboloid of one sheet (HYP)
39	Constraint object
40	Solid of revolution
41	Collection of points (PNTS)

The details of these Minor_Types are provided in Section IV.

2.3.4. Major_Type = 3: Attribute-Only Objects

This type of object stores only attributes in the object interior section; it has no object body elements.

For example, if several objects need to have the same shader parameters, it would be possible to create one attribute-only object to hold these common attributes and serve as a simple form of "macro". Objects that needed to share these attributes could all reference the same attribute object. If the attribute object is altered, then all of the objects that reference it would be updated together. Without this capability, the user would have to update each element individually to alter the attributes.

Conventions will have to be established regarding which attributes of an attribute-only object will be used when a macro reference is performed. For example, rt shaders will only be interested in the value of the "oshader=" attribute, while librt's tree-walker might also be interested in the "rgb=", "giftmater=", "nsn=", "material=", and "los=" attributes (assuming that a convention was developed so that a combination could macro-reference an attribute-only object too).

An attribute-only object may not have an object body; thus, flag bit B must always be zero for this type of object.

As used by the rt family of applications codes, these attribute-only objects will contain "macros" for shaders. The shader name and its parameters shall be encoded as a single ASCII string, which

is the value of the "oshader=" attribute. An rt shader named "macro" (or equivalent) would take a single parameter "obj=", which would specify the name of the attribute-only object in the database from which the actual shader and shader parameter information would be extracted.

There will be one attribute-only object with a reserved object name of "_GLOBAL" that will be used to contain various kinds of states that are global to the entire ".g" database and that had previously been found in the database header itself. There will be the following BRL-CAD-specific attributes whose meaning is predefined for the _GLOBAL object:

- title = The database "title" string previously found in the database header.
- units = The most recent editing units, specified as an ASCII string with a floating point conversion factor. For example, the conversion factor for inches to millimeters would be 25.4.
- regionid_colortable = A string that contains a collection of all the information previously found in "struct material_rec ID_MATERIAL" records. Exact encoding yet to be determined; it's a collection of integer 5-tuples of the form: {low, high, r, g, b}.

In addition, the "comment=" attribute of the "_GLOBAL" object may be used to store human-readable remarks about the database that are not more properly associated with a specific database item. These might include remarks about data sources, model evolution, security classification, and release restrictions. In the absence of some outboard revision-control system, this might also be a place to record modification history, although such use is discouraged.

2.3.5. Bulk Binary Objects (Major_Types 8-31)

This class of objects contains various "bulk" binary data that might otherwise have been placed in auxiliary files.

MGED and stand-alone commands must be built to store/extract these opaque binary objects between a ".g" file and other files. A user might want to use those same MGED commands to store or extract the binary object body of any object for external processing. An easy example to imagine is the importing and exporting of texture maps for external processing, but the same commands could be used for importing and exporting solid parameters in their external binary form.

These objects may be referenced in combination nodes, for organizational purposes, but they cannot be drawn in MGED or raytraced, and doing so would result in a warning message being printed by the tree walker as that arc is traversed. This class may be used by all applications and layers.

The data's purpose may be placed in the "purpose=" attribute. (????????Need a table/registry of presently known values for this attribute.)

Routines that retrieve bulk binary objects should check the minor type and the "purpose=" attribute and send a warning message in the event of a mismatch, but best-effort processing of the object should continue. This will permit some degree of error checking, which should benefit novice users without standing in the way of "creatively" reusing one set of data, (e.g., using one array of values as both a height field and a bwtexture). This allows common data perversion practices, such as interpreting an array of floats as an array of bytes, to continue.

Each application will need to have its own syntax for the user to specify whether the data source is an outboard file or a raw-binary object. For example, the current RT sh_texture module uses the keyword file="name" to indicate an outboard file; that might be supplemented with an additional obj="name" possibility for retrieving from an inboard raw-binary object.

2.3.5.1. Major_Type = 8: Experimental Binary Objects

This class of objects contains bulk binary data and is intended for experimental use by applications developers. Each time a database containing objects of this type is opened, BRL-CAD will issue a user-visible warning. Production software and databases should not use these objects. Developers should obtain registered 16-bit object types from the website in order to avoid collisions with other applications.

2.3.5.2. Major_Type = 9: Uniform Array Binary Objects

This class of objects contain various "bulk" binary data that might otherwise have been placed in an auxiliary file.

Point of Discussion?????Has ramifications... we have to implement type advising, so that applications that use these data can compare the type provided in the minor type code with the type that they're expecting and advise the user (with a warning message) that there is a potential type mismatch.

Table 10. Uniform Array Binary Objects Minor_Type Structure

Minor_Type							
7	6	5	4	3	2	1	0
r	r	Wid		S	Atom		

The 3-bit ``Atom" flag indicates the fundamental data type of the atomic elements in the array according to the following scheme:

Table 11. Atom Flag Definitions

Atom Bits	Data Type
000	Reserved for sanity check
001	Reserved
010	float (IEEE, network order)
011	double (IEEE, network order)
100	8-bit int
101	16-bit int
110	32-bit int
111	64-bit int

The ``S" bit indicates whether an integer type is signed (1) or unsigned (0). Floats and doubles (i.e., atomic types with the highest atom bit equal to 0) are explicitly signed, so they will have the

``S" bit equal to 1. (The bit patterns corresponding to unsigned floats and doubles are reserved for possible other use.)

The 2-bit ``Wid" flag specifies the length (in atomic elements) of the array elements:

Table 12. Wid Flag Definitions

Wid Bits	Atoms per Array Element
00	1
01	2
10	3
11	4

The remaining Minor_Type bits ``r" are reserved for the design committee to use for other purposes, possibly including extensions of the ``Atom" and/or ``Wid" flags.

As examples, data in PIX(5) format, which might be used for a texture map, would have Minor_Type ``0010 0100", indicating a triple of unsigned char, and CMYK data might be stored with Minor_Type ``0011 1011", indicating a quadruple of doubles.

The data's purpose (e.g., height field, texture, bump, displacement, etc.) may be placed in the "purpose=" attribute. ????? Point of Discussion ???(Need a table/registry of presently known values for this attribute.)

2.3.5.3. Major_Type = 10: MIME-Typed Binary Objects

This class of objects contains data, the format of which is specified in the attribute "mime_type". The Minor_Type of these objects should always be zero.

2.3.5.4. Major_Type = 16-31: Registered-Type Binary Objects

This class of objects contains application-specific bulk binary data and is intended for use in production software and databases. Developers can obtain registered 16-bit object types from the website to identify these objects. The data's purpose, (e.g., height field, texture, bump, displacement, etc.) may be placed in the "purpose=" attribute. (Need a table/registry of presently known values for this attribute).

2.3.6. Major_Type = 255: Database Layer Internal Objects

A Minor_Type of 1 indicates that this is a contiguous block of free storage.

A Minor_Type of 2 indicates that this is a database header.

2.4. Object Length

The Object Length specifies the number of 8-byte chunks used to store an object. This includes all bytes from Magic1 through Magic2, inclusive.

2.5. Object Name

The Object_Name element is a string that holds a name unique to that object and drawn from a name space that is global to the database. The Object_Name element is mandatory for all allocated storage in the database. Database free-space management objects are the only objects for which the Object_Name element is optional.

The name is specified in 8-bit ASCII. There is no support for UNICODE. The name is null-terminated, and the null byte is included in Name_Length.

See the section on DLI flags. In the case of Free objects, the name is not retained. Undeleted objects have a different DLI flag code.

2.6. Object Attributes

An object may optionally have an Object_Attributes element which stores an association list (key=value) binding attributes to values:

```
aname1=value1, aname2=value2, ..., anameN=valueN
```

The keys are ASCII strings of unlimited length. These attributes are intended for direct use by programs. There will be a WWW registry of attribute names presently in use to prevent two application developers from using the same attribute name for different purposes.

For attribute names and ASCII attribute values, The decision was taken to support 8-bit ASCII only. The on-disk encoding of this will simply be:

```
aname1 NULL value1 NULL ... anameN NULL valueN NULL NULL
```

where NULL represents a byte with all bits zero. The NULL in place of anameN+1 signals the end of the ASCII attribute data.

PROPOSED: A second type of attribute has an ASCII key but a binary value. Such attributes follow the ASCII-valued ones after the double NULLs:

```
anameN+1 NULL uintN <uintN  binary bytes> [...0 or more binary attribute pairs]
```

where, for each binary attribute pair, the uintN is of size AWid and is the length of the binary value for its ASCII key.

Every object in the database may have zero or more attributes attached to it; the meaning of these attributes will vary depending on which application or library processes them.

There are several aname conventions that all BRL-CAD applications are expected to respect. There will be a WWW extendable registry of "in-use" anames, so that independent applications developers may select aname strings for their own use without fear of name conflicts later. The initial registry would include:

- comment = Every object may optionally have a comment that contains a string of an arbitrary number of newline-terminated lines of text. These are strings for use by humans only. None

of the BRL-CAD software may parse or interpret these strings other than to print them and edit them when requested by the user. They are provided for the modeler to place notes in.

- nsn = The American National Stock Number (NSN) for this part, when known.
- material = The format of this string is not currently defined as there are conflicting naming/coding conventions employed by the various standards organizations (e.g., ISO, ASME, etc.).
- region = For combinations, indicates this combination is a region. Boolean.
- inherit = For combinations, indicates whether attributes from lower combinations in tree will replace higher ones. Boolean, default=0.
- oshader = For combinations, read by the "rt" program, optical shader name and parameter string (separated from each other by white space). Meaningful only at or above a region node, and only on a combination, or in an attribute-only "macro".
- rgb = For combinations, when present indicates optical rgb color is specified.
- region_id = For regions, GIFT compatibility. Integer.
- giftmater = For regions, GIFT compatibility. Integer. (Point of Discussion?????Should we use negative values for air codes, positive for non-air, so we can eliminate air codes?)
- aircode = For regions, air code. Integer. 0 is the same as attribute not specified. (Point of Discussion?????Possibly eliminated in favor of negative giftmater values).
- los = For regions, GIFT compatibility. Integer.
- component = For regions, the name of the MUVES component containing this object.
- rlist = The proposed BRL-CAD "replacement list" field would be stored on a binary-block attribute ("rlist="). [deferred implementation]
- macro = If present, specifies name of an attribute-only object to be consulted for additional attribute values.

All other attributes, from whatever source, would be stored similarly, including application-specific and end-user-created attributes.

2.7. Object Body

The contents of the Object Body are opaque?? to the database layer. The contents of this element are interpreted based upon the Object_Type. The Object_Body is not constrained to start on a chunk boundary.

2.8. Padding and Length Rounding

The minimal object is a Free object (with no name) 8 bytes long:

```
Magic1 (1 byte)
HFlags = 000xxxxx (1 byte)
AFlags = 0000xx00 (1 byte)
BFlags = 0000xx00 (1 byte)
ObjType = Free (2 bytes)
ObjLen = 8 (1 byte)
Magic2 (1 byte)
```

This is why we have chosen the 8-bit size for our chunks. Pad bytes are inserted as necessary in the Object Footer immediately before the second magic number so that the final byte of the object is the Magic2 byte. The pad bytes are not counted as part of the Body_Length, but are counted as part of the Object_Length.

The minimal valid object is thus the following Free object:

```
Magic1 (1 byte)
HFlags = 00000x10 (1 byte), OWid=00, NP=0, NWid=00, DLI=10
AFlags = 000xx000 (1 byte), AWid=00, AP=0, AZ=000
BFlags = 000xx000 (1 byte), BWid=00, BP=0, BZ=000
Object_Type = RESERVED (2 bytes)
Object_Length = 8 (1 byte)
Magic2 (1 byte)
```

The header of the database will always look like this:

```
Magic1 (1 byte)
HFlags = 000xxx01 (1 byte), HWid=00, NP=0, DLI=01
AFlags = 00000000 (1 byte), AWid=00, AP=0, AZ=000
BFlags = 00000000 (1 byte), BWid=00, BP=0, BZ=000
Object_Type = RESERVED (2 bytes)
Object_Length = 8 (1 byte)
Magic2 (1 byte)
```

The hex and ASCII dump of this object would look something this:

```
76 01 00 00 00 01 00 35 |v.....5|
```

The minimal valid allocated database storage object (with an Object_Name, no Object_Attributes or Object_Body) would thus be:

```
Magic1 (1 byte)
HFlags = 00100x00 (1 byte), OWid=00, NP=1, NWid=00, DLI=00
AFlags = 000xx000 (1 byte), AWid=00, AP=0, AZ=000
BFlags = 000xx000 (1 byte), BWid=00, BP=0, BZ=000
Object_Type = OPAQUE?????_BINARY (2 bytes)
Object_Length = 16 (1 byte)
Name_Length = 2 (1 byte)
Object Name (1 character + null byte) (2 bytes)
Pad (5 bytes)
```

```
Magic2 (1 byte)
```

Without the padding, that (rather useless) object would be 11 bytes long. Given the rounding requirements, it is clear that all allocated storage objects in the database must be at least 16 bytes long. A database object with a minimal Object_Body would need 12 bytes, which would need to be padded out to 16 bytes as well:

```
Magic1 (1 byte)
HFlags = 001xxxxx (1 byte)
?? correctly xfer these data to A/B flags: IFlags???? = 00x1xxxx (1 byte)
AFlags = 00000000 (1 byte), AWid=00, AP=0, AZ=000
BFlags = 00000000 (1 byte), BWid=00, BP=0, BZ=000
Object Type (2 bytes)
Object Length = 16 (1 byte)
Name Length = 2 (1 byte)
Object Name (1 character + null byte) (2 bytes)
Body Length = 1 (1 byte)
Body Data (1 byte)
Pad (4 bytes)
Magic2 (1 byte)
```

2.9. How Objects Are Grouped into a Database

2.10. Details of BRL-CAD-Specific Nongeometric Database Object Types

2.11. Details of BRL-CAD-Specific Geometric Database Object Types

2.12. Extensions for Deferred Implementation

2.13. Community Feedback on the Proposal

2.14. Database Library Application Programming Interface (API)