

Volume 6

*OGC CDB Rules for Encoding CDB Models using
OpenFlight (Best Practice)*

Open Geospatial Consortium

Submission Date: 2020-01-21

Approval Date: 2020-08-24

Publication Date: 2020-xx-xx

External identifier of this OGC® document: <http://www.opengis.net/doc/BP/CDB/openflight/1.2>

Internal reference number of this OGC® document: 16-009r5

Version: 1.2

Category: OGC® Best Practice

Editor: Carl Reed

Volume 6: OGC CDB Rules for Encoding CDB Models using OpenFlight (Best Practice)

Copyright notice

Copyright © 2020 Open Geospatial Consortium

To obtain additional rights of use, visit <http://www.opengeospatial.org/legal/>

Warning

This document defines an OGC Best Practice on a particular technology or approach related to an OGC standard. This document is **not** an OGC Standard and may not be referred to as an OGC Standard. It is subject to change without notice. However, this document is an **official** position of the OGC membership on this particular technology topic.

Document type: OGC® Best Practice

Document subtype:

Document stage: Approved

Document language: English

License Agreement

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications. This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

Table of Contents

1. Scope	10
2. Conformance	11
3. References	12
4. Terms and Definitions	13
4.1. A note on OpenFlight	13
5. Conventions	14
5.1. Identifiers	14
5.2. Schemas	14
6. CDB OpenFlight Models	15
6.1. OpenFlight File Header	15
6.2. OpenFlight Model Tree Structure	16
6.2.1. CDB Model Tree Structure	17
6.2.2. T2DModel Tree Structure	18
6.2.3. The Use of Node Names	20
6.2.4. Model Master File	21
6.2.5. Referencing Other OpenFlight Files	22
6.3. Modeling Conventions	23
6.3.1. Model Coordinate Systems	24
6.3.2. Geometry	30
6.3.3. Roof Tagging	31
6.3.4. Relative Priority	32
6.4. Model Identifiers	32
6.4.1. GSModel and GTModel Identifier	32
6.4.2. MModel Identifier	33
6.4.3. 2DModel Identifier	33
6.5. Model Zones	33
6.5.1. Definition	34
6.5.2. Global Zones	35
6.5.3. Zone Attributes	36
6.5.4. Implementation Guidelines	37
6.5.5. Model Zone Naming	40
6.5.6. Usages	42
6.6. Model Points	59
6.6.1. Definition	60
6.6.2. Usages	60
6.7. Model Conforming	65
6.7.1. Non Conformal (Absolute) Mode	67
6.7.2. Point Conformal Mode	67

6.7.3. Vertex Conformal Mode	68
6.7.4. Line Conformal Mode	68
6.7.5. Plane Conformal Mode	69
6.7.6. Surface Conformal Mode	71
6.8. Model Levels-of-Detail	72
6.8.1. Exchange LODs	74
6.8.2. Additive LODs	74
6.8.3. Significant Size	75
6.8.4. LOD Limits	76
6.8.5. LOD Generation Guidelines	78
6.9. Model Switch Nodes	78
6.9.1. Definition	78
6.9.2. Usage	79
6.10. Model Articulations	84
6.10.1. Definition	85
6.10.2. Usage	88
6.11. Model Light Points	88
6.12. Model Attributes	89
6.12.1. Definition	89
6.12.2. Vendor Attributes	90
6.12.3. Examples	91
6.13. Model Textures	91
6.13.1. Handling of Multi-textures	92
6.13.2. Default Gamma Corrections	95
6.13.3. Texture Dimension	95
6.13.4. Texture Palette	96
6.13.5. Usages	99
6.14. Model Descriptor (Metadata) Datasets	113
6.14.1. Model Name	114
6.14.2. Model Identification	114
6.14.3. Model Mass	115
6.14.4. Model Parts	116
6.14.5. Model Textures	116
6.14.6. Model Configurations	119
6.14.7. Model Composite Materials	123
Annex A: Conformance Class Abstract Test Suite (Normative)	125
A.1. Conformance class: CRS	125
A.2. Conformance Class: Tree Structure	125
A.3. Conformance Class: Modeling Conventions	127
A.4. Conformance Class: Model Zones	129
A.5. Conformance Class: Model Points	132

A.6. Conformance Class: Model Level Of Detail	133
A.7. Conformance Class: Model Switch Nodes	133
A.8. Conformance Class: Damage Status	134
A.9. Conformance Class: Model Articulations	135
A.10. Conformance Class: Model Textures	136
Annex B: OpenFlight v16.0 Technical Description – Annotated	140
B.1. Contents	140
B.2. OpenFlight® Scene Description	147
B.3. OpenFlight Concepts	147
B.4. Database Hierarchy	147
B.5. Node Attributes	149
B.6. Palettes	150
B.7. Instancing	151
B.8. Replication	151
B.9. Bounding Volumes	151
B.10. Multitexture	151
B.11. OpenFlight File Format	152
B.12. OpenFlight Record Types	152
B.12.1. Control Records	153
B.12.2. Node Primary Records	157
B.12.3. Mesh Nodes	171
B.12.4. Mesh Record	172
B.12.5. Local Vertex Pool Record	175
B.12.6. Light Point Nodes	182
B.12.7. Multitexture	212
B.12.8. Transformation Records	219
B.12.9. Bounding Volume Records	223
B.12.10. Palette Records	231
B.13. Texture Files	271
B.13.1. Texture Pattern Files	271
B.13.2. Texture Attribute Files	271
B.14. Road Path Files	280
B.15. Road Zone Files	281
B.16. Linkage Editor Parameter IDs	282
B.17. OpenFlight Opcodes	288
B.17.1. Obsolete Opcodes	290
Annex C: Summary of Changes Version 15.7	292
C.1. C - Overview	292
C.2. Format Changes	292
C.2.1. Continuation Record	292
C.2.2. Header Record	293

C.3. Mesh Nodes	293
C.4. Mesh Record	294
C.5. Local Vertex Pool Record	296
C.6. Mesh Primitive Record	301
C.7. Multitexture	302
C.7.1. Multitexture Record	303
C.7.2. UV List Record	304
C.8. Texture Attribute File	306
C.8.1. Subtexture	306
Annex D: Summary of Changes Version 15.8	307
D.1. D - Overview	307
D.2. Document Corrections	307
D.2.1. Text Record	307
D.2.2. CAT Record	308
D.3. Format Changes	308
D.3.1. Header Record	309
D.3.2. Group Record	310
D.3.3. Level of Detail Record	311
D.3.4. External Reference Record	311
D.3.5. Indexed String Record	312
D.3.6. Face Record	312
D.3.7. Mesh Record	313
D.3.8. Local Vertex Pool Record	313
D.3.9. Vertex Palette Records	316
D.4. Light Points	319
D.4.1. Light Point Appearance Palette Record	320
D.4.2. Light Point Animation Record	324
D.4.3. Indexed Light Point Record	326
D.4.4. Light Point System Record	327
D.4.5. Texture Mapping Palette Record	327
Annex E: Summary of Changes Version 16.0	329
E.1. E - Overview	329
E.2. Document Corrections	329
E.2.1. Header Record	329
E.2.2. Face Record	330
E.2.3. Mesh Record	330
E.2.4. Switch Record	331
E.2.5. Texture Mapping Palette Record	332
E.2.6. Indexed String Record	333
E.2.7. Bounding Convex Hull Record	334
E.2.8. Bounding Histogram Record	334

E.3. Format Changes	334
E.3.1. External Reference Record	334
E.3.2. Face Record	335
E.3.3. Mesh Record	335
E.3.4. Light Point Appearance Palette Record	335
E.3.5. Shader Palette Record	336
E.4. Texture Attribute File	337
E.4.1. Texture Mapping Palette Record	338
E.5. Index	338
Annex F: Revision History	339

i. Abstract

This volume defines the OpenFlight implementation requirements for a CDB conformant data store. Please also see Volume 1 OGC CDB Core Standard: Model and Physical Structure for a general description of all of the industry standard formats specified by the CDB standard. Please read section 1.3.1 of that document for a general overview.

ii. Keywords

The following are keywords to be used by search engines and document catalogues.

ogcdoc, OGC document, cdb, openflight

iii. Preface

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

iv. Submitting organizations

The following organizations submitted this Document to the Open Geospatial Consortium (OGC):

Organization name(s)

- CAE Inc.
- Carl Reed, OGC Individual Member
- Envitia, Ltd
- Glen Johnson, OGC Individual Member
- KaDSci, LLC
- Laval University
- Open Site Plan
- University of Calgary
- UK Met Office

v. Submitters

All questions regarding this submission should be directed to the editor or the submitters:

Name	Affiliation
Carl Reed	Carl Reed & Associates

Chapter 1. Scope

This volume of the CDB standard defines a set of conventions to represent 2D and 3D models based on version 16.0 of the OpenFlight Scene Description Database Specification as annotated in Appendix C of this document.

For ease of editing and review, the standard has been separated into 16 Volumes, one being a schema repository.

- Volume 0: OGC CDB Companion Primer for the CDB standard (Best Practice).
- Volume 1: OGC CDB Core Standard: Model and Physical Data Store Structure. The main body (core) of the CDB standard (Normative).
- Volume 2: OGC CDB Core Model and Physical Structure Annexes (Best Practice).
- Volume 3: OGC CDB Terms and Definitions (Normative).
- Volume 4: OGC CDB Rules for Encoding CDB Vector Data using Shapefiles (Best Practice).
- Volume 5: OGC CDB Radar Cross Section (RCS) Models (Best Practice).
- Volume 6: OGC CDB Rules for Encoding CDB Models using OpenFlight (Best Practice).
- Volume 7: OGC CDB Data Model Guidance (Best Practice).
- Volume 8: OGC CDB Spatial Reference System Guidance (Best Practice).
- Volume 9: OGC CDB Schema Package: <http://schemas.opengis.net/cdb/> provides the normative schemas for key features types required in the synthetic modeling environment. Essentially, these schemas are designed to enable semantic interoperability within the simulation context (Normative).
- Volume 10: OGC CDB Implementation Guidance (Best Practice).
- Volume 11: OGC CDB Core Standard Conceptual Model (Normative).
- Volume 12: OGC CDB Navaids Attribution and Navaids Attribution Enumeration Values (Best Practice).
- Volume 13: OGC CDB Rules for Encoding CDB Vector Data using GeoPackage (Normative, Optional Extension).
- Volume 14: OGC CDB Guidance on Conversion of CDB Shapefiles into CDB GeoPackages (Best Practice).
- Volume 15: OGC CDB Optional Multi-Spectral Imagery Extension (Normative).

Chapter 2. Conformance

This standard defines requirements for implementing OpenFlight content in a CDB compliant data store.

Requirements for 1 standardization target types are considered.

Conformance with this standard shall be checked using all the relevant tests specified in Annex A (normative) of this document. The framework, concepts, and methodology for testing, and the criteria to be achieved to claim conformance are specified in the OGC Compliance Testing Policies and Procedures and the OGC Compliance Testing web site ^[1].

In order to conform to this OGC™ interface standard, a software implementation shall choose to implement:

1. Any one of the conformance levels specified in Annex A (normative).

All requirements-classes and conformance-classes described in this document are owned by the standard(s) identified.

[1] www.opengeospatial.org/cite

Chapter 3. References

The following normative documents contain provisions that, through reference in this text, constitute provisions of this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

OpenFlight Specification version 16.4

http://www.presagis.com/products_services/standards/openflight/

Chapter 4. Terms and Definitions

This document uses the terms defined in Sub-clause 5.3 of [OGC 06-121r8], which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this standard.

For the purposes of this document, the additional terms and definitions provided in the OGC CDB Terms and Definitions document (Volume 3) apply.

4.1. A note on OpenFlight

OpenFlight (or .flt) ^[2] is a 3d geometry model file format originally developed by Software Systems Inc. for its MultiGen ^[3] real-time 3d modeling package in 1988. Originally called Flight, the format was designed as a nonproprietary 3D model format for use by real-time 3d **visual simulation** image generators. The format was later renamed to OpenFlight to denote its nonproprietary image generation (IG) usage. The MultiGen modeling package (known now as Creator ^[4]) and the OpenFlight format were rapidly adopted by the early commercial **flight simulation** industry in the later 80's and early 90's. **NASA Ames** was the first customer for the MultiGen modeling package.

The early advantage OpenFlight held over many 3d geometry model file formats (.obj, .dxf, .3ds) was its specific real-time 3d graphics industry design. This means that the format is polygon based (rather than **NURB** surfaces), and provides a real-time tree structure essential for real-time IG systems. Most early graphics file formats worried more about visual esthetics for non-real-time based rendering graphics packages such as **Wavefront Technologies**, or **Alias Systems Corporation**.

The OpenFlight file format is still widely used today in the high end real-time visual simulation industry as the standard interchange format between different IG systems, and is currently administrated by **Presagis**.

[2] From Wikipedia.

[3] <https://en.wikipedia.org/w/index.php?title=MultiGen&action=edit&redlink=1>

[4] https://en.wikipedia.org/wiki/OpenFlight#cite_note-3

Chapter 5. Conventions

This sections provides details and examples for any conventions used in the document. Examples of conventions are symbols, abbreviations, use of XML schema, or special notes regarding how to read the document.

5.1. Identifiers

The normative provisions in this standard are denoted by the URI

<http://www.opengis.net/spec/CDB/1.1/openflight/{requirement}>

All requirements and conformance tests that appear in this document are denoted by partial URIs that are relative to this base.

For the sake of brevity, the use of “req” in a requirement URI denotes the literal:

<http://www.opengis.net/spec/core/cdb/1.1/openflight>

An example might be:

req/openflight/crs

5.2. Schemas

The following XML Schemas can be found in the OGC CDB Standard schema repository:

1. Base_Material_Table.xsd
2. Composite_Material_Table.xsd
3. Configuration.xsd
4. Defaults.xsd
5. Feature_Data_Dictionary.xsd
6. Lights.xsd
7. Lights_Tuning.xsd
8. Model_Components.xsd
9. Model_Metadata.xsd
10. OpenFlight_Model_Extensions.xsd
11. Vector_Attributes.xsd
12. Version.xsd

Chapter 6. CDB OpenFlight Models

This volume of the CDB standard defines a set of conventions for representing 2D and 3D models based on version 16.0 of the OpenFlight Scene Description Database Specification as annotated in Appendix C of this document. The conventions presented here address the needs of several types of simulation clients including OTW, FLIR, NVG, CGF, radar, and laser, acoustic, magnetic, visual and thermal sensors.

Requirements Class – CRS Metadata	
/req/core/metadata	
Target type	Operations
Dependency	OpenFlight Specification
Dependency	File Management system
Requirement 1	/req/core/openflight/header-crs

6.1. OpenFlight File Header

The OpenFlight Header Record contains descriptive coordinate reference system (CRS) metadata about the manner vertices are encoded, and how these vertices are applied to an earth model and a projection type. On the other hand, the CDB standard itself mandates a prescribed set of conventions in this regard.

Requirement 1	http://www.opengis.net/spec/CDB/1.1/openflight/header-crs
	<p>As a result; the following OpenFlight Header records <i>SHALL</i> be set as follows:</p> <p>Projection Type:</p> <ul style="list-style-type: none">• Flat Earth, value 0: This is the type of projection used by most CDB Models, GTModel, GSModel, and MModel.• Geodetic, value 5: This is the type of projection used by T2DModels. <p>Earth Ellipsoid Model:</p> <ul style="list-style-type: none">• WGS-84, value 0: This is the Earth model to use with T2DModels. The field is ignored with other CDB Models.

6.2. OpenFlight Model Tree Structure

The internal structure of OpenFlight models is a tree structure that consists of nodes having child nodes as well as sibling nodes ^[5]. This type of tree structure is called a directed acyclic graph, or DAG. The general tree structure of a Model resembles that of Figure 6-1: General OpenFlight Tree Structure.



Figure 6-1. General OpenFlight Tree Structure

The CDB standard uses Group Nodes to arrange Models in a hierarchical manner. This way of organizing models helps identify components of interest.

The CDB standard refers to a number of OpenFlight nodes to store meaningful data for simulation client-devices. For a complete list of nodes supported by the CDB standard, see Appendix C. The nodes listed here are the ones referred to by one or more CDB conventions.

An example of an ancillary record is the OpenFlight comment record. A comment record may appear once, anywhere after a node's primary record. The CDB standard relies on comment records to extend the definition of OpenFlight nodes. Instead of using the Extension Record to create new primary and ancillary records, the CDB standard uses comments to store the extra attributes required by the specialization of existing OpenFlight nodes ^[6]. Comment records were chosen over OpenFlight extensions in order to minimize any changes to the Creator tool or the need to develop a plug-in to Creator. Using this approach, anybody can create CDB-compliant models using a plain version of Creator. Nevertheless, the development of Creator CDB plug-ins would improve modeler's efficiency. Such plug-ins could, for instance, offer a menu-based GUI to allow modelers to enter and edit CDB comments, while ensuring that syntax and conventions are fully adhered to.

The text contained in the comment record is formatted using the XML notation.

Requirements Class – tree structure

/req/core/tree-structure	
Target type	Operations
Dependency	OpenFlight Specification
Dependency	CDB File Management system
Requirement 2	/req/core/openflight/model-global-zone
Requirement 3	/req/openflight/2dmodel
Requirement 4	/req/openflight/2dmodel-rules
Requirement 5	/req/openflight/2dmodels
Requirement 6	/req/openflight/xref

6.2.1. CDB Model Tree Structure

Based on Figure 6-1 above, the internal structure of CDB Models resemble this one:



Figure 6-2. Internal Structure of CDB Models

Requirement 2	req/model-global-zone
	<p>All CDB Models <i>SHALL</i> have a global zone as their root node. This node identifies the model. Global zones are defined in section 6.5.2 below.</p>

6.2.2. T2DModel Tree Structure

A T2DModel being a collection of 2DModels, each individual 2DModel occupies its own subtree of the graph. The general structure of a T2DModel is as follows:

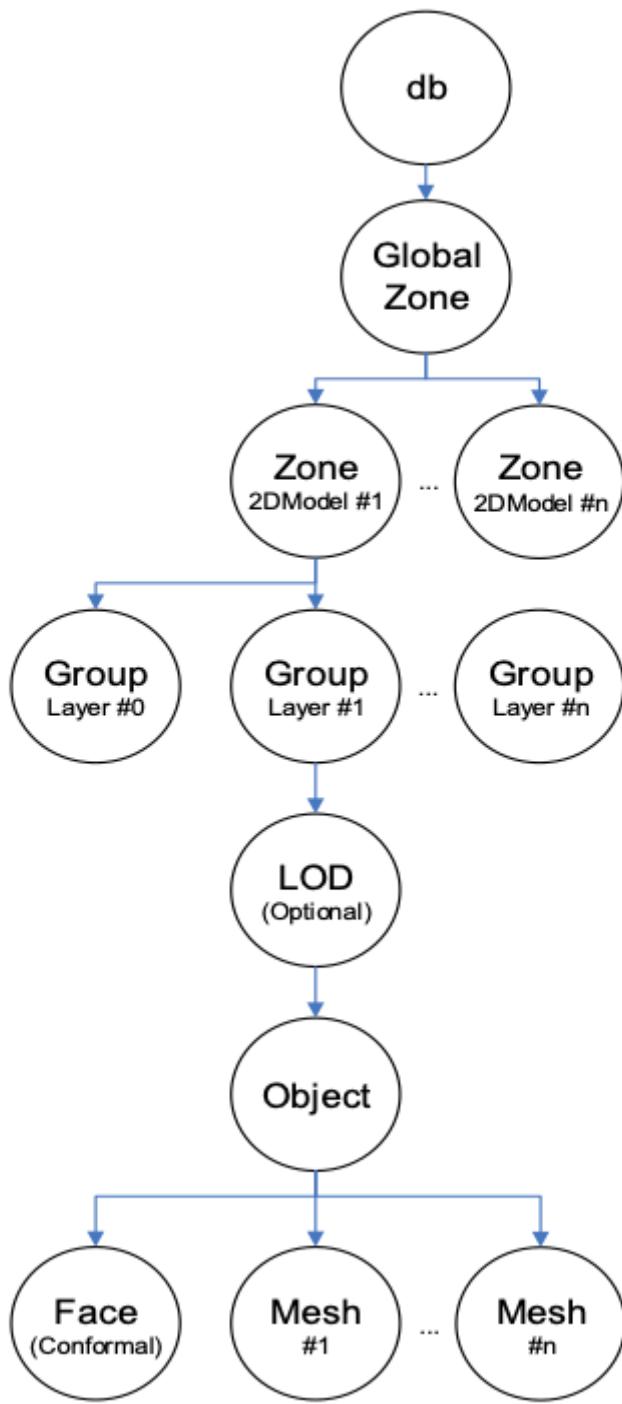


Figure 6-3. Internal Structure of T2DModels

Requirement 3	req/openflight/2dmodel
----------------------	------------------------

Each 2DModel *SHALL* be implemented as a Model Zone (section 6.5) with its own subgraph. 2DModels are disjoint and non-overlapping.

A 2DModel is comprised of multiple layers; the layer number is expressed by the group's relative

priority (section 6.3.4). Each layer has an optional LOD node followed by a fixed hierarchy of regular OpenFlight Object, Face, and Mesh nodes.

Restrictions

T2DModels being an alternate representation of the terrain and its imagery and materials, a number of restrictions are necessary to ensure client devices can consume the dataset efficiently.

Requirement 4	req/openflight/2dmodel-rules
	A 2DModel <i>SHALL</i> have at least two layers, layer 0 and layer 1.
	Layer 0 <i>SHALL</i> always empty because it represents the terrain on top of which subsequent layers are applied.
	Each layer is composed of zero, one, or more OpenFlight Object nodes.

All Face and Mesh nodes share exactly the same set of graphic attributes (color, textures, material, and other flags). Stated differently, the Face and Mesh nodes provide the shape of the layer while the Object node controls its appearance.

Subfaces are not permitted because coplanar geometry is implemented through layers.

Node Attributes

Requirement 5	req/openflight/2dmodels
	For T2DModels, node attributes (section 6.12) <i>SHALL</i> be permitted only at the zone levels; that is, at the global zone or at the individual 2DModel zones. Node attributes are not permitted at the Group, LOD, Object, Face, and Mesh node levels.

6.2.3. The Use of Node Names

Although the CDB standard defines naming conventions for objects stored in an OpenFlight file, the standard does not constrain the OpenFlight node names themselves. Instead, the CDB standard defines names that are assigned via XML tags stored in the comment record.

The following question arises, “Why not establish a set of CDB conventions around node names?” The answer lies primarily around constraints imposed by tools used to edit/create OpenFlight files. Tools such as Creator require unique node names throughout the OpenFlight file. The OpenFlight format specification itself does not state that node names must be unique; however, a tool such as Creator prevents the modeler from entering the same node name twice.

To circumvent this limitation, the CDB standard provides naming conventions through XML tags inserted in the comment record following a node's primary record. This way of doing things leaves modelers with the needed freedom in naming nodes. The CDB standard defines how to organize a model; all extra object attributes that do not fit in the current OpenFlight records are stored in the comment record, including object names.

Here is an example of XML tags stored in the comment record for a Group Node.

Table 6-1: Sample XML Tag Used in a Comment Record

```
<CDB:Zone  
name = "zone name"  
/>
```

All XML tags defined by the CDB standard belong to a single XML namespace that is appropriately named CDB ^[7].

6.2.4. Model Master File

A Model Master file is an OpenFlight file that contains only external references to other OpenFlight files. The purpose of the master file is to ensure a Model is seen as a single “object” even though its constituents are stored in separate files. The use of a model master file provides a convenient means for modelers to reference all of the constituent files that make up the model. There is no other purpose associated to the master file.

The concept of a Model Master file is used in a single case, to regroup all representations (all LODs) of a geotypical model into a single OpenFlight file. However, the concept applies to all types of CDB Models. The concept can also be used to regroup a model's shell with its interior. For this reason, expect new usage of the Model master file in future version of the Standard.

The master file is useful in two circumstances: when modelers create or edit Models, and when client devices want to discover at once all constituents of Models.

For modelers, it is useful (if not required) to edit a model using a single source file to present a coherent view of the model as a whole. For this reason, a master file with a set of LOD-XRef nodes is perfect to assemble and present a unified view of the model to edit.

Figure 6-4: Typical Structure of a Model Master File presents the general structure of a master file.



Figure 6-4. Typical Structure of Model Master File

The value found in the Significant Size field of the above LOD nodes matches the values found in Table 3-1 of Volume 1 CDB Core Standard: CDB LOD vs. Model Resolution. The next section provides details on XRef nodes themselves.

6.2.5. Referencing Other OpenFlight Files

Requirement 6	req/openflight/xref+
An OpenFlight external reference (XRef) node is used to refer to another OpenFlight file. The reference is made by specifying the filename and its path (absolute or relative). The CDB Standard requires that all references <i>SHALL</i> be made using a relative path.	

The XRef node (and its External Reference record) supports a number of options: Override flags, View-As-Bounding-Box flag, and Target Node Name. The CDB standard supports none of these options.

Here are two cases to illustrate the use of XRef nodes.

Models Straddling Multiple Files

In the case of GTModels, GSModels, and MModels, the OpenFlight geometry can straddle multiple files. It could be seen in a moving model (e.g., a helicopter) whose pilot could be stored in a separate file. In that case, the file containing the pilot resides in the same directory as the file containing the helicopter itself. The helicopter would be stored in file 1:

```
\CDB\MMModel\600_MMModelGeometry\1_Platform\2_Air\225_United_States\21.Utility_Helo\1_2_225_21_x_x_x\600_S001_T001_1_2_225_21_x_x_x.flt
```

The pilot would be stored in file 2:

```
\CDB\MMModel\600_MMModelGeometry\1_Platform\2_Air\225_United_States\21.Utility_Helo\1_2_225_21_x_x_x\600_S001_T002_1_2_225_21_x_x_x.flt
```

The XRef node in file 1 would contain the following string:

```
.\600_S001_T002_1_2_225_21_x_x_x.flt
```

where 1_2_225_21_x_x_x is the complete DIS code of the helicopter.

Models with Multiple Model-LODs

This is the case of the master file of a geotypical model. The master file (known as the GTModelGeometry Entry File) refers to all levels of details of the geometry files that reside in different sub-directories. Assuming a geotypical model representing a gothic church, the master file itself would reside in a directory such as this one:

```
\CDB\GTModel\500_GTModelGeometry\A_Culture\L_Misc_Feature\015_Building\500_S001_T001_AL015_050_Church-Gothic.flt
```

The targets of the XRef nodes would all reside in directories such as these:

```
\CDB\GTModel\500_GTModelGeometry\A_Culture\L_Misc_Feature\015_Building\Lnn\510_S001_T001_Lnn_AL015_050_Church-Gothic.flt
```

The resulting strings to use in the XRef nodes in the master file would resemble this:

```
.\Lnn\510_S001_T001_Lnn_AL015_050_Church-Gothic.flt
```

where Lnn corresponds to the LOD the XRef file resides in.

6.3. Modeling Conventions

Requirements Class – Modeling Conventions

/req/model-conventions

Target type	Operations
-------------	------------

Dependency	OpenFlight Specification
------------	--------------------------

Requirement 7	/req/openflight/crs-models
Requirement 8	/req/openflight/model-origin
Requirement 9	/req/openflight/t2-model-coordinates
Requirement 10	/req/openflight/roll-pitch-yaw
Requirement 11	/req/openflight/geometry
Requirement 12	/req/openflight/geometry-layer-constraint

6.3.1. Model Coordinate Systems

Requirement 7	req/openflight/crs-models
CDB Models <i>SHALL</i> use the same coordinate system convention as OpenFlight ^[8] . The X-axis traverses the model from left to right, the Y-axis goes from the back to the front and the Z-axis extends from the bottom to the top of the model.	

Figure 6- 5: Model Coordinate System is a screenshot ^[9] of Creator ^[10] showing a CAD view of a semi-transparent cube.



Figure 6-5. Model Coordinate System

Origin

The location of the model origin is defined in the following manner:

Requirement 8

req/openflight/model-origin

In the XY plane, the origin *SHALL* be located at the center of the bounding rectangle.

Along the Z axis, the origin *SHALL* be selected to allow the model to be correctly positioned on ground for ground-related models, or on a water plan for surface and subsurface platforms.

The following examples will illustrate the above two rules.

Fixed wing aircraft – A KC-130 Hercules is illustrated below with its landing gears fully extended.



Figure 6-6. Coordinate System - Aircraft

Helicopter – An AH-1W Super Cobra is shown below. Note that no equipment is mounted on its winglets.



Figure 6-7. Coordinate System - Helicopter

Surface ship – Shown below is the Arleigh Burke DDG-51 guided missile destroyer. Note how the XY plane defines the waterline.



Figure 6-8. Coordinate System - Ship

Land Platform – The M1A2 Abrams is a main battle tank shown here with its desert skin.



Figure 6-9. Coordinate System - Ground Based Model

Lifeform – A human lifeform (a soldier) is presented here.



Figure 6-10. Coordinate System - Lifeform

Cultural Feature – When a Model represents a cultural feature, the origin is the point of insertion of the model into the ground. In general, the XY plane (at Z = 0) delimits the basement from the first floor.



Figure 6-11. Coordinate System Cultural Feature

Power Pylon – In the case of an electricity pylon, the front (and back) of the model is aligned with the general direction of the attached wires.



Figure 6-12. Coordinate System - Power Pylon

Local Coordinate Systems

Most OpenFlight nodes can have a local transformation used to create a local coordinate system. The origin and orientation of this coordinate system are expressed by a single transformation matrix.

When a transformation is specified for a node's primary record, only the matrix record (opcode 49) is considered by CDB client-devices. All other transformation records (opcode 76, 78, 79, 80, 81, 82

and 94) are discarded ^[11].

Units

GSModels and GTModels

The MODL attribute of the GSFeature and GTFeature datasets are used to reference GSModels and GTModels. In turn, the position of GSModels and GTModels are obtained from the coordinates of each point of the associated vector data set; these coordinates are interpreted as the latitude (y), longitude (x), and elevation (z) coordinates that position the model within the CDB ^[12].

GSModels and GTModels are drawn to real-world dimensions using meters as their unit of measurement.

MModels

MModels are usually internally activated and positioned by the client-devices in response to a running simulation.

In the case of Moving Model location features, the MMDC attribute of the GTFeature dataset is used to reference a MModel. Otherwise, the MModel behaves exactly as a GTModel as described in section 6.3.1.3.1 above.

MModels are drawn to real-world dimensions using meters as their unit of measurement.

T2DModels

Requirement 9	req/openflight/t2-model-coordinates
Vertex latitude (y) and longitude (x) coordinates <i>SHALL</i> be expressed in decimal degrees. The values <i>SHALL</i> be relative to the file's (implicit) origin which is the south-west corner of the tile.	

Note however, that the file's origin and size are implicitly defined by the tile position and the tile level-of-detail. The absolute position of each vertex is obtained by adding the vertex relative value to the tile origin.

T2DModels are used to model features that have no significant height with respect to the neighboring terrain; they are generally conformed to the terrain using the “Surface Conformal Mode” as explained in section 6.7, Model Conforming. Note that the vertices of T2DModels need not have Z-coordinate values that are always zero. For example, it is permissible to model a road linear that is modestly elevated with respect to the neighboring terrain. Client-devices must be capable of handling T2DModels that are either perfectly surface-conformed to the terrain (all vertices have Z=0) or modestly elevated (vertices with Z>0) with respect to the terrain. Note that surface-conformed models with vertices with Z-coordinate values less than zero are, by definition, below the terrain.

Roll, Pitch, Yaw

Requirement 10	req/openflight/roll-pitch-yaw+
Pitch, Roll and Yaw angles refer to rotations around the X, Y, and Z axes, respectively. Angles <i>SHALL</i> be measured in degrees. The Roll and Yaw angles vary from ± 180 degrees while the Pitch angle is limited to the range ± 90 degrees.	

6.3.2. Geometry

Requirement 11

req/openflight/geometry

Implementers of the CDB standard *SHALL* adhere to the following set of constraints, rules and guidelines when creating the geometry of Models.

- Create convex polygons. All lines joining any two points in the interior of the polygon also lie in the interior.
- Create coplanar vertices. All of the vertices defining a polygon should lie in the same plane (required by virtually all rendering engines).
- A polygon's front face *SHALL* be defined by a counter-clockwise ordering of its vertices.
- Avoid T vertices. T-vertices occur when two or more adjacent polygons share an edge, and the polygons do not share a common vertex on that edge.
- Avoid coplanar faces. A coplanar face is a polygon that lies directly on top of another polygon. Z-buffer fighting can occur when a Z-buffer system cannot resolve which polygon to display on top. The CDB standard strongly recommends changing one of the coplanar faces to be a subface of the other in the hierarchy so that the client device such as an Image Generator can draw the faces in the correct order. An alternative is to use the Relative Priority field to implement layering (see section 6.3.4 below).
- Favor mesh over individual polygons. The OpenFlight Mesh record allows the creation of triangle fans, triangle strips, quadrilateral strips, and indexed face sets. This construct is by far preferable to individual polygons built using the OpenFlight Face record.
- Make use of instancing. Avoid repeating identical pieces of geometry. Create one object and repeat it by having multiple instances in different locations..

6.3.3. Roof Tagging

OpenFlight has a provision for tagging polygons that are part of a roof. The Roofline flag can be found in both the Face and Mesh records. This flag is useful to apply a geospecific texture to the roof. When the flag is set, the client-device can discard the texture referenced by the polygon and use instead the geospecific texture that appears on the terrain.

More generally, the Roofline flag is used to tag any polygon whose texture can be replaced with a geospecific texture.

6.3.4. Relative Priority

The Relative Priority field appears in four OpenFlight primary records:

Face Record

Mesh Record

Object Record

Group Record

The field is required to implement layering, a method to handle coplanar geometry. By using the Relative Priority field, the modeler can construct more complex coplanar geometry than what is possible with sub-faces.

Here is the definition of the field according to OpenFlight 16.0:

Relative priority specifies a fixed ordering of the node relative to its sibling nodes. Ordering is from left (lesser values) to right (higher values). Nodes of equal priority may be arbitrarily ordered. All nodes have an implicit (default) relative priority value of zero.

The CDB standard further restricts the use of the Relative Priority field for complex coplanar geometry as follows. Using relative priorities, the modeler defines 'n' layers of coplanar geometry. Layers are numbered from 0 to 'n-1'.

Requirement 12

[req/openflight/geometry-layer-constraint](#)

Layer 0, the base layer, *SHALL* contain geometry that completely encompasses the geometry of subsequent layers. Other layers *SHALL* be processed in order, one after the other. A layer is made of one or more nodes; all nodes of a given layer *SHALL* have the same relative priority.

6.4. Model Identifiers

6.4.1. GSModel and GTModel Identifier

Although the name of the global zone of a GS or GT model is arbitrary, it is strongly suggested to use the value of their MODL attribute to name the global zone.

For instance, a GTModel stored in a file called

D500_S001_T001_AL015_004_Castle.flt

Would have its global zone identified like this:

```
<CDB:Zone name="Castle"/>
```

6.4.2. MModel Identifier

The MModel identifier is the common name of the moving model or the name of its part. The actual name is not covered by the standard. An example of a MModel identifier is “M1A2” for the M1 Abrams tank whose DIS Entity Type is 1_1_225_1_1_3_0.

For instance, a moving model stored in a file called

D600_S001_T001_1_1_225_1_1_3_0.flt

Would have its global zone identified like this:

```
<CDB:Zone name="M1A2"/>
```

6.4.3. 2DModel Identifier

A T2DModel itself does not need to be identified per se. However, it is required to identify individual 2DModels inside a T2DModel. This is done by assigning a unique zone name to each 2DModel. In addition, a 2DModel must have the same zone name across LODs and across tiles. Note that when a 2DModel is clipped by tile boundaries, each of the clipped model fragments will appear in distinct OpenFlight files of the T2DModel Dataset. When clipped, the 2DModel Identifier must appear once, in each of the T2DModel Tile-LODs. This is necessary to identify the parts of a 2DModel that straddle multiple Tile-LODs.

6.5. Model Zones

Requirements Class – Model Zones

/req/core/model-zones

Target type	Operations
Dependency	OpenFlight Specification
Requirement 13	/req/openflight/model-zone-bounding-box
Requirement 14	/req/openflight/zone-name
Requirement 15	/req/openflight/global-zone
Requirement 16	/req/openflight/hot-spot-temperature

Requirement 17	/req/openflight/model-footprint-zones
Requirement 18	/req/openflight/model-footprint-hierarchy
Requirement 19	/req/openflight/model-cutout-zones
Requirement 20	/req/openflight/model-cutout-geometry
Requirement 21	/req/openflight/model-pseudo-interior-zone
Requirement 22	/req/openflight/model-interior-zone

The concept of a model zone is of the utmost importance when creating models, particularly those used in military simulation applications.

A model zone represents a component ^[13] of interest on the Model. A model zone (as well as the component it represents) occupies a certain volume and is delimited by a bounding box. At least one simulator subsystem must be interested in a specific component to justify the creation of a corresponding zone. Examples of zones are a turret on a tank, or an engine on a platform, or an entrance door on a building, etc.

Since the model itself is of interest to the simulation, it will have at least one zone, the global zone. That will be the case for most Models used as cultural features; they will have a single zone. However, Models used as moving models will typically be subdivided into several zones.

To implement the concept of model zones, the CDB standard uses the OpenFlight Group Node. Firstly, a Group Node can have child nodes to represent its own geometry as well as other zones. Secondly, a Group Node can have a bounding volume encompassing its child nodes and that can be used to represent the volume corresponding to the zone.

6.5.1. Definition

A Model Zone is an OpenFlight Group Node with a mandatory Bounding Box and the following XML tags in the comment field.

Requirement 13	req/openflight/model-zone-bounding-box+
-----------------------	---

To be a OpenFlight Group Node a Model Zone *SHALL* have mandatory Bounding Box.

Table 6-2: XML Tags for Zones

```
<CDB:Zone name="name" volume="closed|open">
  ...
</CDB:Zone>
```

Requirement 14	req/openflight/zone-name
The Zone Name is mandatory.	

Remember that if the zone exists, it is because it has its importance for at least one client device. In general, all client devices interested in a zone will use its name to identify and control it.

The volume attribute is optional and specifies whether the zone represents a closed or open volume. By default, a zone represents a closed volume.

[Table 6-3](#) lists the OpenFlight records required to represent a zone.

Table 6-3: OpenFlight Records for a Zone

GROUP
MATRIX (optional)
BOUNDING BOX (mandatory)
COMMENT (mandatory)

Note the use of the MATRIX record. It is necessary when the zone has a different position or orientation than its parent node. A zone can be thought of as a separate Model in itself. A zone has a natural orientation and its local coordinate system must indicate where their front, right, and back sides are. A zone is subject to the same convention as the model itself regarding the orientation of its coordinate system.

6.5.2. Global Zones

Requirement 15	req/openflight/global-zone
A CDB-compliant Model <i>SHALL</i> have at least one zone that encompasses the whole model and that is called the model global zone. The global zone is mandatory.	

Figure 6-13 illustrates the location of the global zone in the graph hierarchy.



Figure 6-13. Model Global Zone

The global zone contains the name of the Model contained in the OpenFlight file.

6.5.3. Zone Attributes

A Model Zone can have any number of attributes using the general mechanism described later in section 6.12, Model Attributes. However, two specific attributes are described here because of their particular relevance to the concept of Model Zone.

Material

The Material attribute provides an indication of the principal material the zone is made of. Since the majority of man-made models are made of one principal material as well as several less important materials, it is strongly suggested to use the Material attribute in the model global zone to specify what that principal material is.

The Material attribute is an index into the Composite Material Table located within the Model Descriptor file described in section 6.14. The value of the Material attribute is a strictly positive integer. The syntax of the XML tag is:

```
<CDB:Zone>  
  <Material> index </Material>  
</CDB:Zone>
```

The Material attribute can also be assigned to OpenFlight Object, Face, and Mesh nodes. The preferred syntax would be the following:

```
<CDB:Object>  
  <Material> index </Material>  
</CDB:Object>  
  
<CDB:Face>  
  <Material> index </Material>  
</CDB:Face>  
  
<CDB:Mesh>  
  <Material> index </Material>  
</CDB:Mesh>
```

However, for compatibility with version 3.1 and 3.0 of the CDB standard, a simplified (but deprecated) syntax is still supported for Object, Face, and Mesh nodes when the Material is the only attribute.

```
index  
</CDB:Material>
```

Temperature

When the zone has a heat source, such as an engine, it is known as a Hot Spot. The maximum temperature the zone can reach is specified using the Temperature attribute as shown here:

Table 6-4: XML Tags for Hot Spots

```
<CDB:Zone>  
  <Temperature> maximum temperature </Temperature>  
</CDB:Zone>
```

Requirement 17 req/openflight/hot-spot-temperature+

The temperature *SHALL* be expressed in Celsius degrees. Only integer values are permitted.

\CDB\Metadata\Model_Components.xml supplies a comprehensive list of zones that are candidates for hot spots. Typical hot spot names are Engine and Chimney to name only these two. Other zones that are of interest for hot spots simulation are wings leading edge and other surfaces subjected to friction.

6.5.4. Implementation Guidelines

This section provides a set of guidelines to implement the concept of model zones. The guidelines provided here are also applicable to Model Points described in section 6.6.

A zone is made of at least one Group Node.



Figure 6-14. Simple Zone

A zone may have an optional articulation by adding a DOF node.



Figure 6-15. Articulated Zone

To simplify the following diagrams, we will use a single circle to represent a zone, whether the zone is a single Group Node, or a pair of group and DOF nodes.

In general, a zone has a graphical representation as well as other child zones.



Figure 6-16. Zone Hierarchy

The graphical representation of a zone is itself subject to several possible implementations using various OpenFlight nodes.

The simplest way to associate a graphical representation to a zone is to use an Object node with a combination of graphic primitives available in OpenFlight: polygons, triangles, quads, and meshes.



Figure 6-17. Simple Zone Graphical Representation

The modeler is also free to use a combination of LOD and Switch nodes to control the graphical representation of a zone.

For instance, an LOD node inserted before the object node is useful to inform the client device on how significant the graphical representation is.



Figure 6-18. Additive LOD to Control the Graphical Representation

If the modeler wants to provide two (or more) graphical representations for a zone, he should use two (or more) LOD nodes.



Figure 6-19. Exchange LODs to Select the Graphical Representation

Levels of details are discussed in length in Section 6.8, Model Levels-of-Detail.

If the modeler has several distinct graphical representations for the zone, he is also free to use a switch node to select between these representations.



Figure 6-20. Switch Node to Select the Graphical Representation

CDB Switches are discussed in depth in Sections 6.9, Model Switch Nodes.

6.5.5. Model Zone Naming

A Model Zone Node is uniquely and unambiguously identified by concatenating with backslashes (\) the names of all Model Zones traversed to reach it. When sibling CDB nodes have identical names, their name is appended with a sequence number in square brackets. Nodes are numbered starting with 1. Siblings are sorted in ascending order according to their X, Y, and Z coordinates. The leftmost sibling has the smallest XYZ coordinate while the rightmost sibling node has the largest XYZ coordinate. As a result, identical sibling CDB nodes are sorted from left to right (X-axis), then back to front (Y-axis), then bottom to top (Z-axis).

The following example provides a sample of Model Zone and Model Point names that would be used for a tactical fighter aircraft; the fighter has two pylons per wing, each pylon having 3 attach points. The resulting paths to each Model Zones and Model Points are as follows:

```
\Fighter  
\Fighter\Wing[1]  
\Fighter\Wing[1]\Pylon[1]  
\Fighter\Wing[1]\Pylon[1]\Attach_Point[1]  
\Fighter\Wing[1]\Pylon[1]\Attach_Point[2]  
\Fighter\Wing[1]\Pylon[1]\Attach_Point[3]  
\Fighter\Wing[1]\Pylon[2]  
\Fighter\Wing[1]\Pylon[2]\Attach_Point[1]  
\Fighter\Wing[1]\Pylon[2]\Attach_Point[2]  
\Fighter\Wing[1]\Pylon[2]\Attach_Point[3]  
\Fighter\Fuselage  
\Fighter\Fuselage\Attach_Point  
\Fighter\Wing[2]  
\Fighter\Wing[2]\Pylon[1]  
\Fighter\Wing[2]\Pylon[1]\Attach_Point[1]  
\Fighter\Wing[2]\Pylon[1]\Attach_Point[2]  
\Fighter\Wing[2]\Pylon[1]\Attach_Point[3]  
\Fighter\Wing[2]\Pylon[2]  
\Fighter\Wing[2]\Pylon[2]\Attach_Point[1]  
\Fighter\Wing[2]\Pylon[2]\Attach_Point[2]  
\Fighter\Wing[2]\Pylon[2]\Attach_Point[3]
```

Here is how to interpret some of these paths:

The global zone is identified as \Fighter

The left wing is \Fighter\Wing[1]

The leftmost attach point is \Fighter\Wing[1]\Pylon[1]\Attach_Point[1]

The rightmost attach point is \Fighter\Wing[2]\Pylon[2]\Attach_Point[3]

There is a single attach point on the fuselage, \Fighter\Fuselage\Attach_Point

The inner pylon on the left wing is \Fighter\Wing[1]\Pylon[2]

The inner pylon on the right wing is \Fighter\Wing[2]\Pylon[1]

6.5.6. Usages

Model Landing Zones

Landing zones are used primarily by the Computed Generated Forces (CGF) sub-system of the simulator. The landing zone information can be used during the set-up of mission scenarios since it provides CGF the location of known landing zones. Typically, landing zones are used to specify the location and dimension of helipads, aircraft carrier decks, etc. \CDB\Metadata\Model_Components.xml lists several CDB Components that can act as landing zones.

Landing zones must have a bounding box that tightly fits the landing area. If required by the geometry of the landing zone, the modeler should create a local coordinate system that is axially oriented with the landing zone. Inserting a MATRIX record after the GROUP record does this.

The width and the length of the landing zone can be extracted by the client-device from the bounding box associated with the Group Node representing the zone. The landing zone geometry must be located under the Group Node to obtain meaningful dimensions.

Model Footprint Zones

A Model Footprint ^[14] conceptually represents the footprint (i.e., the terrain surface outline) of a model on the ground. The Model Footprint is modeled as a set of OpenFlight Face or Mesh records.

Requirement 17

req/openflight/model-footprint-zones

Client-devices *SHALL* assume that the geometry that is associated with a Model Footprint Zone is hidden, regardless of the value of the OpenFlight *Hidden* flag of associated geometry. However, CDB content creation tools *SHALL* set the *Hidden* flag of the associated geometry ^[15].

The footprint geometry should be terrain conformed using the “Surface Conformal Mode” as

explained in section 6.7, Model Conforming. This instructs client-devices to conform this footprint to the underlying terrain altimetry, regardless of its level-of-detail.

The Model Footprint is the set of polygons or meshes that result from the intersection of the model geometry with its XY plane ^[16].

A polygon that is tagged as Model Footprint can be used by client-devices to identify the portion of the terrain that is covered by a model. The Cultural Footprint can be used by client-devices such as:

- Map generators that may not be interested in the full 3D geometry of a Model.
- Procedural SE generation software that may use model footprints to automatically extrude such footprints into 3D models.
- Simulation of ground-based SAF entities that would use model footprints to avoid collisions with features such as buildings and trees.

Requirement 18

req/openflight/model-footprint-hierarchy

The CDB standard requires that the Model Footprint *SHALL* be placed under a CDB Footprint Zone node.

This CDB node facilitates the identification and discovering of footprints by client-devices. The subgraph representing the Footprint is presented here.



Figure 6-21. Footprint Zone Structure

The Footprint zone is followed by an OpenFlight Object node with the necessary OpenFlight Face or Mesh nodes containing the footprint itself. All Face/Mesh nodes must have their *Terrain Culture Cutout* flag set. A Footprint zone is defined by the following XML tags.

Table 6-5: Footprint Zone XML Tags

```
<CDB:Zone name = "Footprint"/>
```

Model Cutout Zones

A Model Cutout Zone conceptually represents clipping geometry that is used to cut out the terrain from a 2DModel or a 3DModel. Cutouts are typically used in conjunction with model interiors and tunnels. The Model Cutout geometry defines a simple 3D convex volume (open or closed). A typical implementation of a Model Cutout Zone for a modeled building would consist of a simple cube. Similarly, the Model Cutout Zone for a tunnel entrance would consist of a simple cylinder or a partially-open cube (see Figure 5-9: Modeling of Wells, Overhanging Cliffs and Tunnels).

Requirement 19

req/openflight/model-cutout-zones+

The Model Cutout is modeled as a set of OpenFlight Face or Mesh records. Client-devices *SHALL* assume that the geometry that is associated with a Model Cut-Out Zone is hidden and cut-out.

The Model Cutout is modeled as a set of OpenFlight Face or Mesh records. Client-devices are required to assume that the geometry that is associated with a Model Cut-Out Zone is hidden and cut-out. Client-devices should ignore the value of the OpenFlight *Hidden* and *Terrain Culture Cutout* flags of associated geometry. However, CDB content creation tools are required to set the *Hidden* and *Terrain Culture CutOut* flags of the associated geometry ^[17].

The Model Cutout geometry should be terrain conformed using the “Surface Conformal Mode” as explained in the 6.7, Model Conforming. This instructs the client-device to conform the Model Cutout to the underlying terrain altimetry, regardless of its level-of-detail.

It is specified using the following XML syntax:

Table 6-6: XML Tags for Landing Zones

```
<CDB:Zone name="Cutout">
```

Polygons or meshes that are tagged as Model Cutout can be used by client-devices to identify the portion of the terrain that needs to be removed in order to reveal the interior of the model (say a building interior or a tunnel interior). The Model Cutout is necessary for models straddling the terrain surface and whose interior is modeled and viewed from within. The reason for this is that when the model is altitude-conformed onto the terrain, a hole must be cut into the terrain-LOD, so that the terrain itself does not visually interfere with the modeled building or tunnel interior.

Requirement 20

req/openflight/model-cutout-geometry

The CDB standard requires that the Cutout geometry *SHALL* be placed under a CDB Model Cutout Zone node.

This CDB node facilitates the identification and discovery of model cutouts by client-devices. The subgraph representing the cutout is presented here.



Figure 6-22. Cutout Zone Structure

Model Interior Zones

This section focuses on how to represent the interior of Models for an intelligent use by client-devices.

A Model is composed of 2 parts: a shell, and an optional interior. The shell contains both the exterior and the pseudo-interior. Client-devices need only access the shell if they do not need to penetrate and interact with the interior of the models; otherwise, they require both the shell and the interior. The shell of a Model is stored in five (5) datasets:

- ModelGeometry
- ModelTexture
- ModelSignature
- ModelDescriptor

- ModelMaterial

The optional model interior is stored in four (4) datasets:

- ModelInteriorGeometry
- ModelInteriorTexture
- ModelInteriorDescriptor
- ModelInteriorMaterial

Refer to CDB Standard Volume 7 OGC CDB Data Model Guidance section 6.5 for guidelines on Handling of Model Interiors.

Model Pseudo-Interior Zone

The pseudo-interior is the portion of the shell that contains geometry also represented in the interior dataset. This geometry represents what is visible from the outside and is necessary to ensure the integrity and completeness of the shell.

Requirement 21

[req/openflight/model-pseudo-interior-zone](#)

Since the pseudo-interior is a placeholder for the real interior, it *SHALL* be placed under its own subgraph and identified by a CDB zone whose name is “Interior”.



Figure 6-23. Model Shell Structure

The name “Interior” is a reserved component name allowing a client-device to identify the node that is to be replaced by an entire dataset, namely the ModelInteriorGeometry dataset. The pseudo interior is mutually exclusive with the real interior defined in section 6.5.6.4.2, Model Interior Zone,

below.

Figure 6-23 also illustrates how to structure the shell of a Model that has a real interior. The model is divided in three components: the footprint of its exterior, the geometry of its exterior, and the geometry of its pseudo-interior. Therefore the names of these three components are “Footprint”, “Exterior”, and “Interior” as illustrated by the following XML tags.

Table 6-7: Shell Zones XML Tags

```
<CDB:Zone name = "Footprint"/>  
<CDB:Zone name = "Exterior"/>  
<CDB:Zone name = "Interior"/>
```

Footprints were discussed earlier in section 6.5.6.2, Model Footprint Zones.

Model Interior Zone

Requirement 22	req/openflight/model-interior-zone+
	The Model interior itself <i>SHALL</i> have a global zone whose name is “Interior”.

The Model interior itself must have a global zone whose name is “Interior”. Accordingly, the graph of the interior of the model will present the following structure. Note that real interior must not include the modeled representation of the shell.



Figure 6-24. Model Interior Structure

The Interior zone contains one or more floors as well as the partitions separating these floors. An Interior zone is defined by the following XML tags.

Table 6-8: Interior Zone XML Tags

```
<CDB:Zone name = "Interior">  
  <Ground_Floor> index </Ground_Floor>  
</CDB:Zone>
```

The `<Ground_Floor>` is optional. It contains the index of the Floor that represents the ground floor of the model interior. By default, the ground floor is floor number 1.

The subgraph representing the Interior zone has the following structure.



Figure 6-25. Interior Zone Structure

The Interior zone has two (2) kinds of child nodes: Floor and Partition. The Interior has at least one Floor. When the Interior has several Floors, the separating Partitions appear as siblings of the Floor nodes. These Partitions contain external Apertures that connect two Rooms on different Floors. These external Apertures are later referenced by Rooms.

Model Interior Topology

To navigate through the interior of Models, simulator client-devices need to know the connections between the elements composing the interior, such as floors, rooms, doors, or fixtures. In addition, these elements must be identified and attributed for use by computer generated forces (CGF) client-devices. For this reason, the CDB Standard has opted for reuse and adoption of version 2 of the UHRB specification ^[18].

The CDB standard maps the UHRB Object Model to the OpenFlight Scene Graph using the concept of CDB nodes.

The UHRB object model proposes twelve (12) classes. Of these, four (4) are abstract base classes and one (1) is a provision for future expansion of the UHRB specification. The remaining seven (7) concrete classes are mapped to CDB Zone nodes. The UHRB Class Names and their corresponding CDB Zone Names are:

Table 6-9: UHRB Class Names and corresponding CDB Zone Names

UHRB Class Name	CDB Zone Name
UHRB_TEMPLATE	Interior
UHRB_FLOOR_LEVEL_COMPONENT	Floor
UHRB_SURFACE_COMPONENT	Surface
UHRB_ROOM_COMPONENT	Room
UHRB_FIXTURE_COMPONENT	Fixture

UHRB Class Name	CDB Zone Name
UHRB_APERTURE_COMPONENT	Aperture
UHRB_FIXED_PARTITION_COMPONENT	Partition

The above CDB nodes are treated the same way as any other CDB nodes. In particular, Floor, Room, Partition, Aperture, Fixture, and Surface nodes are numbered following the conventions established in section 6.5.5, Model Zone Naming; they also have zone attributes such as the Material Index.

Model Interior Topology Attributes

This section describes the CDB mechanism that expresses the possible connections between compartments and apertures. The definition of a CDB Zone is extended with the addition of one XML tag indicating which other components are connected to this one.

The following table presents the revised syntax of the XML tags defining a CDB Zone. The addition is highlighted in yellow.

Table 6-10: XML Tags for Zone Connections

```
<CDB:Zone name="name" volume="closed|open">
  <Material> _index_ </Material>
  <Temperature> _value_ </Temperature>
  <ConnectTo> _path_ </ConnectTo>
  ...
</CDB:Zone>
```

The `<ConnectTo>` tag may appear zero or more times, allowing for the definition of any number of connections to other components. The other tags (Material and Temperature) retain their current definition. In particular, the use of the `<Material>` tag is encouraged to define the material the components are made of.

The presence of the `<ConnectTo>` tag is restricted to a set of three (3) components: global zone, compartments and apertures. A connection is unidirectional; it goes from the zone that contains the `<ConnectTo>` tag to the zone referenced by the path. The path is either relative or absolute. When a relative path is used, it identifies a sibling of the current zone. Here are some path examples.

Table 6-11: Examples of Absolute and Relative Paths

Example 1:

```
<CDB:Zone name="Interior">  
<ConnectTo> \Interior\Section[1]\Level[1]\Aperture[5] </ConnectTo>  
</CDB:Zone>
```

Example 2:

```
<CDB:Zone name="Aperture[5]">  
<ConnectTo> \Interior\Section[1]\Level[2]\Compartment[3] </ConnectTo>  
</CDB:Zone>
```

Example 3:

```
<CDB:Zone name="Compartment[3]">  
<ConnectTo> Aperture[1] </ConnectTo>  
<ConnectTo> \Interior\Section[1]\Level[1]\Aperture[5] </ConnectTo>  
</CDB:Zone>
```

Example 1 is an absolute path, expressed as a directory name, starting with the topmost zone, the global zone. It tells us that there is one entrance into the model interior through the fifth aperture (Aperture[5]) on the first level (Level[1]) of the first section (Section[1]) of the model interior (\Interior).

Example 2 is also an absolute path. It tells us that the fifth aperture (Aperture[5]) has a single connection to the third compartment (Compartment[3]) of the second level (Level[2]) of the first section (Section[1]) of the model interior (\Interior).

Example 3 illustrates how to use a relative path. It tells us that the third compartment (Compartment[3]) has two exits. The first exit is through the first aperture (Aperture[1]) of the current level. The second exit is through the fifth aperture (Aperture[5]) on the first level (Level[1]) of the first section (Section[1]) of the model interior (\Interior).

Example 1 tells us to use Aperture 5 to enter into the model interior. Example 2 further informs us that Aperture 5 brings us into Compartment 3. Example 3 says that we can exit Compartment 3 through either Aperture 1 or 5.

The global zone (the top level node) node provides the list of apertures representing entrances into the model. If the global zone does not provide at least one aperture to enter the model, then the model interior is unreachable. To exit a compartment, it must connect to at least one aperture; if not, you may be able to enter the compartment, but you will not be able to exit. Finally, an aperture allows entrance into compartments. An aperture without connection is an exit point; in that case, a

compartment must connect to the aperture.

The CDB Standard Schema Package provides the XML schema governing the construction of a valid CDB Zone. The schema includes provision for the <ConnectTo> tag.

Floor Zone

A Floor zone contains one or more Rooms, and all Partitions shared by these Rooms. A Floor is defined by the following XML tags.

Table 6-12: Floor Zone XML Tags

```
<CDB:Zone name = "Floor">  
<Label> floor name </Label>  
</CDB:Zone>
```

The <Label> is optional. It can be used to give the Floor a meaningful name such as “Ground Floor”, “Basement”, “Mezzanine”, or “Penthouse”.

The subgraph representing a Floor has the following structure.



Figure 6-26. Floor Zone Structure

The Footprint of a Floor is the minimum enclosing polygon containing all of the footprints of all of the Rooms on the Floor as well as the footprints of all of the Partitions associated with those Rooms. The Footprint is defined as per section 6.5.6.2, Model Footprint Zones. The Partitions contain the Apertures that connect two Rooms together. These Apertures are later referenced by Rooms.

Room Zone

A Room zone owns all its Surfaces and may contain Fixtures. A Room is defined by the following XML tags.

Table 6-13: Room Zone XML Tags

```

<CDB:Zone name = "Room">

<Label> _room name_ </Label>

<Aperture> _path to aperture 1_ </Aperture>

... other apertures as needed

<Partition> _path to partition 1_ </Partition>

... other partitions as needed

</CDB:Zone>

```

The <Label> is optional. It can be used to better identify the Room by its usual name. Examples are cubicle, toilet, conference room, atrium, office, electrical room, janitor room, etc.

The <Aperture> is optional but is likely to appear at least once, unless the Room is permanently closed and cannot be accessed. It points to one Aperture that connects this Room with another Room on this Floor or on another Floor. Two Rooms on the same Floor are connected through an Aperture in a Partition on the current Floor. Two Rooms on two different Floors are connected through an external Aperture in a Partition from the Interior zone. The path to an Aperture is built as specified in section 6.5.5, Model Zone Naming.

The <Partition> is also optional, but again, is likely to appear several times since a typical Room has a floor, a set of walls, and a ceiling.

The subgraph representing a Room has the following structure.



Figure 6-27. Room Zone Structure

The footprint is the smallest polygon containing all of the bottom surfaces when projected onto the XY plane ^[19]. There can be zero or more Fixtures in a Room. The Surfaces making up the volume of

the Room are separated in three (3) groups (Bottom, Side, and Top) as defined by the UHRB specification.

Fixture Zone

A Fixture zone is defined in the same manner as a Room; it is made of a number of Surfaces defining a closed volume. The Fixture is defined by the following XML tags.

Table 6-14: Fixture Zone XML Tags

```
<CDB:Zone name = "Fixture">  
<Label> _fixture name_ </Label>  
<Moveable> _true/false_ </Moveable>  
</CDB:Zone>
```

The `<Label>` is optional. It allows the modeler to describe what this fixture represents.

The `<Moveable>` flag is optional. It indicates whether or not the Fixture can move or if it is fixed. By default, the fixture does not move; if it does, the flag is set to true. A piece of furnitures is an example of moveable fixture while a column is an example of a fixed one.

The subgraph representing a Fixture is similar to that of a Room, except for the need to differentiate between the kinds of Surfaces. Its structure is presented here.



Figure 6-28. Fixture Zone Structure

The Footprint is the smallest polygon containing all of the Surfaces when projected onto the XY plane. The Surfaces form a closed volume, meaning there is no hole in the Fixture.

Alternately, to permit reuse of common fixtures stored in the GTModel Library, the Fixture may reference an existing model through the use of an XRef node. In that case, the following subgraph is to be used.



Figure 6-29. Fixture Zone Structure (XRef Subgraph)

Partition Zone

A Partition zone has Apertures, makes reference to all Surfaces composing it, and refers to its adjacent Rooms. The Partition is defined by the following XML tags.

Table 6-15: Partition Zone XML Tags

```

<CDB:Zone name = "Partition">

<Label> _partition name_ </Label>

<Room> path to adjacent room 1 </Room>

<Room> path to adjacent room 2 </Room>

<Surface> _path to surface 1_ </Surface>

... other surfaces as needed

</CDB:Zone>

```

The `<Label>` is optional. It allows the modeler to better identify the type of Partition: wall, floor, ceiling, etc.

The <Room> tag is mandatory and is used to identify the two Rooms adjacent to the Partition^[20]. For this reason, there must be exactly two <Room> tags. The path to an adjacent Room is as specified in section 6.5.5, Model Zone Naming. Note that UHRB defines the concept of an “outside” room when the partition defines a building outside wall. In CDB, the path of this outside room is \Shell\Exterior as illustrated in 6.5.6.4.1, Model Pseudo-Interior.

The <Surface> tag appears as many times as necessary to refer to all Surfaces making up this Partition. A path similar to the one used to refer to a Room is used to refer to a Surface. Note that a Partition does not refer to the Surfaces that belong to its Aperture; that will be taken care of by the Apertures themselves.

The subgraph representing a Partition has the following structure.



Figure 6-30. Partition Zone Structure

The Footprint is the smallest polygon containing all of the referenced Surfaces when projected onto the XY plane. A Partition can have zero or more Apertures in it.

Aperture Zone

An Aperture zone provides a mean by which one can enter or exit a Room. The Aperture zone is defined by the following XML tags.

Table 6-16: Aperture Zone XML Tags

```

<CDB:Zone name = "Aperture">

<Label> _aperture name_ </Label>

<Is_Open> _true/false_ </Is_Open>

<Is_Fixed> _true/false_ </Is_Fixed>

<Damage_Level> _percentage of damage_ </Damage_Level>

<Room> _path to room 1_ </Room>

<Room> _path to room 2_ </Room>

<Surface> _path to surface 1_ </Surface>

... other surfaces as needed

</CDB:Zone>

```

The <Label> is optional. It allows the modeler to better identify the type of Aperture: door, window, trap, etc.

The <Is_Open> and <Is_Fixed> flags are both optional; they are considered false when not provided.

The <Damage_Level> tag is also optional and provides a mean to indicate the level of damage of the Aperture. The value is expressed as a percentage using an integer in the range 0 (no damage) to 100 (destroyed).

The <Room> tag appears exactly two times and points to the two Rooms connected by this Aperture. Sometimes one of these two rooms may be an “outside” room - a concept defined in UHRB. In CDB, the path of this outside room is \Shell\Exterior as illustrated in 6.5.6.4.1, Model Pseudo-Interior.

The <Surface> tag appears as many times as necessary to refer to all Surfaces making up this Aperture.

The subgraph representing an Aperture has the following structure.



Figure 6-31. Aperture Structure

The Footprint is the smallest polygon containing all of the referenced surfaces when projected onto the XY plane.

Surface Zone

A Surface zone contains useful geometry. That's all it does. The Surface zone is a plain CDB zone. Its subgraph is presented here.



Figure 6-32. Surface Zone Structure

A Surface is composed of one or more OpenFlight Object nodes holding the geometry defining the surface: face or mesh records.

6.6. Model Points

A model point is similar to a model zone; it identifies a location on the model that is of interest to at least one simulation client device. A point defines a local coordinate system on the model. Hence, a point has a position and an orientation.

In some respect, a point and a zone are similar and can be used interchangeably. A zone is used when the component of interest is physically modeled and has a graphical representation. When the zone is not modeled but still represents a component of interest, a point is used to indicate its presence.

Again, the OpenFlight Group Node mechanism provides a convenient means of implementing the concept of a point because a transformation can be added to the node.

Requirements Class – Model Points	
/req/openflight/model-points	
Target type	Operations
Dependency	Openflight Specification
Requirement 23	/req/openflight/model-point-damage=states
Requirement 24	/req/openflight/model-dis-origin
Requirement 25	/req/openflight/model-viewpoint

6.6.1. Definition

Table 6-17 below presents the syntax of the XML tags stored in the node's comment record.

Table 6-17: XML Tags for Points

```
<CDB:Point name = "name">  
... point attributes  
</CDB:Point>
```

The point name is mandatory while the point attributes are optional. In general, a point can have the same name as a zone. Table 6-18 lists the OpenFlight records required to represent a point.

Table 6-18: OpenFlight Records for a Point

GROUP
MATRIX (mandatory)
COMMENT (mandatory)

A model point is used in several occasions such as defining the attach point where another Model can anchor itself.

6.6.2. Usages

Model DIS Origin

A Model that is intended as a DIS entity requires a point that defines the origin of the entity's coordinate system. This point is the center of the entity's bounding volume excluding its articulated and attached parts [21]. On a DIS network, the location of an entity is expressed relative to this point.

Requirement 23 req/openflight/model-point-damage=states

There *SHALL* be a single definition of this point for all damage states and all levels of details for a given model. If the DIS origin is not defined, it *SHALL* default to the model origin.

The following XML tag identifies the point.

Table 6-19: XML Tags for the DIS Origin

```

<CDB:Point

name = "DIS_Origin"

/>

```

Requirement 24

[req/openflight/model-dis-origin](#)

The CDB Point representing the DIS Origin *SHALL* be positioned and oriented according the definition provided by the DIS Standard. This definition says that the DIS Origin is at the center of the bounding box of the entity, without articulated and attached parts. The standard also says what the orientation must be. The X-axis points forward, the Y-axis points to the right, and the Z-axis points down. All axes are aligned with the bounding box defined above.

The intent of the DIS Standard is to have its axis system aligned with the body of the entity. When it comes to air platform, the body is associated with the fuselage of the entity. To illustrate the difference in orientation between the DIS entity's bounding box and the CDB Model's bounding box, consider the Chinook helicopter shown below.



Figure 6-33. Orientation of the Chinook Helicopter

The fuselage of this helicopter has a pitch angle of approximately 1.6 degrees when resting on its wheels. Below is a snapshot of its fuselage, without rotors and landing wheels.

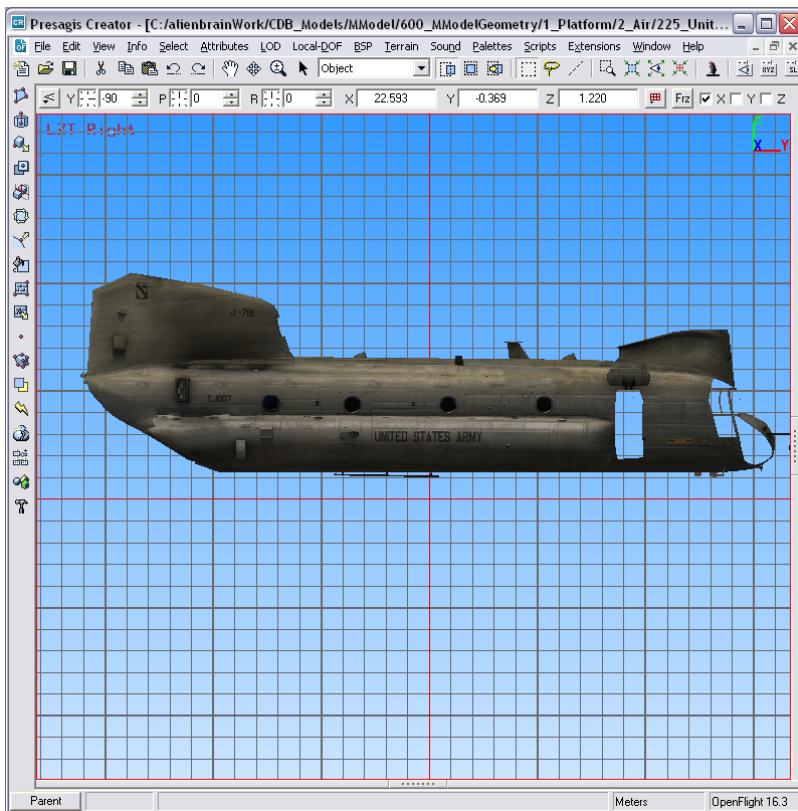


Figure 6-34. The Body of the Chinook Helicopter

From the snapshots above, it is clear that the orientation of the DIS origin must be such that its XY plane makes an angle of 1.6 degrees with respect to the XY plane of the CDB axis system.

Here is a recommended way of defining the DIS Origin:

1. Create the Group Node and tag it as a CDB Point whose name is DIS_Origin.
2. Make the CDB Point a child of the zone that best represents the entity's bounding volume without any articulated and attached parts.
3. Ensure the zone is properly oriented with respect to the CDB axis system.
4. Add a translation to position the origin at the center of the above bounding volume.
5. Add a rotation to align the X-axis to the front of this bounding volume.
6. Add another rotation to align the Z-axis with the bottom of the bounding volume.
7. By doing so, the Y-axis should already point correctly to the right side of the box.

Example

The snapshot below shows the proper location and orientation of the DIS origin on the Chinook. The DIS origin is represented by a set of 3 orthogonal Blue-Red-Green arrows. The blue arrow indicates the X axis; the green arrow points down and represents the Z axis.



Figure 6-35. The DIS Origin of the Chinook Helicopter

If you watch carefully, you will notice that the DIS axis system is aligned with the fuselage and makes an angle with the CDB XY plane.

Model Viewpoint

To generate the correct view of the outside world from a model's viewpoint, a client device needs an indication of where is the viewpoint located with respect to the model's origin. The viewpoint corresponds to the pilot's seat in an aircraft, the driver's seat in a ground vehicle, the navigation post on a ship bridge, the periscope on a submarine, or the eyes of a soldier.

Requirement 25

[req/openflight/model-viewpoint](#)

The viewpoint's local coordinate system *SHALL* be oriented such that the Y-axis indicates the viewing direction and the Z-axis points up.

The viewpoint has optional attributes to define the field of view available from this position. The field of view is defined by a frustum aligned along the local Y-axis. The horizontal field of view lies in the local XY plane while the vertical field of view is in the YZ plane.

Table 6-20: XML Tags for a Viewpoint

```

<CDB:Point name="Viewpoint">

<FOV>

<Horizontal>min max</Horizontal>

<Vertical>min max</Vertical>

</FOV>

</CDB:Point>

```

All values are expressed in degrees using decimal numbers. The default values are $\pm 30.0^\circ$ in both directions for a total of 60.0° of horizontal and vertical fields of view.

Model Attach Point

A Model can be attached to another Model by mean of an attach point.

An attach point defines the position to which other (subordinate) models can attach themselves. For instance, a fighter has a number of attach points defined to receive missiles or external fuel tanks.

Table 6-21: XML Tags for Attach Point

```

<CDB:Point

name = "Attach_Point"

/>

```

The orientation of the attach point is used to indicate how the two models connect together. A connection occurs by superimposing the coordinate system of the subordinate model with the coordinate of the attach point.

Model Anchor Point

The anchor point defines the location where a subordinate Model attaches to a parent Model. The anchor point is the counterpart to the attach point. Both can be seen as the male/female part of a connector and its receptacle.

Table 6-22: XML Tags for Anchor Point

```

<CDB:Point

name = "Anchor_Point"

/>

```

The orientation of the anchor point is used to indicate how the subordinate model connects to the parent model. A connection occurs by superimposing the anchor point (of the subordinate model) with the attach point (of the parent model).

The default anchor point of a subordinate model is its origin.

Model Center of Mass

The Center of Mass (CM) of a Model is a specific point where, for many purposes, the Model behaves as if its mass was concentrated there.

Table 6-23: XML Tags for Center of Mass

```
<CDB:Point  
name = "Center_Of_Mass"  
/>
```

6.7. Model Conforming

Historically, the integration of models onto the terrain has been performed during the database compilation process. These offline approaches varied considerably from vendor to vendor because there were no standardized approaches related to terrain meshing structures, varying visual priority and hidden-surface removal mechanisms, runtime LOD mechanisms, number of LODs, etc.

This section describes a series of model conformal modes that instruct client-devices on how they should conform models to the underlying terrain.

All of the conformal modes rely on the conforming of the model origin and/or model vertices onto the terrain mesh directly beneath the model. Note that the Z-component of the model's vertices is with respect to model's XY plane (as shown in Figure 6-36).



Figure 6-36. Conforming Vertices to Terrain

Note that by definition, all portions of the model below its XY plane represent some form of under ground basement. The conforming of models on steep or rough terrain may yield unusual results because portions of the basement may be visible. This may require the modelers to level the terrain in the immediate vicinity or adjust the model.

A modeler can specify a model's conformal mode by adding the following XML tags to the zone representing the model.

```
<CDB:Zone>
<Conformal mode="..."/>
</CDB:Zone>
```

The conformal modes are listed in [Table 1](#) below.

Table 6-24: Conformal Modes

Conformal Mode
Absolute
Point

Conformal Mode

Vertex

Line

Plane

Surface

6.7.1. Non Conformal (Absolute) Mode

When attributed as a Non Conformal model, none of the model vertices are conformed to the underlying terrain. Instead the model's Z-values are used as-is, as elevation values. As a result the model is absolutely positioned and behaves independently of the terrain. The shape and orientation of the model is preserved. This conformal mode is typically used for the modeled representation of point-features. Typical use-cases include buildings, trees, and poles.

6.7.2. Point Conformal Mode

The Point Conformal mode conforms a single point of the model (its origin) onto the underlying terrain. All of the other model vertices are translated along the Z-axis; as a result, the shape of the model is preserved by this conformal operation. In effect, the Point Conformal mode dynamically positions a model on the underlying terrain so as to preserve the model's relative altitude over the terrain. Point-conforming is the default conformal mode for the modeled representation of point-features. Typical use-cases include buildings, trees, and poles.



Figure 6-37. Origon Conformal Mode

6.7.3. Vertex Conformal Mode

The Vertex Conformal mode conforms each of the vertices of a model on the underlying terrain. The shape of the model is **not** preserved by this conformal operation. The model's XY plane defines a reference plane used by client-devices to adjust the elevation of each of the model's vertices. This conformal mode is used for 3D models that represent typically long 3D lineal features or large 3D areal features that need to follow the terrain profile. Typical uses include fences, walls, trenches, and forest canopies.



Figure 6-38. Vertex Conformal Mode Example

6.7.4. Line Conformal Mode

The Line Conformal mode conforms each of the two reference vertices of the “linear” model on the underlying terrain. All of the other model vertices are sheared along this axis; as a result, the shape of the model is **not** preserved by this conformal operation. The model's XY plane defines a reference plane used by client-devices to adjust the elevation of the two reference vertices. This conformal mode is used for models that represent lineal features such as powerlines and monorails.



Figure 6-39. Line Conformal Mode

The line that is used to specify the conforming is defined by a Face node with the following XML tags:

```
<CDB:Face>
<Conformal_Line/>
</CDB:Face>
```

This Face node defines a single line with two vertices, the first one, V_s , being the start and the second, V_e , the end of the line.

6.7.5. Plane Conformal Mode

The Plane Conformal mode conforms each of the three reference vertices of the “planar” model on the underlying terrain. The resulting three vertices define a model transformation matrix that can then be applied to the vertices of the model. As a result, the shape of the model **is** preserved by this conformal operation, but the model undergoes a change in pitch and roll angles. Given this property, there are relatively few cases where this conformal mode can be used [22]. However, as shown in Figure 6-41, this conformal mode is required when conforming the curved segments of 3D (raised profiled) modeled road features.

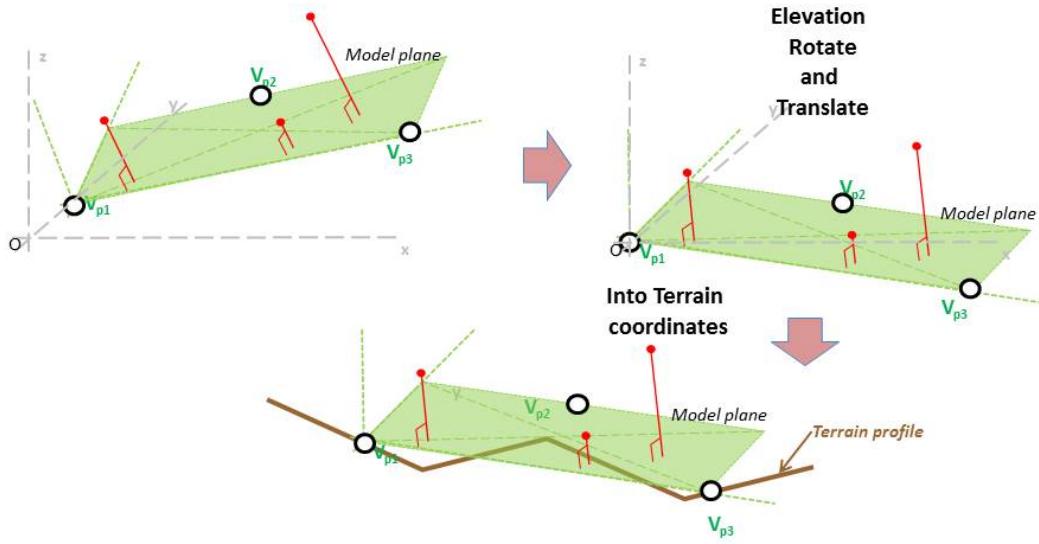


Figure 6-40. Plane Conformal Mode

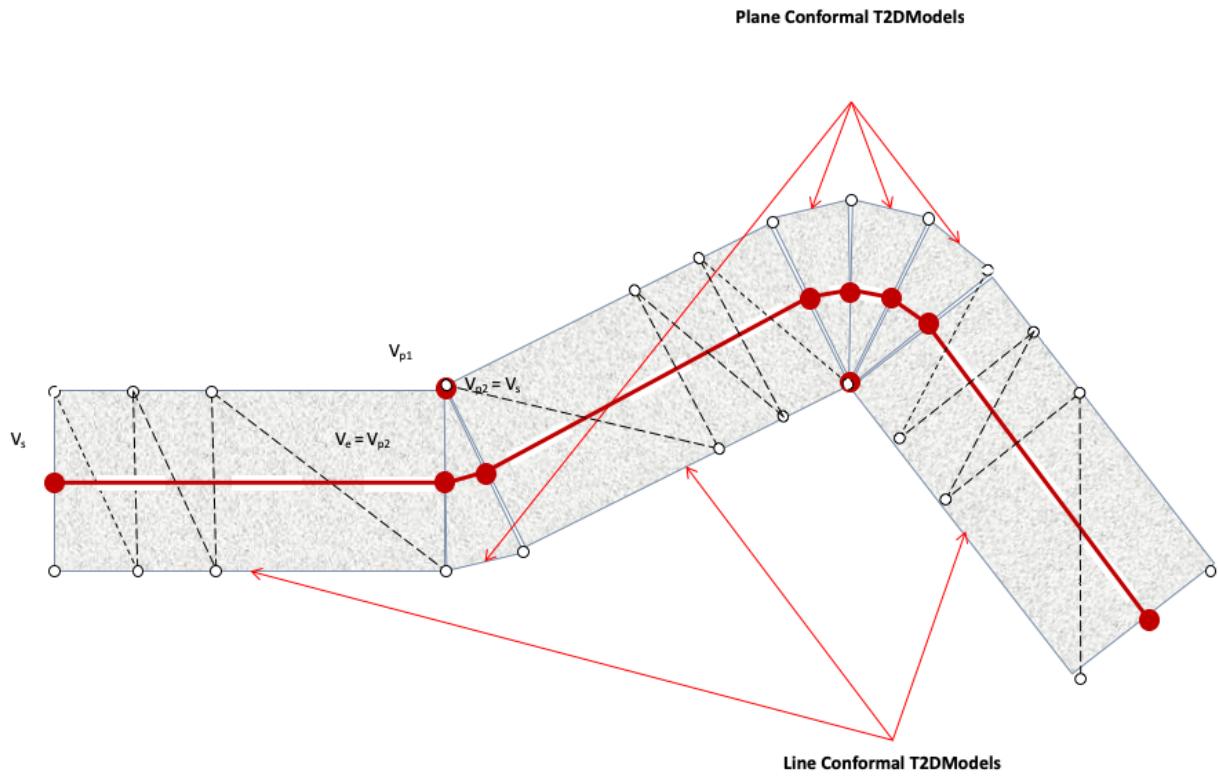


Figure 6-41. Application of Line and Plane Conformal Modes on 3d Roads

The plane that is used to specify the conforming is defined by a Face node with the following XML tags:

<CDB:Face>

<Conformal_Face/>

</CDB:Face>

The Face node has exactly 3 vertices defining the plane used for the conforming. The only restriction on these 3 vertices is that they must not be collinear.

6.7.6. Surface Conformal Mode

This conformal mode is used for models whose points, edges and surfaces must all conform exactly to the underlying terrain. The Surface Conformal mode requires that the model's edges and surfaces be clipped to the underlying terrain. The original vertices and the added vertices resulting from the clipping operation are conformed to the underlying terrain. As a result, the shape of the model is **not** preserved by this conformal operation. This conformal mode is primarily used for the modeled representation of 2D surface-feature such as paint markings and other terrain overlays. In addition, it can be used for 3D models that represent typically long or large 3D lineal and 3D areal features that need to **perfectly** follow the terrain profile. Note that in most cases, the vertex conformal mode provides an adequate solution for 3D models and is more economical to use than the surface conformal mode.



Figure 6-42. Surface Conformal Mode

6.8. Model Levels-of-Detail

Requirements Class – Model Levels of Detail

/req/openflight/model-lod

Target type	Operations
Dependency	Openflight Specification
Requirement 26	/req/openflight/significant-sizes

A levels-of-detail model structure is essential when the intent is to use a model in a real-time application such as flight simulation. The level-of-detail mechanism provides client-devices with the essential structure for deterministic operation. Deterministic operation can be achieved only if a client-device can:

- control the paging bandwidth from the CDB main storage device
- control client-device processing load
- control client-device memory footprint
- control run-time publishing processing load and
- control run-time publishing memory footprint

For this reason, it is recommended to create LODs, especially for complex Models, and for models that are used extensively, in great density in the CDB data store. This is most critical for geospecific cultural models (especially in densely modeled geospecific areas of the synthetic environment) since they can consume a significant portion of the paging bandwidth and memory footprint of the client-devices. As a corollary, simple Models should not be made more complex by adding unnecessary level of details. The CDB standard provides rules for determining model complexity, and selecting the appropriate LOD, as defined in Chapter 3 of Volume 1: OGC CDB Core Standard:Model and Physical Database Structure.

OpenFlight LOD nodes now support two methods of specifying the criteria to determine if a level of detail is active, that is if the user application should traverse the node and its children. The first method, the classic one, is to specify the switch in and switch out distances in real world units. Using this method, a level of detail is active when the distance from the viewpoint to the center of the LOD is within the switch-in and switch-out distances. The second method uses the Significant Size associated with the LOD node to determine when to activate the node.

Requirement 26

req/openflight/significant-size

The CDB Standard *SHALL*:

- Use the Significant Size associated with the LOD node to determine when to activate the node
- The Significant Size of LOD node 2 *SHALL* be less than the Significant Size of LOD node 1.
- Features whose Significant Size is larger than 110 km (the dimension of a geocell) *SHALL* be clipped (if a continuous surface) or ungrouped (if multiple discontinuous surfaces) until the maximum size criteria is met.
- LOD nodes *SHALL* be sorted in decreasing order of their Significant Size attribute.
- Sibling LOD nodes *SHALL* be mutually exclusive.

There are several problems associated with the classic, range-based method. In a visual system for instance, the switching distance should be based on both range and the system resolution of the entire visual system; a database designed to rely solely on a range-based switching criteria is not truly portable, especially if the intent is to use it on systems with wildly different visual resolution. Furthermore, the blending or morphing of models solely based on range criteria can lead to undesirable effects. When the viewpoint moves quickly, the distance over which the model is LOD-transitioning should be large enough to avoid the “popping-in” of the higher LOD version. On the other hand, if the viewpoint is moving very slowly, the distance over which the model is LOD-transitioning should be reduced to avoid the “LOD-ghosting” of the higher LOD version. These two constraints make implicit assumptions on the model’s speed. In applications where the aircraft’s flight regime varies considerably (V22 for example), it is impossible to find a single set of LOD start and end points that simultaneously cater to all flight modes (hover versus cruise). Here again, a database design that directly encodes the start and end points of a model’s LOD transition is not truly portable, because it makes implicit assumptions on the speed it will be used for. Thus, in a tactical fighter application, the start and end points of a model’s LOD transition need to be widely spaced apart to prevent a popping effect at the onset of the LOD transition. Conversely, in a tank application, the start and end points of a model’s LOD transition need to be much more closely spaced to prevent a ghosting effect as the higher LOD model is blended-in. If the client device wants to implement some form of transition between LODs, the criteria should be based on a user-defined duration. Transitions between LODs can involve fading in the next LOD while fading out the current one. That fading operation should not last forever. It should be accomplished in a relative short period of time. The second method to transition from one LOD to the next is to use morphing. In the case of morphing, the transition period is less critical because the client-device (typically an Image Generator) does not blend-in two models together.

The consequences of such implicit assumptions result in a database that is highly client-device, and

application-specific.

For all these reasons, the CDB standard has selected the second method to control the LOD mechanism.

Two methods exist to implement LODs, exchange or addition. The two methods can be used simultaneously and are not mutually exclusive.

In the first method, details are progressively added to the model, as the viewpoint gets closer. With the second method, different representations of the same model are substituted for one another based on the viewing distance. Figure 6-43: Exchange and Additive LOD Nodes, illustrates the general organization of Models with both types of LOD nodes.

6.8.1. Exchange LODs

In the exchange LOD method, different representations of the same model are substituted for one another based on the model's Significant Size. It is up to the client-devices to derive appropriate LOD transition viewing distance from the model's Significant Size and the client-device's field-of-view and resolution parameters.

6.8.2. Additive LODs

Additive LOD is just a special case of the more general Exchange LOD paradigm. When a LOD node has no sibling LOD, it becomes an Additive LOD node. That does not change the fact that at most one LOD node gets selected when its Significant Size justifies it.



Figure 6-43. Exchange and Additive LOD Nodes

The LOD nodes in light brown represent Exchange LODs and are mutually exclusive. The two dark brown shaded LOD nodes are considered Additive LODs because they do not have another sibling node of type LOD.

Note that to make sense, the Significant Size of LOD node 2 must be less than the Significant Size of LOD node 1 (Requirement 26). The same is true for the Significant Size of LOD node 3 with respect to LOD node 2. Also, LOD node 3 has an additional constraint, its Significant Size must be greater than the one assigned to LOD node 4. If these constraints are not followed, LOD node 4 will be selected before LOD nodes 2 or 3 have a chance of being displayed.

6.8.3. Significant Size

The concept of a Significant Size is a recent improvement of the OpenFlight Specification. When a finer model LOD is created, the modeler typically adds additional geometric detail, additional features (such as markings), or refines the shape of curved surfaces (such as engines, wheels), etc. When assigning a Significant Size to a model LOD, the modeler needs to answer the following question: When I created a new model LOD, I did so to create additional detail in my model. What is the largest dimensional change in geometry for this new model LOD? In other words, what is the largest dimensional difference of a surface between this LOD and the next coarser LOD? In effect, the value of Significant Size corresponds to the “modeling difference” between the LOD and the next coarser LOD. At runtime, a client-device converts this modeling difference value from its real-world dimensional value into a viewing error value (typically measured in pixels or degrees). The client-device can then activate the appropriate model LOD because it knows that the modeler’s intent in creating the LOD was to show features, eliminate all modeling discrepancies whose dimension equaled that of the Significant Size dimension associated with that model’s LOD. This contribution of the LOD to the scene is based on the LOD’s Significant Size as well as other parameters (such as system resolution) relevant to the simulation model used by the client device.

Version 16.0 of the OpenFlight Specification introduces the concept of Visual Significance that is different from the concept of Significant Size. The concept of Visual Significance translates in two fields called Significance and they are found in the Group Record and Object Records [23]. Here is the definition of this field as found in reference [11]:

“Significance can be used to assist real-time culling and load balancing mechanisms, by defining the visual significance of this group with respect to other groups in the database. Normally the value of this attribute is zero”.

The CDB standard mandates a value of zero for Visual Significance; the value zero indicates the object or the group has no particular significance and is not more or less important than any other objects or groups. Any other values, whether negative or positive, are reserved for future use by this Specification.

Definition of Significant Size

The Significant Size is defined as the “size” of the model, expressed in meters. By extension, it applies equally well to a submodel represented by an Additive LOD. In the case of an Exchange LOD, the Significant Size is the difference between two representations of the model or submodel.

Estimating the Size of the Model

Many models have shapes that resemble a cube (with roughly equal length, width, and height), and thus their significant size can be simply estimated by the length of the diagonal of their bounding box. As the shape of a model departs from that of a simple cube, either with respect to aspect ratio,

or with respect to the amount of negative space within its bounding box, the model's significant size should be decreased proportional to the amount of departure.

How to use the Significant Size

The Significant Size is used to distribute models into appropriate CDB LODs. Once a value is assigned to the coarsest LOD of a model, subsequent LODs of the same model can be distributed to subsequent CDB LODs. Use Table 3-1 and the model Significant Size to identify the CDB LOD it belongs to.

For instance, if the size of a building is estimated to 75 meters, then its coarsest LOD will be stored in CDB LOD 0, according to Table 3-1. On the other hand, a 2-meter park bench will appear in CDB LOD 5.

6.8.4. LOD Limits

The number of vertices per LOD is limited to ensure a smooth progression between all representations of the same model. [Table 6-25](#) below gives the maximum number of vertices allowed for each Model-LOD.

Table 6-25: Maximum Number of Vertices per Model-LOD

Model LOD	Maximum Number of Vertices	CDB LOD
0	128	CDB LOD _c + 0
1	512	CDB LOD _c + 1
2	2048	CDB LOD _c + 2
3	8192	CDB LOD _c + 3
4	32768	CDB LOD _c + 4
5	131072	CDB LOD _c + 5
6	524288	CDB LOD _c + 6
7	2097152	CDB LOD _c + 7

The table above shows that a model is allowed up to 8 levels of details, numbered from 0 to 7. Model-LOD 0 is the coarsest level of detail of the model and may count up to 128 vertices. As the complexity of subsequent Model-LODs augments, a higher vertex count is permitted. The CDB LOD that is associated with a Model-LOD is expressed relative to the CDB LOD assigned to its coarsest representation, designated by CDB LODs.

How to Assign CDB LODs

To illustrate the use of [Table 6-25](#), take a 3D model representing a building with three representations:

- Coarsest LOD:
 - 8 vertices

- Significant Size estimated to 25 meters
- Medium LOD:
 - 2244 vertices
 - Significant Size estimated to 4 meters
- Finest LOD:
 - 10320 vertices
 - Significant Size estimated to 10 cm

The CDB LODs are first established by looking up the Significant Sizes of the three representations in Table 3-1:

- Coarsest LOD:
 - 25 m is CDB LOD 2
 - This is CDB LODc
- Medium LOD:
 - 4 m is CDB LOD 4
- Finest LOD:
 - 10 cm is CDB LOD 10

We then use the vertex count to identify the Model-LOD in [Table 6-25](#):

- Coarsest LOD:
 - 28 vertices is Model-LOD 0
- Medium LOD:
 - 2244 vertices is Model LOD 3
 - Should be assigned to CDB LODc $(2) + 3 = 5$
- Finest LOD:
 - 10320 vertices is Model LOD 4
 - Should be assigned to CDB LODc $(2) + 4 = 6$

Since all representations of the model must meet both the constraints associated with the Significant Size and the Vertex Count, the final CDB LODs are the maximum ones identified above.

- Coarsest LOD:
 - CDB LOD 2
- Medium LOD:
 - CDB LOD 5
- Finest LOD:
 - CDB LOD 10

6.8.5. LOD Generation Guidelines

The following guidelines should help modelers produce efficient CDB models for use in real-time environments. There are two ways of proceeding; one way is to create the finest representation of the model and then simplify it until the coarsest representation is obtained. The other way consists in creating the coarsest representation first and then refining it until the desired representation is obtained. In general:

- The coarsest Model-LOD is the simplest possible geometric representation of the model using at most 128 vertices.
- A coarser Model-LOD is created by removing details from a finer Model-LOD.
- Alternately, a finer Model-LOD is created by adding details to a coarser Model-LOD.
- In both cases, the size of the details that are removed or added to a Model-LOD should be consistent with its Significant Size.
- Model-LOD 0 is mandatory; the others are optional and exist only if Model-LOD 0 isn't sufficient to represent the model with a proper level of detail.
- Multiple Model-LODs do not need to be consecutive.

6.9. Model Switch Nodes

A Switch Node allows the selection of zero or more children by invoking a selector mask. Any combination of children can be selected per masks and the number of definable masks is unlimited. The CDB standard makes use of OpenFlight Switch Nodes to control the state of Model Components (zones and points).

Requirements Class – Model Switch Nodes	
/req/openflight/model-switch-nodes	
Target type	Operations
Dependency	Openflight Specification
Requirement 27	/req/openflight/switch-mask
Requirement 28	req/openflight/switch-mask-name

6.9.1. Definition

XML tags in the comment record are added to the switch's primary record to identify it as a CDB Switch.

Table 6-25b: XML Tags to Create a CDB Switch

```

<CDB:Switch name = "switch name">
    ...
    ... switch attributes
</CDB:Switch>

```

Requirement 27	req/openflight/switch-mask+
-----------------------	-----------------------------

The switch *SHALL* contain one mask per state. Note that the first mask, mask index 0, is the default mask. This means that the value of the Current Mask field in the Switch record *SHALL* be 0.

As an example, if the switch has 3 children, each representing a separate state of the parent zone, then the switch needs 3 masks, each selecting one child.

Requirement 28	req/openflight/switch-mask-name
-----------------------	---------------------------------

In addition to defining a mask for each switch state, each mask *SHALL* be named. The name of the mask *SHALL* be representative of the state selected by that mask.

The actual name is at the discretion of the modeler.

The corresponding OpenFlight records are as follow:

Table 6-26: OpenFlight Records to Create a CDB Switch

SWITCH
COMMENT (mandatory)
INDEXED STRING

6.9.2. Usage

Articulations

Switch nodes provide an alternative to DOF nodes when an articulated part is implemented for only a few positions. An example of this use of switches is the control of undercarriage or control surfaces on aircraft. Suppose the modeler wants to represent the flaps in two distinct positions: flaps up and flaps down. A switch is the simplest way to implement these two flaps positions. In this example, the switch name could be “Flap Control” and the two mask names could be “Flap Up” and “Flap Down”.

Suppose the modeler wants to provide two positions for the door on a hangar: open and close. In addition, when the door is open, the modeler provides a representation for the interior of the

hangar, which is not the case when the door is closed. Again, the use of a switch is appropriate to provide the control over the door position. A proper name for the switch would be “Door Position” and the appropriate names for the two masks would be “Door Closed” and “Door Open”.

Damage States

Switch nodes can be used to select one of many modeled representations of damages. A zone has at least a normal (usually undamaged) state. When other states exist, an OpenFlight switch node is used to select which state is active. A single damage state can be active at any time.

Requirements Class – Damage Status	
/req/openflight/damage-status	
Target type	Operations
Dependency	Openflight Specification
Requirement 29	/req/openflight/damage-transition
Requirement 30	req/openflight/damage-order

Figure 6-45: General Damage State Tree Structure shows the general organization of a zone with several states.

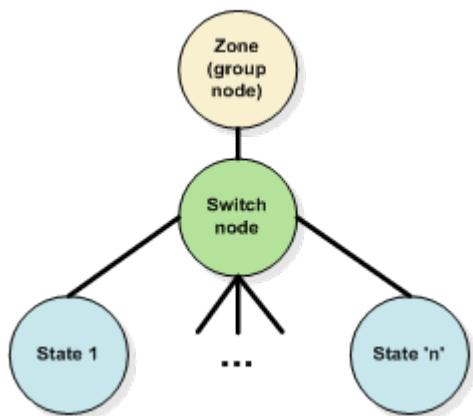


Figure 6-45. General Damage State Tree Structure

Each damage state represents the zone with a certain level of damage. This level of damage is expressed as a percentage from 0 to 100%. A level of damage of 0 % means the zone is not damaged at all. At the opposite end, a percentage of damage of 100 % indicates the zone is completely destroyed.

To identify a damage state switch, use the following XML tags in the switch comment record.

Table 6-27: XML Tags for Damage State Switch

```

<CDB:Switch name = "Damage_State">

<Damage_Level>...</Damage_Level>

</CDB:Switch>

```

The XML element <Damage_Level> is a list of percentages representing the transitions between child nodes of the switch. The list counts ‘n-1’ entries where ‘n’ is the number of states.

Requirement 29

req/openflight/damage-transition

The percentages representing the transitions *SHALL* be limited to the range [0, 99]. The value 100 is not allowed because the level of damage *SHALL* exceed the transition value in order to select the correct state.

To illustrate the concept of level of damage, assume a damage state switch has 3 child nodes representing the zone in normal, damaged, and destroyed states. Also, assume that the modeler’s intent is to switch to the damaged state when the level of damage exceeds 25 %, and to switch to the destroyed state when the level of damage exceeds 75 %. Here, the XML tag associated with the switch should look like this.

Table 6-28: Example of a Damage State Switch with Two Transitions

```

<CDB:Switch name = "Damage_State">

<Damage_Level>25 75</Damage_Level>

</CDB:Switch>

```

Requirement 30

req/openflight/damage-order

The ordering of damage states *SHALL* be from left (normal state) to right (destroyed state). All intermediate states must represent increasingly damaged states from a slightly damaged state to an almost destroyed state. The number of states is left to the discretion of the modeler.



Figure 6-46. Damage States Ordering

While the number of damage states is left to the discretion of the modeler, some choices are better than others. Since a Model is meant to be used in a simulator and since many simulators are DIS-compliant, it is suggested to create the same number of CDB damage states as there are DIS damage states for the corresponding entity.

For instance, if the Model represents a DIS land platform such as the M1A2 tank, the modeler could create four damage states to match the four corresponding DIS damage states labeled No Damage, Slight Damage, Moderate Damage and Destroyed.

The DIS and the HLA standards are relatively vague regarding the definition of damage states. In the case of the DIS standard, the damage state is a field that belongs to a structure called the Entity Appearance. The field has only 2 bits and, accordingly, accommodates four different values. For HLA, version 2 of the RPR-FOM defines the damaged appearance as a 32-bit enumeration for which only 4 values have been defined so far – the same values as the one defined by DIS, that is No Damage, Slight Damage, Moderate Damage and Destroyed.

For both DIS and HLA, it is obvious that the damage state is meant to be a visual damage state.

The question to answer is the following: “What should the universally accepted visual appearance be for a slightly (or moderately) damaged state?”

In the DIS world, a platform is often qualified in terms of Mobility and Fire Power ^[24]. Using these two criteria, it is possible to define the following guidelines.

- A slightly damaged model should represent a platform with limited mobility. However, its firepower is intact and it should be apparent that the entity is still capable of firing its weapons.
- A moderately damaged model should represent a platform for which both mobility and firepower are reduced without being completely out of service.

As a corollary, here are the definitions of normal and destroyed states.

- An undamaged model should represent a platform for which both mobility and fire power are completely operational.

A destroyed model should represent a platform for which both mobility and firepower are completely out of service.

Temporal Anti-aliasing

Temporal anti-aliasing may be achieved with the use of special textures. These textures are often required to aid IG client-devices to eliminate strobing effects on model rotating objects such as helicopter rotors, aircraft propellers, or vehicle wheels.

Figure 6-47: Example of a Texture Representing a Rotor, is an example of a semi-transparent texture used to simulate a rotating helicopter rotor.



Figure 6-47. Example of a Texture Representing a Rotor

Motion blur textures are general base textures with a Texture Kind of S001. The Texture Index (Tnn) is used to sequentially number several motion blur textures representing the same object.

The use of motion blur textures can be combined with DOF and Switch nodes to produce efficient switching between several versions of a single rotating part.

The following subtree illustrates how four versions of the above rotor could be modeled using one solid version and three blurred versions.



Figure 6-48. Multiple Versions of Rotating Parts

In this example, three textures are used to represent an increasingly blurred rotor.

In order to detect the presence of the above construct, the following XML comment must be added to the switch node.

Table 6-29: XML Tags for Motion Blur Switch

```
<CDB:Switch name = "Motion_Blur">  
<Blurriness>...</Blurriness>  
</CDB:Switch>
```

The children of the switch node could be any OpenFlight nodes. Most likely, the nodes that contain the geometry will be OpenFlight Object nodes.

When modeling solid and blurred objects in this manner, the CDB Standard requires that the leftmost child node contains the solid version of the object while the sibling nodes to the right contain increasingly blurred version of the same object.

The XML element <Blurriness> is a list of percentages representing the transitions between child nodes of the switch. The list counts ‘n-1’ entries where ‘n’ is the number of child nodes.

Requirement 31

req/openflight/blurring-transition

The percentages representing the transitions *SHALL* be limited to the range [0, 99]. The value 100 is not allowed because the level of blurriness *SHALL* exceed the transition value in order to select the correct child node.

To illustrate the concept of level of blurriness, assume a motion blur switch has two child nodes. Also, assume that the modeler’s intent is to switch to the second node when the level of blurriness exceeds 10 %. Here, the XML tag associated with the switch should look like this.

Table 6-30: Example of a Motion Blur Switch with One Transition

```
<CDB:Switch name = "Motion_Blur">  
<Blurriness>10</Blurriness>  
</CDB:Switch>
```

6.10. Model Articulations

Requirements Class – Model Articulations

/req/openflight/model-articulations	
Target type	Operations
Dependency	Openflight Specification
Requirement 32	req/openflight/articulation-node-rotation
Requirement 33	/req/openflight/gimbal-limits
Requirement 34	req/openflight/articulation-flags

6.10.1. Definition

An OpenFlight DOF node is used to implement the concept of a CDB Articulation. The node gives the modeler controls over all 9 degrees of freedom, translation, rotation and scaling on all 3 axes. Generally, only one degree of freedom is allowed at a time and most often, that single degree of freedom is a rotation about a single axis. However, the modeler is free to allow any translation, rotation and, even scaling; even though stretching an articulation does not usually produce a realistic effect.

Since only one articulation is allowed per zone, the zone name is sufficient to identify and control the DOF node.

A CDB Articulation node is an OpenFlight DOF node with attributes in the form of XML tags. [Table 6-31](#) below presents the syntax of the XML tags stored in the DOF node's comment record.

Table 6-31: XML Tags for DOF

```

<CDB:Articulation name="name" id="id">

<Translation>

<X rate="value" />

<Y rate="value" />

<Z rate="value" />

</Translation>

<Rotation>

<X rate="value" />

<Y rate="value" />

<Z rate="value" />

</Rotation>

<Scaling>

<X rate="value" />

<Y rate="value" />

<Z rate="value" />

</Scaling>

</CDB:Articulation>

```

The above XML tag is necessary in two circumstances:

1. The articulation represents a DIS Articulated Part.
2. The articulation is to be animated automatically.

A CDB Articulation node has an optional name that is used to self-document the articulation. The optional identifier provides the corresponding DIS Articulated Part. It is suggested to use a name inspired from the DIS Articulated Part ID, when the identifier is supplied. For instance, DIS identifies as Primary Gun 1 the articulated part whose ID is 4416. That example would generate the following XML tags:

```
<CDB:Articulation name="Primary Gun 1" id="4416" />
```

Section 4.7.3 in reference [4] provides a list of DIS Articulated Part IDs.

It is possible to specify an optional Rate-of-Change for each Degree of Freedom along their X, Y, and Z axes for Translation, Rotation, and Scaling. The translation rate is expressed in meters per second. The rotation rate is expressed in degrees per second. Finally, the scaling rate is in units per second. When not specified, a default rate of zero is assumed.

Requirement 32

req/openflight/articulation-node-rotation

The translation rate *SHALL* be expressed in meters per second.
The rotation rate *SHALL* be expressed in degrees per second.
Finally, the scaling rate *SHALL* be expressed in units per second.
When not specified, a default rate of zero *SHALL* be used.

For instance, a primary radar antenna that rotates at a rate of 10 degrees per second about its Z-axis would require the following XML tags:

```
<CDB:Articulation name="Primary Radar 1" id="5376">  
  
<Rotation>  
  
<Z rate="10"/>  
  
</Rotation>  
  
</CDB:Articulation>
```

Another example, to illustrate how to attribute a rotating wind mill; assuming the mill rotates about the Y-axis at a rate of 5 degrees per second:

```
<CDB:Articulation>  
  
<Rotation>  
  
<Y rate="5"/>  
  
</Rotation>  
  
</CDB:Articulation>
```

Requirement 33

req/openflight/gimbal-limits

Gimbal limits are mandatory on DOF nodes and the appropriate flags *SHALL* be set to specify which degrees of freedom are controlled by a particular articulation.

Requirement 34

req/openflight/articulation-flags+

The Flags field is located at offset 376 in the OpenFlight DOF record and its value cannot be zero because the articulation *SHALL* control at least one degree of freedom.

6.10.2. Usage

Rotating Parts

A common problem in simulation is to correlate the linear speed of a model with the angular speed of its wheels. More generally, the client device simulation models often require the dimension of rotating parts. This information can be obtained from the zone extent; the bounding box surrounding a zone provides the dimension of rotating parts.

6.11. Model Light Points

The CDB standard does not make a distinction between light points and light sources. Both represent real lights that emit light and that can illuminate neighboring objects. In most current visual systems, a light point is a simple representation of a point source of light when viewed from a distance; it has no observable lighting effect on its immediate surroundings. In real-life however, as an observer moves closer to the light, its lighting or illuminating effect on the surrounding objects becomes increasingly observable; furthermore, the actual shape of the light also becomes more distinct.

In a typical simulator, client-devices may choose to limit the representation of the light to a single point and neglect the illuminating effect of the light on neighboring objects and terrain. For this reason, it is up to the client (and its RTP) to determine whether a light can illuminate its surroundings or not; the decision is based on the type of light and the inherent capabilities/capacity of the client.

Another point to consider is the fact that a light may have a very different representation depending on the client device. For instance, consider the visual representation of a light by an IG compared with the representation required by a radar system, NVG device or a FLIR device.

For all of these considerations, the CDB standard has adopted the following approach in defining lights. The OpenFlight file defines only the position, direction and the name of the light type; no other attributes are specified. The CDB standard provides a very elaborate light type naming convention. This convention permits clients to internally derive all of the properties and parameters needed to render the light. The approach is entirely device-independent. Modelers need not concern themselves with hundreds of parameters, many of which are often specific to underlying algorithms within the client. The naming convention ensures that the client has all of the information needed to capture the modeler's intent. Because the approach is device-independent, the rendering is limited only by the client's capabilities, not by the database itself.

As a result, lights in OpenFlight are defined by inserting an Indexed Light Point record into the OpenFlight scene graph. A vertex and a normal are stored in a Vertex List record that defines the

position and direction of the light. The name of the light type is stored in the Light Point Appearance Palette record.

The light type's name fully defines the appearance, animation and other characteristic relevant to the field of simulation. It is the responsibility of the client to supply the internal parameters that correspond to each of the light types supported by the CDB standard.

Light type naming conventions are defined in Section 2.3 of the Volume 1: OGC CDB Core Standard:Model and Physical Database Structure, CDB Core Model, Light Naming, and the list of names is presented in Annex J, Volume 2: OGC CDB Core: Model and Physical Structure: Annexes (formerly CDB Best Practice Volume 2 Appendix E).

Requirement 35	req/openflight/light-vertex-list
	A Vertex List record <i>SHALL</i> follow the OpenFlight Indexed Light Point record.

The list of vertices contains one vertex if a single light point is defined. The list contains several vertices when multiple independent light points are defined. An optional matrix and replication count permits the definition of a light string.

Table 6-32: OpenFlight Records for a Light Point

INDEXED LIGHT POINT
MATRIX (optional)
REPLICATE (optional)
PUSH LEVEL
VERTEX LIST
POP LEVEL

6.12. Model Attributes

This section defines a general attribution mechanism to add CDB and Vendor-specific attributes to any OpenFlight nodes. These attributes follows the rules of inheritance; they are automatically propagated from higher levels through lower levels of the OpenFlight graph. A child node inherits the attribution of its parent node.

6.12.1. Definition

Model attributes are added to OpenFlight nodes through a Comment record containing XML tags. The general format is as follows:

```

<CDB:node name="...">

<ns:Attribute name="..." value="..."/>

... other attributes

</CDB:node>

```

<CDB:node> identifies the node to which the attributes are added. The node token can take the following values:

- Zone
- Point
- Group
- Object
- Switch
- Face
- Mesh
- Articulation
- Light
- XRef
- LOD

The XML namespace (ns) of the attribute is optional; when present it identifies the owner of the attribute. When not specified, the default namespace is CDB.

Any CDB Attributes that are listed in section 5.7.1.3 can be used as node attributes. The name of the attribute is the key to search for the matching symbol into the metadata file named CDB_Attributes.xml; this file is described in section 5.1.7 and provides the means to interpret the value of the attribute.

6.12.2. Vendor Attributes

A vendor attribute is identified by its XML namespace. The standard uses the CDB namespace; a vendor may use any other string to identify itself.

Requirement 36

req/openflight/model-vendor-attributes

The definition of vendor attributes *SHALL* be stored in Vendor_Attributes.xml

It is understood that vendor attributes are not interpreted by any other client-devices other than

those supported by the vendor. Reliance by a vendor on Vendor Attributes can reduce the interoperability of the CDB with other vendors.

6.12.3. Examples

To add the LPH attribute to a CDB Light node, use the following comment:

```
<CDB:Light>  
  
<Attribute name="LPH" value="300"/>  
  
</CDB:Light>
```

Assume a T2DModel contains the Los Angeles International Airport as one of its 2DModels; the zone associated with the airport could use the APID attribute in the following manner:

```
<CDB:Zone name="Los Angeles International Airport">  
  
<Attribute name="APID" value="KLAX"/>  
  
</CDB:Zone>
```

A company named “Acme Inc.” uses the string “Acme” as the namespace qualifying its vendor-specific attributes. If the company wants to add the MyAttr attribute to a CDB Articulation, it could do so by using the following XML tags:

```
<CDB:Articulation name="Primary Gun 1" id="4416">  
  
<Acme:Attribute name="MyAttr" value="-1.23"/>  
  
</CDB:Articulation>
```

To interpret the attribute, a client application searches the file Vendor_Attributes.xml for an attribute whose symbol is MyAttr. When found, the application knows how to parse and interpret the attribute’s value. Furthermore, if the client application recognizes the identification of the vendor (Acme:), it knows what to do with MyAttr.

6.13. Model Textures

Requirements Class – Model Textures

/req/openflight/model-textures

Target type	Operations
-------------	------------

Dependency	Openflight Specification
------------	--------------------------

Requirement 37	req/openflight/texture-file-loading
-----------------------	-------------------------------------

Requirement 38	req/openflight/quarterly-textures
Requirement 39	req/openflight/monthly-textures
Requirement 40	req/openflight/texture-mipmap
Requirement 41	req/openflight/texture-palette-path
Requirement 42	req/openflight/texture-shadow-geometry
Requirement 43	req/openflight/model-skin-textures
Requirement 44	req/openflight/model-night-maps
Requirement 45	req/openflight/night-map-generation

To achieve a certain degree of realism, models require the use of textures. Furthermore, textures add details to a model without increasing its polygon count. This is excellent to reduce the complexity of the geometry but at the same time, it creates a load management issue for client devices that are interested in these textures.

Requirement 37	req/openflight/texture-file-loading+
-----------------------	--------------------------------------

In the case of GTModels and MModels, textures are separate files that *SHALL* be loaded after the model geometry files are read and loaded by client devices; in the case of GSModels and T2DModels, the textures can be loaded concurrently with the model geometry files.

A client device discovers the existence of textures while loading the model.

One of the goals of the CDB standard is to allow client devices to implement efficient load management mechanisms. For this reason, the Specification decouples as much as possible the texture aspect of a model from its geometry aspect. This is done by storing all textures related to Models in separate directories.

Recall that the texture filenames itself are constructed from the dataset number, the texture type (selectors 1 and 2), and the texture LOD and these are then concatenated to a modeler-specific texture name. Section 6.13, Model Textures, provides a description and usage of all of the CDB texture types for Models. The values of component selectors 1 and 2 convey a semantic meaning to the texture (time-of-year, paint scheme, night map, light map, normal map, etc) and determine whether the texture is to be used as base texture or as a subordinate texture and whether the texture is switchable (described in the next section).

6.13.1. Handling of Multi-textures

In OpenFlight, several types of textures can be applied in various combinations. Textures fall in two broad categories: Base and Subordinate.

Base Texture Layer

Base textures ^[25] are set of mutually exclusive model textures that provide texture color/intensity

modulation for the model. While a model can have many base textures, only one base texture can be referenced and applied to model geometry at a time.

The CDB standard supports the following type of Base Textures.

1. *Year-Round Texture*: A year-round texture used with GTModels, MModels, GSModels, T2DModels and T3DModels. In the case of MModels, base-textures are often replaced with an appropriate Paint Scheme texture (Uniform, Camouflage or Airline).
2. *Quarterly Textures*: A set of 4 textures, each representative of a quarter within the calendar year used with GTModels, GSModels, T2DModels and T3DModels.

Requirement 38	req/openflight/quarterly-textures
The textures <i>SHALL</i> be provided as a complete set, i.e., it is assumed that all 4 textures of the same kind (i.e., all four textures have their component selector 1 set to 003) and are all present in the model's texture directory.	

The presence of a quarterly texture reference in model geometry tells the client-device that a quarterly texture set is available. This allows the client-device to select any one of the available 4 textures at rendering time. Only one of the textures need be referenced by the OpenFlight scenegraph geometry, preferably the third quarter texture. It is also assumed that all 4 textures share the same UV mapping.

1. *Monthly Textures*: A set of 12 textures, each representative of a month within the calendar year used with GTModels, GSModels, T2DModels and T3DModels.

Requirement 39	req/openflight/monthly-textures
The textures <i>SHALL</i> be provided as a complete set, i.e., it is assumed that all the 12 textures are of the same kind (i.e., all twelve textures have their component selector CS1 = 002) and are all present in the model's texture directory.	

The presence of a monthly texture reference in model geometry tells the client-device that a monthly texture set is available. This allows the client-device to select any one of the available 12 textures at rendering time. Only one of the textures need be referenced by the OpenFlight scenegraph geometry, preferably the June texture. It is also assumed that all 12 textures share the same UVmapping.

2. *Camouflage Paint Scheme Textures*: Used on MModels with camouflage paint schemes should make use of this texture kind. Camouflages are listed in Annex O (Volume 2: OGC CDB Core: Model and Physical Structure Annexes). It is also assumed that all textures share the same UVmapping.

3. *Airline Paint Scheme Textures*: Used on MModels that represent commercial aviation airliners should make use of this texture kind to implement the airlines paint scheme and logos. This base texture addresses the need for multiple skins painted on identical aircraft type. For instance, the B767-300ER is operated by more than 60 airlines throughout the world. Annex O Volume 2: OGC CDB Core: Model and Physical Structure Annexes provides a complete list of Airliners. It is also assumed that all textures share the same UVmapping.
4. *Shadow Map Textures*: Used on MModels as pre-computed orthographic projections of the MModel. These textures are base textures used to accelerate the rendering of MModel shadows. Shadow map usage conventions are described in section 6.13.5.1, Model Shadow Textures.
5. *Motion Blur*: Used on MModels as pre-computed motion blurred textures of rotating parts (e.g., rotor disks). These textures are base textures used to aid client-devices in eliminating temporal aliasing artifacts. Motion blur textures conventions are described in section 6.9.2.3, Temporal Anti-aliasing.

Subordinate Texture Layer

Base textures can be supplemented with one or more ^[26] subordinate textures. Subordinate textures form a set of model textures that can be used to provide additional color/intensity modulation or illumination modulation detail to the Base texture.

The CDB standard supports the following types of subordinate textures.

1. *Night Map*: This subordinate texture is used to represent changes to models in their night configuration, typically as a result of lighting effects emanating from inside the model through windows. Night map textures conventions are described in greater detail in section 6.13.5.3, Model Night Maps.
2. *Detail Texture (Micro/Macro)*: This subordinate texture is used to add details to a base texture that lacks the necessary resolution to provide the correct depth perception. Detail textures conventions are described in greater detail in section 6.13.5.6, Model Detail Texture Maps.
3. *Contaminants*: These textures are used to simulate thin layers of matter that accumulate on surface top. Contaminant textures conventions are described in greater detail in section 6.13.5.7, Model Contaminant and Skid Mark Textures.
4. *Normal Map*: Normal mapping is a technique used for faking the lighting of bumps and dents; when used in conjunction with a render's light sources, it can add surface detail without using more polygons. This subordinate texture is a 3-component texture that encodes the normals at each texel. Tangent-space normal maps conventions are described in greater detail in section 6.13.5.5, Model Tangent-space Normal Maps.
5. *Reflection Map*: Conventions are described in detail in section 6.13.5.8, Model Cubic Reflection Maps.
6. *Light Map*: This subordinate texture is used to represent the effect of external light sources onto a model. Light map textures conventions are described in greater detail in section 6.13.5.4, Model Light Maps.
7. *Gloss Map*: A texture that describes whether a surface is matte or gloss; described in section 6.13.5.9, Model Gloss Maps.
8. *Material Texture*: To specify the composite materials at the level of a single texel; described in

section 6.13.5.10, Model Material Textures.

Client-devices are required to use the modeler supplied layer number to determine the order in which the subordinate textures are to be rendered. The base layer is always rendered first, followed by subordinate layer 1, 2, 3, etc. Gaps within the layer sequence are permitted.

Note that layer numbers are not assigned nor reserved to specific subordinate textures.

Texture Mapping Conventions

The following table provides the texture mapping for use with each kind of textures.

Table 1. Texture mapping for use with each kind of textures

Base Texture	Subordinate Texture	Kind	Mapping
Kind	Mapping	001	Modulate
051	Decal	002	Modulate
052	N/A	004	Modulate
053	Modulate	005	Modulate
054	Modulate	006	Modulate
055	Modulate	007	Modulate
056	Add	008	Modulate
057	N/A	009	Modulate

6.13.2. Default Gamma Corrections

The default gamma corrections of 3D model texture datasets are defined by the following set of parameters found in the Defaults.xml metadata file.

- Default_GSModelTexture_Gamma
- Default_GSModelInteriorTexture_Gamma
- Default_GTModelTexture_Gamma
- Default_GTModelInteriorTexture_Gamma
- Default_MModelTexture_Gamma

If a parameter is not found in Defaults.xml, or if Defaults.xml is not found in the metadata directory, assume a default gamma correction of 1.0.

See Annex S Volume 2: OGC CDB Core: Model and Physical Structure Annexes for the complete list of default parameters.

6.13.3. Texture Dimension

It is generally accepted by the modeling community to limit texture dimensions to a power of 2. The CDB standard goes a step further and enforces this practice.

To preserve the original texture resolution as much as possible, it is suggested to resize the source texture to the nearest ^[27] power of 2. For instance, if a source texture measures 72 pixels wide by 13 pixels high, it is recommended to resize it to 64 by 16 pixels.

$$\text{Texture Dimension} = 2^n \times 2^m$$

Where n and m are positive integers ($n, m \geq 1$).

Texture Mipmap

Requirement 40	req/openflight/texture-mipmap
-----------------------	-------------------------------

mipmaps associated with a given texture *SHALL* be present in the texture directory. Furthermore, the standard requires that mipmaps *SHALL* be stored in individual files.

$$\text{Number of Mipmaps} = \max(n, m) + 1$$

For instance, a texture whose dimension is $2^3 \times 2^4$ has a total of 5 mipmaps.

Texture Size

The naming conventions of all model textures are described in Chapter 3 Volume 1: OGC CDB Core Standard: Model and Physical Data Store Structure. For texture file whose name uses the W field, the value of the field is a power of 2 representing the largest dimension of a (possibly rectangular) texture.

$$\text{Texture Size} = 2^W$$

Where W is a non-negative integer ($W \geq 0$).

Texel Size

For texture file whose name uses the L field, the value of the field is related to the size of the texels in accordance to Section 3.3.6 Table 3-1: CDB LOD vs. Model Resolution (Volume 1: OGC CDB Core Standard: Model and Physical Data Store Structure).

6.13.4. Texture Palette

The OpenFlight Texture Palette record stores the names of all textures that are possibly referenced by the model; that includes all base and subordinate textures (i.e., all skins and all interchangeable textures). Each palette entry contains the path and filename of one texture.

Requirement 41

req/openflight/texture-palette-path

The path *SHALL* be relative to the OpenFlight file.

Below are examples of entries in the texture palette.

MModel Example

In the case of a moving model, the OpenFlight file resides in the MModelGeometry directory; for instance, the M1A2 resides in

```
\CDB\MMModel\600_MMModelGeometry\1_Platform\1_Land\225_United_States\1_Tank\1_1_225_1_1_3_0\
```

Its main texture is called M1A2 and resides in

```
\CDB\MMModel\601_MMModelTexture\M\1\M1A2\
```

The corresponding palette entry would be

```
...\\..\\..\\..\\..\\601_MMModelTexture\M\1\M1A2\D601_S005_T001_W11_M1A2.rgb
```

GTModel Example

In the case of a geotypical power pylon model, the OpenFlight file resides in the GTModel directory

```
\CDB\GTModel\510_GTModelGeometry\A_Culture\T_Comm\040_Power_Pylon\Lxx\
```

Assuming its texture is called Pylon, it resides in

```
\CDB\GTModel\511_GTModelTexture\P\Y\Pylon\
```

The corresponding palette entry would be

```
...\\..\\..\\..\\..\\511_GTModelTexture\P\Y\Pylon\D511_Sxxx_Txxx_Lxx_Pylon.rgb
```

GSModel Example

In the case of a geospecific model, its OpenFlight file resides in the GSModelGeometry directory. An example is

```
\CDB\Tiles\_lat\_lon\_300_GSModelGeometry\Lxx\Ux\
```

If the model refers to a geospecific texture, it resides in

```
\CDB\Tiles\_lat\_lon\_301_GSModelTexture\Lxx\Ux\
```

The corresponding palette entry would be

```
...\\..\\..\\..\\301_GSModelTexture\Lxx\Ux__\latlon__D301_Sxxx_Txxx_Lxx_Ux_Rx_TNAM.rgb
```

If the model refers to a geotypical texture, it resides in

```
\CDB\GTModel\511_GTModelTexture\_T\_N\_TNAM_
```

And the corresponding palette entry would be

```
...\\..\\..\\..\\..\\..\\..\\GTModel\511_GTModelTexture\_T\_N\_TNAM\511_Sxxx_Txxx_Lxx_TNAM.rgb
```

T2DModel Example

In the case of a tiled 2D model, its OpenFlight file resides in the T2DModelGeometry directory. An example is

```
\CDB\Tiles\_lat\_lon\_310_T2DModelGeometry\Lxx\Ux\
```

If the model refers to a geospecific texture, it resides in

```
\CDB\Tiles\_lat\_lon\_301_GSModelTexture\Lxx\Ux\
```

The corresponding palette entry would be

```
...\\..\\..\\..\\301_GSModelTexture\Lxx\Ux__\latlon__D301_Sxxx_Txxx_Lxx_Ux_Rx_TNAM.rgb
```

If the model refers to a geotypical texture, it resides in

```
\CDB\GTModel\501_GTModelTexture\_T\_N\_TNAM_
```

And the corresponding palette entry would be

```
..\..\..\..\..\..\..\GTModel\501_GTModelTexture\_T_\_N_\_TNAM_\D511_Sxxx_Txxx_Lxx_TNAM  
.rgb
```

6.13.5. Usages

Model Shadow Textures

Ideally, Model shadows should be generated at runtime by the client-device from the model's actual geometry. However, depending on the technique used by the client device, special textures called projected shadow maps may be used to cast shadows from Models.

When the projected shadow map technique is used, special object nodes are used to store the shadow polygons.

Shadow Geometry

Requirement 42 req/openflight/texture-shadow-geometry

When geometry exists for the purpose of casting shadows, it *SHALL* be located under an object node whose Shadow flag is set.



Figure 6-49. Using Shadow Polygons

Several object nodes can be used to store several polygons all textured with projected shadow maps. It may be desirable to also create separate shadow maps for major articulated parts and locate these shadow objects just under their corresponding DOF nodes.

Shadow Maps

Projected shadow maps are created by applying one, two or three orthographic projections on the model (or optionally on major articulated parts of the model). Figure 6-50: Example of a Shadow Map in the XY Plane shows one example of a shadow map of an aircraft in the XY plane. Similar

maps can also be produced for the YZ and ZX planes.



Figure 6-50. Example of a Shadow Map in the XY Plane

A projected shadow map is a monochrome (single-component) texture without transparency. It represents the mask to cut out the contour of the model. In theory, a black and white texture would be enough; however, shades of gray are permitted to represent semi-transparent surfaces that could be present on the model. In any case, a value of 0 (black) means the model does not block the passage of lights. The opposite value, 1 (white), indicates the model completely obstructs the light.

Because the shape of the model may change with damage states, each model state should have its own set of projected shadow maps. Section 6.9.2.2 describes damage states.

Shadow maps are general base textures. Their Texture Kind is 007 and their Texture Index is a sequence number when several shadow maps exist for the same Model.

To illustrate the naming convention, assume the shadow map from Figure 6-50: Example of a Shadow Map in the XY Plane, is called “aircraft”. According to Section 3.5.2.1, MModelTexture Naming Convention, and Section 5.5, MModel Library Datasets, the resulting file name would be:

D601_S007_T001_Wnn_aircraft.rgb

The value Wnn represents the texture size, 2^{nn} , and is explained in Section 6.13.3.2, Texture Size.

Note that if a client-device generates shadows on its own, without the support of pre-computed projected shadow maps, it can ignore all OpenFlight object nodes whose Shadow flags are set as well as all textures associated with these nodes.

Model Skin Textures

Models skins are base textures that correspond to one or more moving models paint schemes or one or more time-of-year representations of the cultural feature.

For instance, the same tank can be painted with several different colors to match various areas of operation. Below are two examples of the same tank, the M1A2 Abrams, painted for operation in a

desert area (Figure 6-51) or in a forest area (Figure 6-52).



Figure 6-51. The M1A2 Abrams with Desert Camouflage



Figure 6-52. The M1A2 Abrams with a Forest Camouflage

The two different textures for this tank qualify for use as skins since they have been designed in such a way that they can be exchanged for one another without affecting their mapping on the affected polygons.

Requirement 43

req/openflight/model-skin-textures

The mapping of textures for a given model or cultural features *SHALL* be identical since only the texture is changed, not the UV mapping.

OpenFlight itself does not provide an explicit mechanism to change the base texture assigned to faces. In fact, OpenFlight supports a single base texture per face record. The other textures that can be added to a face are called layers and none of them is a replacement for the base texture.

In order to have several skins for a single model, the CDB standard provides the mechanism defined in 6.14.5.2, Texture Switch. The enumeration values for each skin can be found in Annex O, Volume 2: OGC CDB Core: Model and Physical Structure Annexes.

The M1A2 used in the above examples has two skins. Each skin is made of a single texture that happens to be a mosaic of all the individual textures used by the model. Figure 6-53: M1A2 Desert Skin Mosaic below shows one of the M1A2 skins.

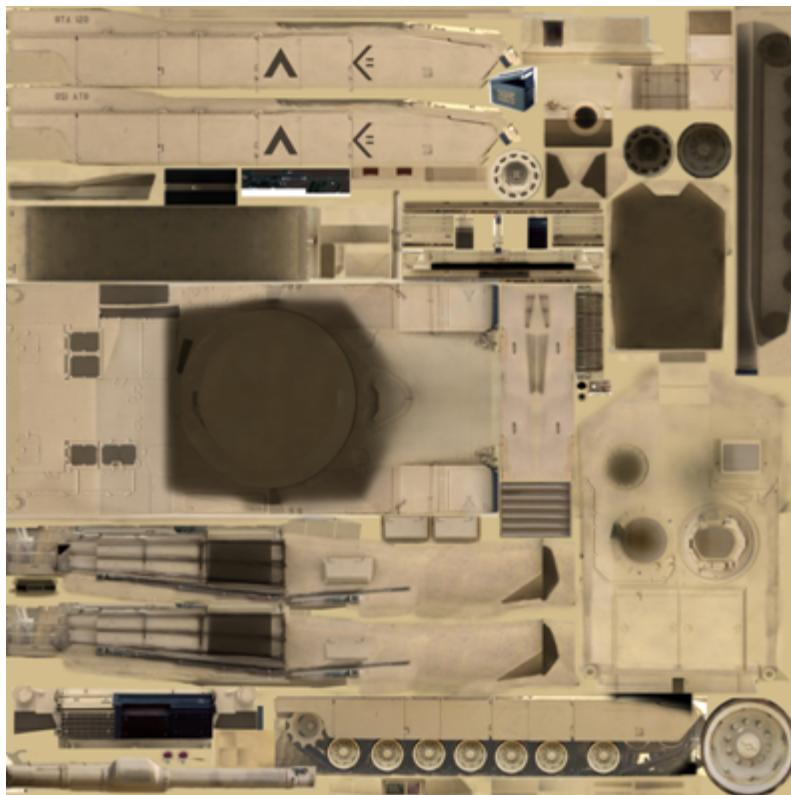


Figure 6-53. M1A2 Desert Skin Mosaic

The following texture kinds implement the concept of model skins:

- Kind 002 – Monthly Representation
- Kind 009 – Quarterly Representation
- Kind 004 – Uniform Paint Scheme
- Kind 005 – Camouflage Paint Scheme

- Kind 006 – Airline Paint Scheme

Paint schemes apply to moving models only. Annex O lists available paint schemes (Volume 2: OGC CDB Core: Model and Physical Structure Annexes).

Time-Of-Year representations are appropriate for cultural features. A good example is the case of leafy trees. Depending on the hemisphere and the latitude, several textures represent leafy trees at different stages during the year.

All texture kinds listed above are mutually exclusive; also, all instances of textures of a kind are mutually exclusive. Since all texture kinds above are base textures, and because only one base texture can be active at any one time on a face of a model, it follows that only one skin can be active at a time.

Model Night Maps

Night maps fall under the category of subordinate textures. Night and light maps (see next section) are both used at night to represent how interior and exterior light sources change the appearance of a Model. It is possible for a Model to have both a night and a light map, or just a light map. However, it is not possible to have a night map alone.

Simulator client-devices invoke a night map when the (simulated) light sources located inside the model need to change its appearance at night. This is the case when the interior light sources shine through openings like windows and portholes. To simulate the effect of lights emitted through these openings, a night map is created; it adds these bright window details normally missing from the base day texture.

The creation of a night map for models is left to the discretion of the modeler. Creating additional geometry for the windows and changing the material associated with the polygons to incorporate an emissive component can also produce a lighting effect similar to night maps. However, this approach requires additional (unnecessary) model geometry that adds additional computational load in the client-devices. For this reason, the use of night maps is recommended.

Light maps differ from night maps in that they combine the effect of exterior lighting with interior lights. A light map acts as a (colored) filter to mask portions of the model that are no longer visible at night when no ambient light exists.

A Model may have a relatively different aspect at night. This difference comes from two changes in the environment. The ambient illumination provided by sunlight is totally absent at night; only the moon and man-made light sources affect the appearance of objects. When present, the moon provides only a modest level of illumination when compared to the sun. In addition, the model itself might have internal lights turned on that are not modeled in day version of the texture but that do affect its appearance at night.

To illustrate these differences, imagine a building as seen during the day. No light seems to come out of its windows because the average daytime sunlight overwhelms any man-made lighting (internal to the building and coming out of the windows). At night, the outside walls of the building have not changed but light is now emanating from the windows. This is an important change that requires a modification to the texture used to represent the walls.

Another example of the use of a night map is the case of an aircraft flying at night. During daytime, the aircraft windows look dark while at night, light comes from the inside and the windows appear white.

Night maps are used to add details to base textures; details that are not visible during the day and that become visible at night. Therefore, a night map is not a replacement for the base texture. It is used in conjunction with the base texture.

The next figures illustrate the purpose of night maps.

Figure 6-54: Base Texture is the base texture used in modeling a commercial aircraft, the Airbus 330, during the day. Notice that portholes are represented by dark rounded rectangles because the lights in the cabin are off. The same is true for the cockpit windows located in the bottom right of the texture.



Figure 6-54. BaseTexture

Figure 6-55: Night Map, is the model's corresponding night map and shows the same portholes but this time brighter to reflect the fact that cabin lights are on. This time, notice the appearance of the cockpit windows as well as the presence of colors in them. Remember that this texture is used to add details that may be missing from the base texture.



Figure 6-55. Night Map

Requirement 44

req/openflight/model-night-maps

There are constraints imposed on night maps:

- A night map *SHALL* have the same size as its base texture.
- A night map *SHALL* use the same UV mapping as its base texture.
- A night map has a similar format as its base texture (RGB or Intensity) plus an alpha channel.

Night Map Generation

A night map is mapped on top of its base texture using a Decal texture environment. Since a night map is a subordinate texture, it is mapped on polygons using an OpenFlight multitexture record.

Requirement 45

req/openflight/night-map-generation

The Effect field of this multitexture record *SHALL* contain the value 0 indicating to use the Texture Environment mapping defined in the Texture Attribute file. The Environment Type field found in the Texture Attribute file *SHALL* contain the value 2 indicating a Decal environment mapping.

The night map alpha channel is in fact a mask identifying which portions of the base texture are replaced by the night map. Accordingly, the alpha channel contains a value of 0 when the corresponding texel of the base texture is left intact. However, the alpha channel will contain the value 1 when the corresponding texel of the base texture is replaced by the equivalent night map texel. Overall, the Decal environment mapping applies the following transformation to the base texture.

$$C = C_b \cdot (1 - A_n) + C_n \cdot A_n$$

where...

C_b is the color component (or intensity) of the base texture

C_n is the color component (or intensity) of the night map

A_n is the alpha component of the night map

Since the values found in the night map alpha channel are limited to 0 and 1, the resulting color will either be the one found in the base texture when A_n is 0 or the one found in the night map when A_n is 1.

Model Light Maps

Light maps also fall under the category of subordinate textures. Night maps and light maps are both used at night to represent how interior and exterior light sources change the appearance of a Model. It is possible for a Model to have both a night map and a light map, or just a light map. However, it is not possible to have a night map alone.

Light maps differ from night maps in that they combine the effect of exterior lighting with interior lights. A light map acts as a (colored) filter to mask portions of the model that are no longer visible at night when no ambient light exists.

A light map is used when active light sources are located on the outside of the model. This technique is used to simulate the appearance of a model when lit by local spotlights. For instance, spotlights may be used to illuminate a building at night. The light map provides the illumination pattern that represents the spotlight illumination on the building. The technique provides a convenient mean to produce interesting and entirely predictable lighting effects without resorting to computationally intensive local light sources. Their effects are already incorporated into special textures called light maps.

A light map also contains a mask related to the night map when present. Remember that a light map is a filter (a mask) to retain the detail associated with the base texture and its optional night map.

As opposed to a night map, a light map does not have constraints. More specifically:

A light map does not need to be of the same size as its base texture.

A light map has its own UV mapping.

A light map can be an intensity map or an RGB image.

Note that when light sources are modeled with light maps, they only affect the model onto which they are applied.

The next set of figures illustrates how light maps contribute to the lighting of a model. Note that a light map is not applied directly to the model base texture. The light map is first modified to take into account the ambient lighting, and then the resulting lighting is applied to the model.

Figure 6-56: Light Map, is the light map matching the base texture in Figure 6-54: Base Texture. Notice that it combines light lobes representing external light spots with the mask associated with internal light sources from the night map. This mask is used to key in details that stay visible at night.



Figure 6-56. Light Map

Figure 6-57: Combined Effect of Base Textures and Light Maps, shows on the actual aircraft the result of applying the light map from Figure 6-56: Light Map to the base texture from Figure 6-54: Base Texture. Notice that portholes and cockpit windows are still dark since the base texture has not been modified by the night map yet.



Figure 6-57. Combined Effect of Base Textures and Light Maps

Figure 6-58: Combined Effect of Night and Light Maps, shows the result of adding the night map to the base texture and then applying the light map. This time, we can clearly see the lights coming through portholes and cockpit windows.



Figure 6-58. Combined Effect of Night and Light Maps

How and When to Use Night Maps and Light Maps

The CDB standard recommends the use of night maps to represent lights that are internal to the model; this permits the client device to control the appearance of the model with internal lights on or off. This condition is usually true at night, hence the name of the texture.

Similarly, the CDB standard recommends the use of light maps to represent the effect of lights that are external to the model; this permits the client-device to control the appearance of the model with external spotlights on or off.

Note that night and light maps can be applied to any of the skins since skins are base textures.

How and When Not to Use Light Maps

A client device may discard light maps if the effect of external lights is internally generated by its GPU. It can be envisioned that future development of specialized hardware – such as graphics processor unit – will allow more of the lighting effects to be generated in real-time. When this time comes, artificial textures generated off-line such as light maps will become obsolete.

Model Tangent-space Normal Maps

A normal map is an RGB texture (without an alpha channel) where the normal to the surface is encoded in the Red, Green, and Blue channels. The normal (i, j, k) values are encoded in the following manner into the 8-bit value of each channel:

$$R [0, 255] = i [-1.0, +1.0]$$

$$G [0, 255] = j [-1.0, +1.0]$$

$$B [0, 255] = k [-1.0, +1.0]$$

The mapping is identical on all channels; the range of all possible 8-bit values (0, 255) is mapped linearly to the range of floating point values -1.0 to +1.0. This mapping provides a resolution of 2/255 or 0.0078.

In addition, the reader should note that the floating-point value 0.0 has no exact integer equivalent [\[28\]](#). Here, the closest value to 0.0 is approximately ± 0.0039 and is obtained when the channel contains 127 or 128.

Besides this particular encoding of the normal into the RGB channels, a normal map has all the other attributes of a standard RGB texture whose format is defined in the SGI Image File Format [\[29\]](#).

In the industry, there are at least two types of Normal Map: object-space normal map, and tangent-space normal map. Both types have their pros and cons. The CDB standard opts for tangent-space normal map. A sample is shown here.



Figure 6-59. Normal Map Sample

Typically, the normal points away from the surface, and not toward the underlying surface. For this reason, the value of the k-component of the normal is positive, most of the time, resulting in a bluish tint of the map. A negative k-component could indicate the presence of a cliff with an overhang, for instance.

Model Detail Texture Maps

A detail texture map is 1- or 3-component (aka channel) texture where each texel is represented as an 8-bit unsigned integer. A detail texture exhibits two important properties; it has a neutral luminance (intensity) and chrominance (color). This is achieved by applying the following constraints:

The 8-bit unsigned value of each texel is scaled to a floating point value in the range -1.0 to 1.0

The average value of an individual component is always 0.0

The Detail texture is mapped on the underlying surface through a simple addition operation

The net effect of applying a Detail Texture Map is to highlight (> 0) or darken (< 0) fragment details on the underlying surface. When using a single component detail texture map, only the intensity of the resulting image is affected; when using a 3-component detail texture map, the color is also varied.



Figure 6-60. Detail Texture Map Sample

Recall that a detail texture map is a mean of adding high-frequency (spatial) details to a rather low-frequency image.

Model Contaminant and Skid Mark Textures

Historically, Image Generators of civil aviation simulators provided the means for flight instructors to control the appearance of airport runways, taxiways, and roads with various surface contaminants. To this end, the CDB provides a set of standardized Model Contaminant and Skid Mark Textures that are commonly used in flight simulators and listed in Annex O, Volume 1.1: OGC CDB Core: Model and Physical Structure: Informative Annexes. These textures are typically four-component (R, G, B, alpha) textures that act as an overlay to airport surfaces.

Model Cubic Reflection Maps

Reflection mapping (aka environment mapping) is an efficient image-based lighting technique for approximating the appearance of a reflective surface by means of a precomputed texture image. The texture is used to store the image of the distant environment surrounding the rendered object.



Figure 6-61. Environment Used to Produce Reflection Map



Figure 6-62. Resulting Reflection Map



Figure 6-63. Rendered Reflection Map onto Reflecting Cube

The CDB standard assumes that the surrounding environment is stored using a cubic mapping approach. In this technique, the environment is projected onto the six faces of a cube and stored as six square textures or unfolded into six square regions of a single texture. The reflection mapping approach is more efficient than the classical ray tracing approach of computing the exact reflection by tracing a ray and following its optical path. The reflection color used in the shading computation at a pixel is determined by calculating the reflection vector at the point on the object and mapping it to the texel in the environment map. This technique often produces results that are superficially similar to those generated by raytracing, but is less computationally expensive since the radiance value of the reflection comes from calculating the angles of incidence and reflection, followed by a texture lookup, rather than followed by tracing a ray against the scene geometry and computing the radiance of the ray, simplifying the GPU workload.

Note however that in most circumstances, a mapped reflection is only an approximation of the real reflection. Environment mapping relies on four assumptions:

- All radiance incident upon the *statically-positioned* object being shaded comes from an *infinite distance*. When this is not the case, then a) the reflection of nearby geometry appears in the wrong place on the reflected object, and b) no parallax is seen in the reflection.
- The object being shaded is *convex*, such that it contains no self-interreflections. When this is not the case the object does not appear in the reflection; only the environment does.
- The environment map is *valid for the location* for which it was generated.
- The environment is *static*.

Model Gloss Maps

A gloss map is a texture that describes whether a surface is matte or gloss. The texture is used to modulate specular highlights in the same way the material shininess does. A gloss map is stored as an 8-bit single channel texture (a grey-scale image) where texels are mapped to the range 0.0

(matte) to 1.0 (glossy). The values in the gloss map play the same role as the single shininess value found in the OpenFlight material assigned to a polygon. In this way, the gloss map can effectively modulate the specularity on a per-pixel basis. Note that if the material applied to the surface has no specular component, then the gloss map has no effect.

Model Material Textures

Material textures fall under the category of subordinate textures. They are mapped to Models the same way as any other textures. As such, the surfaces these textures are mapped to possess their own set of UV mapping.

A material texture tells the interested client devices (e.g., FLIR, CGF) what the underlying surface is made of. For this reason, a material texture is not at all related to a base texture. The two are completely independent and exist separately. A material texture does not require that a base texture be applied to the model. In fact, it is perfectly possible to create a Model that does not use texture except for a single material texture describing its various materials.

The <Material> tag presented in section 6.5.3 is a high level mean of providing material information about the geometry of a model. With the use of a material texture, the modeler can provide highly detailed material information about the same model.

In short, the <Material> tag supports a polygon-based approach of sensor client devices such as FLIR, NVG, and RADAR. A Material texture is a texel-based approach supporting an implementation of such client devices with a much higher resolution.

In the case of the Raster Material dataset (dataset code 005) applied onto the terrain, it is conceivable that multiple layers and mixtures of materials are required to represent the rich variety of materials found on the earth surface. However, for Models, a single material layer is probably adequate for the vast majority of man-made objects.

6.14. Model Descriptor (Metadata) Datasets

Each type of 3D Models has its set of ModelDescriptor datasets; they are:

1. GSModelDescriptor
2. GSModelInteriorDescriptor
3. GTModelDescriptor
4. GTModelInteriorDescriptor
5. MModelDescriptor

Each file is needed to summarize and regroup the information concerning one portion of a model, its shell or its interior. The information are collected and stored in an XML file to help client devices implement efficient load management mechanism.

The format of the model descriptor file is as follows:

```
<Model_Metadata>
```

```
<Name>...</Name>

<Identification>...</Identification>

<Mass>...</Mass>

<Parts>...</Parts>

<Textures>...</Textures>

<Configurations>...</Configurations>

<Composite_Material_Table>...</Composite_Material_Table>

</Model_Metadata>
```

6.14.1. Model Name

The <Name> is an arbitrary string from the character set presented in section 2.2. This name is the human readable version of the model identification code that follows.

6.14.2. Model Identification

Models are either modeled representation of cultural features or moving models. In both cases, the CDB standard has a unique way to identify them. For moving models, the identification scheme corresponds to their DIS entity type. For cultural features, their feature code is used.

Moving Model Identification

The DIS entity type is a list of up to seven integers and can be specified in two different manners. All fields have a default value of zero.

First, you can use a list of one to seven integers as illustrated here:

```
<Identification>

<DIS_Entity_Type>

<List>...</List>

</DIS_Entity_Type>

</Identification>
```

Or you can use this more verbose syntax to specify the value of individual fields:

```
<Identification>

<DIS_Entity_Type>

<Kind>...</Kind>

<Domain>...</Domain>
```

```
<Country>...</Country>  
<Category>...</Category>  
<Subcategory>...</Subcategory>  
<Specific>...</Specific>  
<Extra>...</Extra>  
</DIS_Entity_Type>  
</Identification>
```

All fields are limited to the range [0, 255] except the country code that can go up to 65535.

Cultural Feature Identification

For cultural features, their feature code is specified in the following manner:

```
<Identification>  
<Feature_Attribute_Catalog_Code>  
<Code>...</Code>  
<Subcode>...</Subcode>  
</Feature_Attribute_Catalog_Code>  
</Identification>
```

The feature code has a fixed format of two letters followed by three digits; it is the same as the feature attribute described in section 5.7.1.3.24 of the CDB Standard, Volume 1: OGC CDB Core Standard:Model and Physical Database Structure. The subcode is an optional integer in the range [0, 999].

6.14.3. Model Mass

The model mass is optional. It makes sense only when the Model represents a moving model.

```
<Mass>  
<Total>...</Total>  
<Metal>...</Metal>  
</Mass>
```

The total mass of the model is expressed in kilograms. The portion of the model that is made of a metallic alloy is expressed as a percentage of the total mass. The value of <Metal> lies in the range [0.0, 1.0].

When the model mass is specified, the total mass is mandatory while the metallic portion is optional. The total mass must be larger than zero. The metallic portion defaults to zero.

6.14.4. Model Parts

A Model may be separated into several parts. If the complexity of a part justifies it, each part may be split into multiple files.

The whole section is optional. It is required only if more than one part exists or if a part has more than one file.

If present, the section is a list of at least one part formatted like this.

```
<Parts>
```

```
<Part no="no" numFiles="numFiles" name="partName" />
```

```
...
```

```
</Parts>
```

The part number is mandatory. It starts at 1 and increases by 1 for each subsequent part. The first part is also referred to as the body of the model.

The number of files is optional and defaults to 1.

The part name ^[30] is optional and is used only to improve the readability of the file.

6.14.5. Model Textures

This section lists all textures that could be possibly used by the model. In the event the model does not use texture, the whole section is omitted. The section contains a list of textures and optional texture switches.

```
<Textures>
```

```
<Texture .../>
```

```
<Texture .../>
```

```
...
```

```
<Switch .../>
```

```
<Switch .../>
```

```
...
```

```
</Textures>
```

Texture Metadata

For each texture, the section provides the client device with the necessary information to decide when and which texture mipmap should be loaded.

The section is formatted like this.

```
<Texture no="number" name="name">  
<Dataset>...</Dataset>  
<Kind>...</Kind>  
<Index>...</Index>  
<Mipmap>min max</Mipmap>  
<Resolution>...</Resolution>  
<Coverage>  
<U>min max</U>  
<V>min max</V>  
</Coverage>  
</Texture>
```

The texture number is a strictly positive integer to uniquely identify the texture. The texture name corresponds to the TNAM field in the texture filename as defined in Section 3.5.2.1, MModelTexture Naming Convention.

The <Dataset>, <Kind>, and <Index> fields correspond respectively to the dataset number and component selectors 1 and 2; they match the D, S and T fields in the texture filename.

The mipmap field defines the smallest and largest mipmap available for this texture. The value of this field is used to compose the W field in the texture filename of moving models (see examples in section 3.5.2.4).

The texture resolution is expressed in texels per meter ^[31]. It is the same for both the U and V axes even though it is recognized that it can differ between the two dimensions. The intent is to provide an indication of how precise the texture is when mapped to the model geometry. It helps client device decide which mipmap is more appropriate to use.

The texture coverage is optional and defines the minimum and maximum values for the U and V texture coordinates. This information indicates if the texture is repeated along one or both axes. If the coverage is in the interval [0, 1], the texture is clamped; otherwise, it is repeated.

Texture Switch

A Texture Switch is defined when switchable textures appear in the list of textures. Switchable

textures are textures that can be exchanged for one another because they share the same UV mapping, as explained in section 6.13.5.2, Model Skin Textures.

The section is formatted like this.

```
<Switch no="number" name="name">  
<State no="number" name="name" textures="list"/>  
...  
</Switch>
```

The switch number is a unique positive integer identifying the switch. The switch name is a unique string limited to 32 characters; all switches are uniquely identified by a number and a name.

A switch has two or more states; each state selecting a list of one or more textures. State numbers are consecutive and start at 1. The state name is a unique string also limited to 32 characters. The list of textures associated with a state contains the texture numbers of the selected textures. Note that a state (e.g., a skin) may require more than one texture, hence the need to specify a list of textures associated with a state.

Example

Assume that the following two textures are stored in the M1A2 texture folder:

```
\CDB\MMModel\601_MMModelTexture\M\1\M1A2\  
D601_S004_T005_Wxx_M1A2.rgb  
D601_S005_T001_Wxx_M1A2.rgb
```

Here is an excerpt of the model metadata presenting the two textures, the switch, and the two corresponding states.

```
<Textures>  
<Texture no="3" name="M1A2">  
<Dataset>601</Dataset>  
<Kind>4</Kind>  
<Index>5</Index>  
...  
</Texture>  
<Texture no="10" name="M1A2">  
<Dataset>601</Dataset>  
<Kind>5</Kind>
```

```

<Index>1</Index>

...
</Texture>

...
<Switch no="1" name="Paint Scheme">

<State no="1" name="Uniform Beige Paint" textures="3"/>

<State no="2" name="Desert Camouflage" textures="10"/>

</Switch>

</Textures>

```

The texture switch is named “Paint Scheme” because it controls the selection of the paint scheme to apply to the M1A2. The first state selects texture 3 which corresponds to a beige uniform paint; the second state selects texture 10 corresponding to a desert camouflage.

Note that the texture switch mechanism is not limited to base textures; it can be used to switch light maps for example.

6.14.6. Model Configurations

Often, a single Model – especially a moving model – comes with a variety of possible equipment and/or ordnance. This can be as diversified as fuel tanks, missiles, radio emitters, etc. To configure a model with its ordnance, the CBD Specification defines the concept of model configuration. A configuration defines the set of equipment and ordnance attached to the various stations found on the model.

The configuration section is optional. It is a list of one or more configurations defined like this.

```

<Configurations>

<Configuration>...</Configuration>

...
</Configurations>

```

Defining Stations in a Configuration

A configuration is a sequence of one or more stations, each defining one piece of equipment in one location.

```

<Configuration name="ConfigName">

<Station name="StationName">

```

```
<Location>...</Location>  
<Equipment>...</Equipment>  
</Station>
```

i. other stations as needed

```
</Configuration>
```

The configuration and station names are both optional and are used for documentation purposes only.

The location of a station is defined by its fully qualified name as specified in section 6.5.5, Model Zone Naming.

Defining Equipment in a Station

The equipment is defined by either its DIS identification or a reference to an external part, and an optional anchor point.

```
<Equipment name="EquipmentName">  
<Identification>...</Identification>  
<External_Part>...</External_Part>  
<Anchor>...</Anchor>  
</Equipment>
```

The equipment name is optional and is used for documentation purposes only.

The anchor point is specified in the same manner as the location of a station, by providing its path (on the subordinate model) as specified in section 6.5.5, Model Zone Naming.

Defining Equipment Names

Either a DIS emitter name or a DIS entity type identifies the equipment. When the equipment is an emitter, the syntax is as follows.

```
<Identification>  
<DIS_Emitter_Name>...</DIS_Emitter_Name>  
</Identification>
```

Emitter names are defined by the DIS standard. For DIS, refer to Section 8.1.1 of reference [4] for a list of DIS Emitter Names. For the HLA standard, the RPR-FOM lists all emitter names. To avoid confusion, both DIS and HLA refer to emitter names using numbers. For instance, the NATO emitter AS 15 KENT altimeter is referred to as emitter 8735.

When the equipment is another entity (e.g., a missile), its DIS entity type is supplied in the following

manner.

```
<Identification>  
  
<DIS_Entity_Type>...</DIS_Entity_Type>  
  
</Identification>
```

Recall that the DIS entity type is a list of up to 7 numbers as defined by reference [4]. For example, the AGM-114K-SAL Hellfire missile would be referred to as:

```
<DIS_Entity_Type>  
  
<List>2 2 225 1 3 5 1</List>  
  
</DIS_Entity_Type>
```

or

```
<DIS_Entity_Type>  
  
<Kind>2</Kind>  
  
<Domain>2</Domain>  
  
<Country>225</Country>  
  
<Category>1</Category>  
  
<Subcategory>3</Subcategory>  
  
<Specific>5</Specific>  
  
<Extra>1</Extra>  
  
</DIS_Entity_Type>
```

Equipment can also be defined by a reference to an external part if need be. A good example of such equipment is a fuel tank.

```
<External_Part>  
  
<Part_Number>...</Part_Number>  
  
<Configuration>...<Configuration>  
  
</External_Part>
```

The external part is identified by its part number as defined previously in the <Parts> section.

The external part may also require its own configuration. Take the example of a Hellfire missile rack attached to an attack helicopter like the Apache. The rack can hold up to 4 missiles. Each missile attaches to one of four separate weapon stations located on the rack. For this more complex

example, assume the rack has only two missiles out of four. This configuration can be specified with the following piece of XML.

```
<External_Part>

<Part_Number>1</Part_Number>

<Configuration>

<Station name="Missile 1">

<Location>\Missile_Rack\Attach_Point[1]</Location>

<Equipment>

<Identification>

<DIS_Entity_Type>

<List>2 2 225 3 5 1</List>

</DIS_Entity_Type>

</Identification>

</Equipment>

</Station>

<Station name="Missile 2">

<Location>\Missile_Rack\Attach_Point[2]</Location>

<Equipment>

<Identification>

<DIS_Entity_Type>

<List>2 2 225 3 5 1</List>

</DIS_Entity_Type>

</Identification>

</Equipment>

</Station>

<Configuration>

</External_Part>
```

With the help of model configurations, it is possible to create several variants of a single Model,

each variant defined by its own configuration.

This way, one Apache can have two configurations, one when equipped with Hellfire missiles and one when equipped with rocket launchers.

6.14.7. Model Composite Materials

The composite material table is the last component of the Model Metadata and is defined in section 2.5.2.2, Composite Material Tables (CMT) in the CDB Standard, Volume 1: OGC CDB Core Standard: Model and Physical Database Structure.

- [5] In Appendix C, the section called *Database Hierarchy* explains in details how OpenFlight organizes its graph.
- [6] CDB-compliant readers must ignore all OpenFlight extension records.
- [7] The syntax to specify a XML namespace is <ns:element> where ns is the namespace and element is the XML element name (or simply tag).
- [8] The DIS standard defines a different orientation for the axes of its coordinate system. DIS defines the X-axis as pointing to the front of the entity; the Y-axis pointing to its right and the Z-axis pointing down. Adoption of the DIS convention for use in the CDB standard has been rejected due to the fact that the majority of models in existence have been created using tools such as Creator. The CDB standard follows the same convention as the one used by these commercial tools. Note that if the CDB standard were to follow the DIS convention, modelers would be required to create and edit their models upside down with respect to the reference plane provided by their tool.
- [9] Most figures in this chapter are actual screenshots from Creator.
- [10] Creator or CAD Creator is a Siemens product. Mention of Creator in the CDB standard IS NOT an endorsement of that product.
- [11] Typically, these transformations are used by modeling editing tools only.
- [12] The local origin of the model is translated to the (lat, long) coordinates of the point feature. The elevation component is either obtained from the point feature (AHGT=True) or obtained from the elevation of the terrain at (lat, long) coordinates of the point-feature (AHGT=False). The model's XY plane is rotated in accordance to the feature's AO1 attribute. The model's Z-axis is adjusted so that it is tangential to the WGS-84 earth model.
- [13] A dictionary of CDB Component Names is provided in Appendix F.
- [14] The OpenFlight Face and Mesh records both have a flag called *Terrain Culture Cutout*. This flag is commonly designated as the *Cultural Footprint* flag within the simulation industry.
- [15] This increases compatibility with OpenFlight readers that are not CDB-compliant.
- [16] The Model Footprint polygon is not an absolute Z-positioned 3D polygon generated by the intersection of the model with the specific terrain it sits on - that would make the footprint of the model specific to that terrain. Furthermore, a different 3D polygon would be required for each possible terrain LOD.
- [17] This increases compatibility with OpenFlight readers that are not CDB-compliant.
- [18] OneSAF Ultra High Resolution Building (UHRB) Object Model.
- [19] This definition of a room footprint comes from the UHRB Specification.
- [20] Note that the CDB standard follows established UHRB conventions as it relates to partitions, namely that all partitions must be clipped so that there are no more than two neighboring rooms.
- [21] This definition can be found on page 3 of reference [4]
- [22] Man-made structures and tree vegetation do not tilt regardless of the terrain they are on.
- [23] Users of Creator 3.0 will find the exact same definition for Significance in both the Group and Object Attributes Help pages.
- [24] Note that, on top of the Damage State field, the DIS Entity Appearance structure has two flags to describe the Mobility and Fire Power of the entity. This is also true for HLA and version 2 of the RPR-FOM which provides for two flags to describe the fire power and mobility of a physical entity on top of the field used to describe the damage state.
- [25] The CDB Standard uses the term "base texture" the same way as OpenFlight and Creator do.
- [26] OpenFlight natively permits up to seven subordinate textures for a total of eight textures including the base texture.
- [27] Note here that we do not recommend resizing to the *next* power of 2; instead, resize to the *nearest* power of 2.
- [28] The conventional OpenGL mapping specifies that -1 and 1 can be represented exactly, but 0 can not.
- [29] <ftp://ftp.sgi.com/sgi/graphics/grafica/sgimage.html>
- [30] As a guideline, it is suggested to set the part name the same as the global zone name of that part. For instance, if the part represents an external fuel tank, a good name for both the part and its global zone would be "*External Fuel Tank*".
- [31] This unit of measurement (texels per meter) is akin to DPI (dot per inch) used to quantify the resolution of printers and displays.

Annex A: Conformance Class Abstract Test Suite (Normative)

A.1. Conformance class: CRS

This test is to validate that the Coordinate Reference System (CRS) header content is properly structured in the Openflight header record.

Conformance Class	/conf/openflight/header-crs	
Requirements Class	/req/openflight/ crs	
Dependency	Openflight Specification	
Test 1	/conf/header-crs/crs	
	Requirement	/req/openflight/model-global-zone
	Test purpose	Verify that the CRS codes are properly set
	Test method	Pass if the values are set to the default of “0”
	Test type	Conformance

A.2. Conformance Class: Tree Structure

Conformance Class	/conf/openflight/tree-structure	
Requirements Class	/req/openflight/tree-structure	
Dependency	Openflight Specification	
Test 2	/conf/tree-structure/model-global-zone	
	Requirement	/req/core/openflight/model-global-zone
	Test purpose	Verify that a CDB OpenFlight Model has a global zone as its root node. This node identifies the model.
	Test method	Visual. Pass if the model has a global zone as its root node. Check file system structure.
	Test type	Conformance
Test 3	/conf/tree-structure/2dmodel	
	Requirement	/req/openflight/2dmodel

	Test purpose	Verify that each 2DModel is implemented as a Model Zone with its own subgraph.
	Test method	Visual. Pass if the a model zone has its own sub-graph
	Test type	Conformance
Test 4	/conf/tree-structure/2dmodel-rules	
	Requirement	/req/openflight/2dmodel-rules
	Test purpose	Verify the A 2DModel has at least two layers: layer 0 and layer 1. Verify that Layer 0 is empty because it represents the terrain on top of which subsequent layers are applied. Verify that each layer is composed of exactly one OpenFlight Object node.
	Test method	Pass if there are at least two layers. Visual inspection
	Test type	Conformance
Test 5	/conf/tree-structure/2dmodels	
	Requirement	req/openflight/2dmodels
	Test purpose	Verify that for T2DModels, node attributes are defined only at the zone level; that is, at the global zone or at the individual 2DModel zones. Node attributes are not permitted at the Group, LOD, Object, Face, and Mesh node levels.
	Test method	Visual, Pass if node attributes are defined only at the zone level
	Test type	Conformance
Test 6	/conf/tree-structure/xref	
	Requirement	req/openflight/xref
	Test purpose	Verify that all external OpenFlight references are made using a relative path.
	Test method	Visual. Pass if external references use relative paths.

	Test type	Conformance
--	-----------	-------------

A.3. Conformance Class: Modeling Conventions

The following are the conformance tests for the OpenFlight modeling conventions.

Conformance Class	/conf/openflight/model-conventions	
Requirements Class	/req/openflight/model-conventions	
Dependency	Openflight Specification	
Test 7	/conf/model-conventions/crs-models	
	Requirement	/req/openflight/crs-models
	Test purpose	Verify that CDB Models use the same coordinate system convention as OpenFlight
	Test method	Visual. Pass if the CRS definitions are the same.
	Test type	Conformance
Test 8	/conf/model-conventions/model-origin	
	Requirement	/req/openflight/model-origin
	Test purpose	Verify that the model origin <i>is</i> located at the center of the bounding rectangle and that along the Z axis, the origin <i>as selected</i> allows the model to be correctly positioned on the ground for ground related models or on a water plane for surface and subsurface platforms.
	Test method	Visual. Pass if the model is correctly positioned in the display.
	Test type	Conformance
Test 9	/conf/model-conventions/t2-model-coordinates	
	Requirement	/req/openflight/ t2-model-coordinates

	Test purpose	Verify that the latitude (y) and longitude (x) coordinates are expressed in decimal degrees and that the values are relative to the file's (implicit) origin which is the south-west corner of the tile.
	Test method	Visual. Pass if coordinates are expressed in decimal degrees..
	Test type	Conformance
Test 10	/conf/model-conventions/roll-pitch-yaw	
	Requirement	req/openflight/roll-pitch-yaw
	Test purpose	Verify that the Pitch, Roll and Yaw angles refer to rotations around the X, Y, and Z axes and Angles are measured in degrees. The Roll and Yaw angles vary from ± 180 degrees while the Pitch angle is limited to the range ± 90 degrees.
	Test method	Visual. Pass if all angles are degrees
	Test type	Conformance
Test 11	/conf/model-conventions/geometry	
	Requirement	req/openflight/geometry
	Test purpose	Verify the implementers of the CDB OpenFlight standard adhere to set of constraints, rules and guidelines as defined in OpenFlight Requirement 11 when creating the geometry of Models.
	Test method	Visual. Pass if all polygons are convex, all vertices of a polygon are in the same place, and that all polygon vertices (coordinates) have a counter-clockwise ordering.
	Test type	Conformance
Test 12	/conf/model-conventions/ geometry-layer-constraint	
	Requirement	req/openflight/ geometry-layer-constraint

	Test purpose	Verify that Layer 0, the base layer, contains geometry that completely encompasses the geometry of subsequent layers. Other layers are processed in order, one after the other. A layer is made of one or more nodes; all nodes of a given layer have the same relative priority.
	Test method	Visual and software. Software can check processing order. Pass if the base layer contains a geometry that encompasses the geometry of sub-layers.
	Test type	Conformance

A.4. Conformance Class: Model Zones

This conformance class tests requirements related to OpenFlight model zones.

A model zone represents a component of interest on the Model. A model zone (as well as the component it represents) occupies a certain volume and is delimited by a bounding box. At least one simulator subsystem must be interested in a specific component to justify the creation of a corresponding zone. Examples of zones are a turret on a tank, or an engine on a platform, or an entrance door on a building, etc

Conformance Class	/conf/openflight/model-zones	
Requirements Class	/req/openflight/model-zones	
Dependency	Openflight Specification	
Test 13	/conf/model-zones/model-zone-bounding-box	
	Requirement	/req/openflight/model-zone-bounding-box
	Test purpose	Verify that a model zone has a Bounding Box.
	Test method	Visual. Pass if there is a bounding box.
	Test type	Conformance
Test 14	/conf/model-zones/zone-name	
	Requirement	/req/openflight/zone-name
	Test purpose	Verify that there is a zone name.

	Test method	Visual. Pass if there is a zone name.
	Test type	Conformance
Test 15	/conf/model-zone/global-zone	
	Requirement	/req/openflight/ global-zone
	Test purpose	Verify that the model has at least one zone that encompasses the whole model and that is called the model global zone.
	Test method	Visual. Pass if of there is one zone encompassing the whole model.
	Test type	Conformance
Test 16	/conf/ model-zone /hot-spot-temperature	
	Requirement	req/openflight/hot-spot-temperature
	Test purpose	Verify that temperatures are expressed as integer Celsius.
	Test method	Visual. Pass if temperatures are integer Celsius
	Test type	Conformance
Test 17	/conf/ model-zone /model-footprint-zones	
	Requirement	req/openflight/ model-footprint-zones
	Test purpose	Verify that Client-devices assume that the geometry that is associated with a Model Footprint Zone is hidden.
	Test method	Visual. Check Client code.
	Test type	Conformance
Test 18	/conf/ model-zone /model-footprint-hierarchy	
	Requirement	req/openflight/model-footprint-hierarchy
	Test purpose	Verify that the Model Footprint is placed under a CDB Footprint Zone node.
	Test method	Visual. Pass if model footprint is correctly placed in hierarchy.

	Test type	Conformance
Test 19	/conf/ model-zone /model-cutout-zones	
	Requirement	req/openflight/model-cutout-zones
	Test purpose	Verify that the Model Cutout is modeled as a set of OpenFlight Face or Mesh records. Verify that client-devices assume that the geometry that is associated with a Model Cut-Out Zone is hidden and cut-out...
	Test method	Visual.
	Test type	Conformance
Test 20	/conf/ model-zone /model-cutout-geometry	
	Requirement	req/openflight/model-cutout-geometry
	Test purpose	Verify that the Cutout geometry is placed under a CDB Model Cutout Zone node.
	Test method	Visual.
	Test type	Conformance
Test 21	/conf/ model-zone /model-pseudo-interior-zone	
	Requirement	req/openflight/model-pseudo-interior-zone
	Test purpose	Verify that since the pseudo-interior is a placeholder for the real interior that it is placed under its own subgraph and identified by a CDB zone whose name is “Interior”.
	Test method	Visual.
	Test type	Conformance
Test 22	/conf/ model-zone /model- interior-zone	
	Requirement	req/openflight/model- interior-zone
	Test purpose	Verify that The Model interior itself has a global zone whose name is “Interior”.
	Test method	Visual.
	Test type	Conformance

A.5. Conformance Class: Model Points

A model point is similar to a model zone; it identifies a location on the model that is of interest to at least one simulation client device. A point defines a local coordinate system on the model. Hence, a point has a position and an orientation. The following tests ensure that model points are correctly modeled and comply with the model points requirements class.

Conformance Class	/conf/openflight/model-points	
Requirements Class	/req/openflight/model-points	
Dependency	Openflight Specification	
Test 23	/conf/model-points/model-point-damage-states	
	Requirement	/req/core/openflight/model-point-damage-states
	Test purpose	Verify that there is a single definition of this point for all damage states and all levels of details for a given model.
	Test method	Visual. Pass if there is a single definition.
	Test type	Conformance
Test 24	/conf/ model-points/model-dis-origin	
	Requirement	/req/openflight/model-dis-origin
	Test purpose	Verify that the CDB Point representing the DIS Origin is positioned and oriented according the definition provided by the DIS Standard. This definition says that the DIS Origin is at the center of the bounding box of the entity, without articulated and attached parts. The standard also says what the orientation SHALL be. The X-axis points forward, the Y-axis points to the right, and the Z-axis points down. All axes are aligned with the bounding box defined above.
	Test method	Visual. Pass if the dis origin is correctly encoded.
	Test type	Conformance

Test 25	/conf/ model-points /model-viewpoint	
	Requirement	/req/openflight/model-viewpoint
	Test purpose	Verify the viewpoint's local coordinate system is oriented such that the Y-axis indicates the viewing direction and the Z-axis points up.
	Test method	Visual inspection. Pass if orientation is correct.
	Test type	Conformance

A.6. Conformance Class: Model Level Of Detail

A levels-of-detail model structure is essential when the intent is to use a model in a real-time application such as flight simulation. The level-of-detail mechanism provides client-devices with the essential structure for deterministic operation. The following test determines whether the OpenFlight model and CDB data store use the proper method to determine if a level of detail is still active.

Conformance Class	/conf/openflight/model-level-of-detail	
Requirements Class	/req/openflight/significant-sizes	
Dependency	Openflight Specification	
Test 26	/conf/model-level-of-detail/significant-sizes	
	Requirement	/req/openflight/significant-sizes
	Test purpose	Verify that the implementation method uses the Significant Size associated with the LOD node to determine when to activate the node
	Test method	Visual.
	Test type	Conformance

A.7. Conformance Class: Model Switch Nodes

Conformance Class	/conf/openflight/model-switch-nodes	
Requirements Class	/req/openflight/switch-mask	
Dependency	Openflight Specification	
Test 27	/conf/model-switch-nodes/switch-mask	

	Requirement	/req/core/openflight/switch-mask
	Test purpose	Verify that the switch contains one mask per state. Note that the first mask, mask index 0, is the default mask. This means that the value of the Current Mask field in the Switch record is 0.
	Test method	Visual. Pass if values are correct.
	Test type	Conformance
Test 28	/conf/ model-switch-nodes/switch-mask-name	
	Requirement	/req/openflight/ switch-mask-name
	Test purpose	Verify that each mask has a name. The name of the mask <i>SHALL</i> be representative of the state selected by that mask.
	Test method	Visual. Pass if each mask has a name
	Test type	Conformance

A.8. Conformance Class: Damage Status

Conformance Class	/conf/openflight/model-switch-nodes	
Requirements Class	/req/openflight/damage-status	
Dependency	Openflight Specification	
Test 29	/conf/ damage-status /switch-mask	
	Requirement	/req/core/openflight/damage-transition
	Test purpose	Verify that the percentages representing the transitions are limited to the range [0, 99]. The value 100 is not allowed because the level of damage cannot exceed the transition value in order to select the correct state.
	Test method	Visual. Pass if values are in valid range.

	Test type	Conformance
Test 30	/conf/ damage-status /switch-mask-name	
	Requirement	/req/openflight/ damage-order
	Test purpose	Verify that the ordering of damage states is from left (normal state) to right (destroyed state). All intermediate states must represent increasingly damaged states from a slightly damaged state to an almost destroyed state.
	Test method	Visual. Pass if ordering is correct.
	Test type	Conformance

A.9. Conformance Class: Model Articulations

Conformance Class	/conf/openflight/model-points	
Requirements Class	/req/openflight/model-articulations	
Dependency	Openflight Specification	
Test 32	/conf/model-articulations/articulation-node-rotation	
	Requirement	/req/core/openflight/model-point-damage-states
	Test purpose	Verify that the translation rate is expressed in meters per second. Verify that the rotation rate is expressed in degrees per second. Finally, verify that the scaling rate is expressed in units per second. When not specified, verify that a default rate of zero is used.
	Test method	Visual. Pass if units of measure are correct.
	Test type	Conformance
Test 33	/conf/model-articulations /gimbal-limits	
	Requirement	/req/openflight/global-limits

	Test purpose	Verify that the appropriate flags are set to specify which degrees of freedom are controlled by a particular articulation.
	Test method	Visual. Pass if flags are correctly set.
	Test type	Conformance
Test 34	/conf/ model-articulations /articulation-flags	
	Requirement	/req/openflight/articulation-flags
	Test purpose	Verify the Flags field, located at offset 376 in the OpenFlight DOF record, has a non-zero value because the articulation control has at least one degree of freedom.
	Test method	Visual inspection. Pass if orientation is correct.
	Test type	Conformance

A.10. Conformance Class: Model Textures

Conformance Class	/conf/openflight/model-textures	
Requirements Class	/req/openflight/model-textures	
Dependency	Openflight Specification	
Test 37	/conf/model-textures/texture-file-loading	
	Requirement	/req/core/openflight/ texture-file-loading
	Test purpose	Verify that for the case of GTModels and MModels, textures are separate files that are loaded after the model geometry files are read and loaded by client devices; in the case of GSModels and T2DModels, the textures can be loaded concurrently with the model geometry files.
	Test method	Visual. Pass if units of measure are correct.
	Test type	Conformance

Test 38	/conf/model- textures/quarterly-textures	
	Requirement	/req/openflight/quarterly-textures
	Test purpose	Verify that the quarterly textures are provided as a complete set, i.e., it is assumed that all 4 textures of the same kind (i.e., all four textures have their component selector 1 set to 003) and are all present in the model's texture directory
	Test method	Visual. Pass if flags are correctly set.
	Test type	Conformance
Test 39	/conf/ model- textures/monthly-textures	
	Requirement	/req/openflight/monthly-textures
	Test purpose	Verify the monthly textures are provided as a complete set, i.e., it is assumed that all the 12 textures are of the same kind (i.e., all twelve textures have their component selector CS1 = 002) and are all present in the model's texture directory.
	Test method	Visual inspection. Pass if orientation is correct.
	Test type	Conformance
Test 40	/conf/ model- textures/texture-mipmap	
	Requirement	/req/openflight/texture-mipmap
	Test purpose	Verify that the mipmaps associated with a given texture are present in the texture directory. Furthermore, verify that the mipmaps are stored in individual files..
	Test method	Visual inspection. Pass if orientation is correct.
	Test type	Conformance
Test 41	/conf/ model- textures/texture-palette-path	

	Requirement	/req/openflight/texture-palette-path
	Test purpose	Verify that the palette path is relative to the OpenFlight file
	Test method	Visual inspection. Pass if orientation is correct.
	Test type	Conformance
Test 42	Requirement	
	/conf/ model- textures/texture-shadow-geometry	
	Requirement	/req/openflight/texture-shadow-geometry
	Test purpose	Verify that when a geometry exists for the purpose of casting shadows that the geometry is located under an OpenFlight object node whose Shadow flag is set
	Test method	Visual inspection. Pass if orientation is correct.
	Test type	Conformance
Test 43	Requirement	
	/conf/ model- textures/model-skin-textures	
	Requirement	/req/openflight/model-skin-textures
	Test purpose	Verify that the mapping of textures for a given model or cultural features is identical since only the texture is changed, not the UV mapping
	Test method	Visual inspection. Pass if orientation is correct.
	Test type	Conformance
Test 44	Requirement	
	/conf/ model- textures/model-night-maps	
	Requirement	/req/openflight/model-night-maps

	Test purpose	Verify that a night map has the same size as its base texture, that a night map uses the same UV mapping as its base texture and that a night map has a similar format as its base texture (RGB or Intensity) plus an alpha channel
	Test method	Visual inspection. Pass if orientation is correct.
	Test type	Conformance
Test 45	/conf/ model- textures/model-night-generation	
	Requirement	/req/openflight/model-night-generation
	Test purpose	Verify that the Effect field of the multitexture record contains the value 0 indicating using the Texture Environment mapping defined in the Texture Attribute file. Verify that the Environment Type field found in the Texture Attribute file contains the value 2 indicating a Decal environment mapping
	Test method	Visual inspection. Pass if orientation is correct.
	Test type	Conformance

Annex B: OpenFlight v16.0 Technical Description – Annotated

This document has been annotated to reflect the conventions established by the CDB standard. Collectively, these conventions are referred to as OpenFlight/CDB. The conventions define how OpenFlight files are interpreted by a CDB-compliant OpenFlight reader; the stated conventions supersede or replace related aspects of this annotated specification. Unless stated otherwise, CDB-compliant OpenFlight readers will ignore any data that fails to conform to the stated conventions.

OpenFlight®

Scene Description

Database Specification

Annotated with CDB conventions

Version 16.0

Document Revision A

November 2004



MultiGen-Paradigm
VISUALIZE REALITY

B.1. Contents

[OpenFlight Concepts](#)

[Database Hierarchy](#)

[Node Attributes](#)

[Palettes](#)

[Instancing](#)

[Replication](#)

[Bounding Volumes](#)

[Multitexture](#)

[OpenFlight Record Types](#)

Control Records

Hierarchy Level Change Records

Push Level Record

Pop Level Record

Push Subface Record

Pop Subface Record

Push Extension Record

Pop Extension Record

Push Attribute Record

Pop Attribute Record

Hierarchy Instancing Records

Instance Definition Record

Instance Reference Record

Node Primary Records

Header Record

Group Record

Object Record

Face Record

Mesh Nodes

Mesh Record

Local Vertex Pool Record

Mesh Primitive Record

Light Point Nodes

Indexed Light Point Record

Light Point Record

Light Point System Record

Degree of Freedom Record

[Vertex List Record](#)

[Morph Vertex List Record](#)

[Binary Separating Plane Record](#)

[External Reference Record](#)

[Level of Detail Record](#)

[Sound Record](#)

[Light Source Record](#)

[Road Segment Record](#)

[Road Construction Record](#)

[Clip Region Record](#)

[Text Record](#)

[Switch Record](#)

[CAT Record](#)

[Extension Record](#)

[Curve Record](#)

[Ancillary Records](#)

[Comment Record](#)

[Long ID Record](#)

[Indexed String Record](#)

[Multitexture](#)

[Multitexture Record](#)

[UV List Record](#)

[Replicate Record](#)

[Road Zone Record](#)

[Transformation Records](#)

[Matrix Record](#)

[Rotate About Edge Record](#)

[Translate Record](#)

[Scale Record](#)

[Rotate and/or Scale to Point Record](#)

[Put Record](#)

[General Matrix Record](#)

[Rotate About Point Record](#)

[Vector Record](#)

[Bounding Volume Records](#)

[Bounding Box Record](#)

[Bounding Sphere Record](#)

[Bounding Cylinder Record](#)

[Bounding Convex Hull Record](#)

[Bounding Histogram Record](#)

[Bounding Volume Center Record](#)

[Bounding Volume Orientation Record](#)

[CAT Data Record](#)

[Extension Attribute Record](#)

[Continuation Record](#)

[Palette Records](#)

[Vertex Palette Records](#)

[Vertex Palette Record](#)

[Vertex with Color Record](#)

[Vertex with Color and Normal Record](#)

[Vertex with Color and UV Record](#)

[NormalandUVRecord](#)

[Color Palette Record](#)

[Name Table Record](#)

[Material Palette Record](#)

[Texture Palette Record](#)

[Eyepoint and Trackplane Palette Record](#)

[Key Table Records](#)

[Linkage Palette Record](#)

[Sound Palette Record](#)

[Light Source Palette Record](#)

[Light Point Appearance Palette Record](#)

[Light Point Animation Palette Record](#)

[Line Style Palette Record](#)

[Texture Mapping Palette Record](#)

[Texture Pattern Files](#)

[Texture Attribute Files](#)

[Vertex Node Parameters](#)

[Face Node Parameters](#)

[Object Node Parameters](#)

[LOD Node Parameters](#)

[Group Node Parameters](#)

[DOF Node Parameters](#)

[Sound Node Parameters](#)

[Switch Node Parameters](#)

[Text Node Parameters](#)

[Light Source Node Parameters](#)

[Clip Node Parameters](#)

[Valid Opcodes](#)

[Obsolete Opcodes](#)

[C - Overview](#)

[Format Changes](#)

[Continuation Record](#)

[Header Record](#)

[Mesh Nodes](#)

[Mesh Record](#)

[Local Vertex Pool Record](#)

[Mesh Primitive Record](#)

[Multitexture](#)

[Multitexture Record](#)

[UV List Record](#)

[Texture Attribute File](#)

[Subtexture](#)

[D - Overview](#)

[Document Corrections](#)

[Text Record](#)

[CAT Record](#)

[Header Record](#)

[Group Record](#)

[Level of Detail Record](#)

[External Reference Record](#)

[Indexed String Record](#)

[Face Record](#)

[Mesh Record](#)

[Local Vertex Pool Record](#)

[Vertex Palette Records](#)

[Light Points](#)

[Light Point Appearance Palette Record](#)

[Light Point Animation Record](#)

[Indexed Light Point Record](#)

[Light Point System Record](#)

[Texture Mapping Palette Record](#)

[Parameters for 3 Point Put Texture Mapping \(Type 1\)](#)

[Parameters for 4 Point Put Texture Mapping \(Type 2\)](#)

[E - Overview](#)

[Document Corrections](#)

[Header Record](#)

[Face Record](#)

[Mesh Record](#)

[Switch Record](#)

[Texture Mapping Palette Record](#)

[Indexed String Record](#)

[Bounding Convex Hull Record](#)

[Bounding Histogram Record](#)

[Format Changes](#)

[External Reference Record](#)

[Face Record](#)

[Mesh Record](#)

[Light Point Appearance Palette Record](#)

[Shader Palette Record](#)

[Texture Attribute File](#)

[Texture Mapping Palette Record](#)

[Parameters for 3 Point Put Texture Mapping \(Type 1\)](#)

[OpenFlight Scene Description Database Specification, version 16.0. November, 2004](#)

B.2. OpenFlight® Scene Description

The following symbols have been used throughout the document to specify the conventions established by OpenFlight/CDB.

□= The record, field or value is supported by OpenFlight/CDB readers and follows the same conventions and usage as the OpenFlight Standard

□= The record, field is not considered by OpenFlight/CDB readers (e.g. ignored)

□= The record, field or value is specific to MultiGen-Paradigm and therefore is not considered by OpenFlight/CDB readers (e.g. ignored)

□= The value for the specified field is not supported by OpenFlight/CDB readers. OpenFlight/CDB readers ignore any fields with values that are not supported.

□= The record, field or value is specific to MultiGen's Creator tool and therefore is not considered by OpenFlight/CDB readers (e.g. ignored)

The primary audience for this document includes software developers whose applications are intended to read and/or write OpenFlight database files. To this end, this document discusses concepts incorporated in OpenFlight and contains a detailed description of the physical layout of OpenFlight files as represented on disk.

B.3. OpenFlight Concepts

The OpenFlight database format supports both simple and relatively sophisticated real-time software applications. The full implementation of OpenFlight supports variable levels of detail, degrees of freedom, sound, instancing (both within a file and to external files), replication, animation sequences, bounding volumes for real-time culling, scene lighting features, light points and light point strings, transparency, texture mapping, material properties, and many other features.

A simple application that interprets an OpenFlight database can implement a subset of the database specification and use databases that contain that subset. Such an application could simply scan for the color palette, faces, and vertices, and ignores groups, objects, and other more sophisticated features.

B.4. Database Hierarchy

The OpenFlight database hierarchy organizes the visual database into logical groupings and facilitates real-time functions such as field-of-view culling, level-of-detail switching, and instancing. Each OpenFlight database is organized in a tree structure.

The database tree structure consists of nodes (historically called beads). Most nodes can have child nodes as well as sibling nodes. In general, nodes can be thought of in three hierarchical classes. Starting from the top of the hierarchy, these three node classes include container nodes, geometry nodes and vertex nodes.

Geometry nodes are nodes that actually represent some physical (renderable) geometry. The at-

tributes of geometry nodes typically include visual attributes such as color, material, texture, etc. The two main geometry nodes in OpenFlight are the face and mesh nodes. Other geometry nodes include the light point and text node. Though OpenFlight allows it, there are very few cases in which at least one geometry node is not contained somewhere below a container node.

Each node type has data attributes specific to its function in the database. The principal node types in OpenFlight are described here:

Group: A group node distinguishes a logical subset of the database. Group nodes can be transformed (translated, rotated, scaled, etc.). The transformation applies to itself and to all its children. Groups can have child nodes and sibling nodes of any type, except a header node. For more information, see [Group Record](#).

Object: An object node contains a logical collection of geometry. It is effectively a low-level group node that offers some attributes distinct from the group node. For more information, see [Object Record](#).

Face: A face node represents geometry. Its children are limited to a set of vertices that describe a polygon, line, or point. For a polygon, the front side of the face is viewed from an in-order traversal of the vertices. Face attributes include color, texture, material, and transparency. For more information, see [Face Record](#).

Mesh: A mesh node defines geometric primitives that share attributes and vertices. See For more information, see [Mesh Nodes](#).

Light point: A light point node represents a collection of light point vertices or a replicated string of a single light point vertex. A light point is visible as one or more self-illuminated small points that do not illuminate surrounding objects. For more information, see [Light Point Nodes](#).

Light point system: A light point system enables you to collect a set of light points and enable/disable or brighten/dim them as a group. For more information, see [Light Point System Record](#).

Light source: A light source node serves as the location and orientation of a light source. The light source position and direction are transformed by the transformations above it in the tree (if any). For more information, see [Light Source Record](#).

Sound: A sound node serves as the location for a sound emitter. The emitter position is the sound offset transformed by the transformations above it in the tree (if any). For more information, see [Sound Record](#).

Text: A text node draws text in a string with a specified font, without injecting the actual geometry into the database as face nodes. This is a leaf node and therefore cannot have any children. For more information, see [Text Record](#).

Vertex: A vertex node represents a point in space, expressed as a double precision 3D coordinates. Each vertex is stored in the vertex palette record. Vertex attributes include x, y, z and optionally include color, normal and texture mapping information. Vertex nodes are the children of face nodes and light point nodes. For more information, see [Vertex List Record](#), [Morph Vertex List Record](#) and [Vertex Palette Records](#).

Morph vertex: A morph vertex node is a second vertex node. The vertex and morph vertex represent the two endpoints of a path between which the actual vertex may be interpolated. One endpoint represents the minimum (non morphed) weighting and the other represents the maximum (fully morphed) weighting. Each endpoint (or weight) is a reference into the vertex palette record. All vertex attributes may be morphed. Morph vertex nodes are the children of face nodes. For more information, see [Morph Vertex List Record](#).

Clip region: A clip node defines a set of clipping planes. Any geometry, of the clip node's children, that falls outside the specified clipping planes is not displayed. For more information, see [Clip Region Record](#).

Degree of freedom: A degree of freedom (DOF) node serves as a local coordinate system with a predefined set of internal transformations. It specifies the articulation of parts in the database and set limits on the motion of those parts. For more information, see [Degree of Freedom Record](#).

Level of detail: A level of detail (LOD) node serves as a switch to turn the display of everything below it on or off based on its range from the viewer, according to its switch-in, switch-out distance and center location. For more information, see [Level of Detail Record](#).

Switch: A switch node is a more general case of an LOD node. It allows the selection of zero or more children by invoking a selector mask. Any combination of children can be selected per mask and the number of definable masks is unlimited. For more information, see [Switch Record](#).

External reference: An external reference node serves to reference a node in another database file, or an entire database file. The referenced (child) node or database is considered an external part of the referencing (parent) database. For more information, see [External Reference Record](#).

B.5. Node Attributes

Nodes in the OpenFlight scene contain attributes whose values describe different properties or characteristics of the node. Most attributes are represented directly on the node itself and are geared toward describing the specific characteristics of that type of node. The level of detail (LOD) node, for example, defines a switch in and switch out distance. Used together, these distances define a range within which the geometry contained in the LOD is visible.

Other attributes are represented indirectly on a node, using a lookup index into a table (palette) of attributes to describe the characteristics of a node. The face node, for example, defines several indirect attributes, including color index, material index and texture index. The values of these index attributes are used to map specific colors, materials and textures to the face node. The definitions of the colors, materials and textures referenced by these index attributes are stored in palettes in the database rather than directly on the nodes themselves.

This mechanism of indirect attribute mapping via palettes has some advantages. It can both save space in the OpenFlight file and can simplify the task of making global changes to nodes in the database.

To see how this indirection saves space, consider the material index attribute on the face node. A material is defined by over 15 separate color and other visual attributes. If each of these attributes were maintained per face in the database, the size of the database would get large quickly. Since it

is common to map a single material to hundreds (or even thousands) of faces in the database, it is much more efficient to store a single material index attribute per face rather than storing the entire material definition.

Also, in terms of changing the appearance of a particular material in your database, when you do change the material definition in the palette, the faces that reference that material get updated automatically. This can make global changes much more simple to accomplish.

B.6. Palettes

In the previous section, indirect attribute mapping was introduced. As part of that discussion, the notion of database palettes was also mentioned briefly. In fact, indirect attribute mapping is not possible without a robust implementation of database palettes. A database palette is a collection (or set) of attribute definitions. As mentioned in the previous section, the material palette defines a set of materials, each material being composed of several different color and visual attributes.

The OpenFlight database supports many different palettes. The most obvious palettes are the color, material and texture palettes. Most palettes support variable numbers of elements while others enforce fixed size constraints. The material and texture palettes are both variable sized palettes that can contain zero or more entries. The color palette, in contrast, is a fixed size palette that contains exactly 1024 entries.

All the database palettes supported by OpenFlight are described in [Palette Records](#). Specific palettes in OpenFlight include:

- "Color Palette Record"
- "Material Palette Record"
- "Texture Palette Record"
- "Texture Mapping Palette Record"
- "Sound Palette Record"
- "Line Style Palette Record"
- "Light Source Palette Record"
- "Light Point Appearance Palette Record"
- "Light Point Animation Palette Record"
- "Vertex Palette Records"
- "Name Table Record"
- "Eyepoint and Trackplane Palette Record"
- "Linkage Palette Record"

B.7. Instancing

Instancing is the ability to define all or part of a database once, then reference it one or more times while applying various transformations. This allows you to define a piece of geometry once and place it multiple times in the scene. OpenFlight supports internal and external instancing with operations such as Rotate, Translate, Scale, and Put.

An internal instance is a subtree of the database that has been declared as an instance definition. An instance definition represents the root of a stand-alone subtree within the database. It is introduced by an instance definition record that contains a unique instance definition number. An instance definition is invoked by an instance reference record in a subsequent part of the database tree.

An external instance refers to an entire database file. It is introduced by an external reference node. An external reference node contains the name of the (child) database file to attach to that point in the referencing (parent) database tree. It also includes attributes that determine whether the child uses its own color, material, and texture palettes, or those of its parent.

Instance definitions can themselves contain instance definitions and references. Internal instances cannot reference themselves. External instances should not reference themselves directly or indirectly. The result of such use is undefined.

Instance definition and instance reference records are described in [Hierarchy Instancing Records](#). External reference records are described in [External Reference Record](#).

B.8. Replication

Replication instances a subtree of the database several times, applying a transformation each time. For example, a string of trees can be represented by a single group node that is instantiated and translated to a new position several times.

Replication is legal for group, face, and light point nodes. Therefore a replication record is an ancillary record of a group, face, or light point node. In conjunction with a replication record there will be one or more ancillary transformation records.

B.9. Bounding Volumes

Bounding volumes can be used by the application to determine if a particular subtree of the database is in view. A bounding volume can be a box, a sphere, or a cylinder. Each group node can have only one bounding volume. The volume normally encompasses the full geometric extent of the group node's children, including any instances and replications. A bounding volume record is an ancillary record of a group node.

B.10. Multitexture

OpenFlight supports eight textures per polygon or mesh as well as eight uv values per vertex. The texture information stored directly on the face, mesh and vertex record is referred to as “the base texture” or “texture layer 0”. Each additional texture layer is stored in ancillary records to the face,

mesh and vertex list records and is referred to as “texture layer N” (for N=1..7). See [Multitexture](#) for more information. OpenFlight supports eight textures per polygon or mesh as well as eight uv values per vertex. The texture information stored directly on the face, mesh and vertex record is referred to as “the base texture” or “texture layer 0”. Each additional texture layer is stored in ancillary records to the face, mesh and vertex list records and is referred to as “texture layer N” (for N=1..7). See [Multitexture](#) for more information.

B.11. OpenFlight File Format

The hierarchical structure of an OpenFlight database is stored on disk as a file. The file consists of a linear stream of binary records. Byte ordering in the file is big endian. All OpenFlight records begin with a 4 byte sequence. The first two bytes of this sequence identifies the record type (opcode) and the second two bytes specify the length of the record. Note that the length includes this 4 byte sequence so the minimum length of any record (that does not contain any additional data) will be 4. Given this very regular structure, OpenFlight records can be read from disk and parsed easily.

- All OpenFlight records are a multiple of 4 bytes in length. When a record contains less than a full multiple of 4 bytes of data, the record is padded up (bytes added to the end of the record) to be a multiple of 4 bytes in length. In some cases, OpenFlight records are padded up to be multiples of 8 bytes in length.
- The length of all records (and fields in all records) as well as the offset of all fields are expressed in bytes.
- Unless explicitly stated otherwise, bit fields and masks are counted starting at 0 (i.e., the first bit is bit number 0).
- Unless explicitly stated otherwise, the elements of matrix records stored in OpenFlight appear in row major order. That is, the elements of the matrix appear in the following order:
row0col0, row0col1, row0col2, row0col3,
row1col0, row1col1, row1col2, row1col3,
row2col0, row2col1, row2col2, row2col3,
row3col0, row3col1, row3col2, row3col3
- The length of all OpenFlight records is limited to the largest value that can be encoded with 2 bytes or 16 bits (65535). For fixed-size records, this maximum size is sufficient. For variable-size records, this limitation is addressed with the Continuation Record. For more information, see [Continuation Record](#).

B.12. OpenFlight Record Types

There are four major categories of records: control records, node primary records, ancillary records and continuation records.

Control records mark the hierarchy of the tree. A push control record (a record containing the push opcode) indicates an increase in the depth of the tree. A push control record drops you down one level in the tree. A pop control record (a record containing a pop opcode) returns you to the previous level of hierarchy. All records between a push and a pop represent sibling nodes at the

same level of hierarchy. Other control records include: instance definition, instance reference, push subface, pop subface, push attribute, and pop attribute.

Each node is represented on disk by one primary record and zero or more ancillary records. The primary record identifies a node type and includes most of the node attribute data. Additional node attributes, such as comments, long ID, and transformations, are stored in subsequent ancillary records. Ancillary records follow the primary record, but precede any control records. Child nodes are introduced by a push control record and are concluded by a pop control record.

Palette records are ancillary records of the header node. Palette records generally follow the header node's primary record, with the exception of behavior (linkage) palette records. Behavior palette records, if present, are the last (non-control) records in the file.

Many records include an eight character ASCII ID consisting of the first seven characters of the node name plus a terminating <nil> character. If the node ID is longer than seven characters, an ancillary long ID record containing the complete ID follows the node primary record.

For example, a record with an object opcode is followed by a push control record. Next comes a record with a face opcode, also followed by a push control record. After that comes the vertex list record(s) that describe the vertices of the face, and then a pop control record. This, in turn, may be followed by another face record for the next face in the same object, or by a pop record to return to object level.

B.12.1. Control Records

Control records indicate a change in the level of the database hierarchy. The three basic types of control records are: level changes, instance definition, and instance reference. Level changes are indicated by push and pop control records. Instance definitions and references are indicated by instance definition and instance reference control records.

Hierarchy Level Change Records

A database contains three distinct types of hierarchy: generic, subface, and attribute. Hierarchy may be skipped by scanning past the push control record for the corresponding pop control record.

Generic A push level control record introduces a generic subtree of the database hierarchy. A pop level control record concludes that subtree.

Subface A push subface control record introduces a subtree of coplanar faces. A pop subface control record concludes that subtree.

Extension A push extension control record introduces a subtree of user defined records. A pop extension control records concludes that subtree.

Attribute A push attribute control record introduces a subtree of records reserved for internal use by MultiGen-Paradigm, Inc.. A pop attribute control record concludes that subtree.

Push Level Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Push Level Opcode 10	□
Unsigned Int	2	2	Length - length of the record	□

Pop Level Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Pop Level Opcode 11	□
Unsigned Int	2	2	Length - length of the record	□

Push Subface Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Push Subface Opcode 19	□
Unsigned Int	2	2	Length - length of the record	□

Pop Subface Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Pop Subface Opcode 20	□
Unsigned Int	2	2	Length - length of the record	□

Push Extension Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Push Extension Opcode 21	□
Unsigned Int	2	2	Length - length of the record	□
Char	4	18	Reserved	□

Unsigned Int	22	2	Vertex reference index; -1 if not vertex extension	🔗
--------------	----	---	--	-------------------

Pop Extension Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Pop Extension Opcode 22	🔗
Unsigned Int	2	2	Length - length of the record	🔗
Char	4	18	Reserved	🔗
Unsigned Int	22	2	Vertex reference index; -1 if not vertex extension	🔗

Push Attribute Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Push Attribute Opcode 122	🔗
Unsigned Int	2	2	Length - length of the record	🔗
Int	4	4	Vertex reference index; -1 if not vertex attribute	🔗

Pop Attribute Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Pop Attribute Opcode 123	🔗
Unsigned Int	2	2	Length - length of the record	🔗

Hierarchy Instancing Records

An instance definition record introduces a stand-alone subtree of the database. The subtree is referenced one or more times from different branches in the database by instance reference records. At the point of reference, the subtree is copied (or possibly shared) as a child of the current parent node.

The instance definition record must appear in the file stream prior to the first instance reference record that references it. A typical usage of these records might look like:

INSTANCE DEFINITION 1

PUSH

The records between this PUSH and POP define the stand-alone subtree that is INSTANCE DEFINITION 1

POP

...

GROUP

MATRIX

PUSH

INSTANCE REFERENCE 1

POP

GROUP

MATRIX

PUSH

INSTANCE REFERENCE 1

POP

In this example, both groups reference instance definition number 1, each presumably applying a different matrix to place the instance in different locations in the scene.

Instance Definition Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Instance Definition Opcode 62	□
Unsigned Int	2	2	Length - length of the record	□
Int	4	2	Reserved	□
Int	6	2	Instance definition number	□

Instance Reference Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Instance Reference Opcode 61	□
Unsigned Int	2	2	Length - length of the record	□

Int	4	2	Reserved	□
Int	6	2	Instance definition number	□

B.12.2. Node Primary Records

Header Record

The header record is the primary record of the header node and is always the first record in the database file. Attributes within the header record provide important information about the database file as a whole.

Format revision level indicates the OpenFlight version of the file. Correctly interpreting the attributes of other records, such as the face and vertex records, depends upon the format revision. The format revision encompasses both Flight and OpenFlight versions.

Some representative values for format revision are:

Format Revision Value	Flight/OpenFlight Version	CDB OpenFlight Reader
11	Flight V11	□
12	Flight V12	□
14	OpenFlight v14.0 and v14.1	□
1420	OpenFlight v14.2	□
1510	OpenFlight v15.1	□
1540	OpenFlight v15.4	□
1550	OpenFlight v15.5	□
1560	OpenFlight v15.6	□
1570	OpenFlight v15.7	□
1580	OpenFlight v15.8	□
1600	OpenFlight v16.0	□

This document describes OpenFlight version 16.0, therefore the attribute descriptions are based upon a format revision level of 1600.

Geographic attributes such as projection type, latitude, and longitude may be stored in the header record. The MultiGen Series II and Creator Terrain options set the value of these attributes when creating terrain databases. Positive latitudes reference the northern hemisphere and negative longitudes reference the western hemisphere.

Delta x, y and z attributes indicate the placement of the database when several separate databases, each with a local origin of zero, are used to represent an area.

Table 2. Header Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Header Opcode 1	□
Unsigned Int	2	2	Length - length of the record	□
Int	12	4	Format revision level	□
Int	16	4	Edit revision level	□
Char	20	32	Date and time of last revision	□
Int	52	2	Next Group node ID number	□
Int	54	2	Next LOD node ID number	□
Int	56	2	Next Object node ID number	□
Int	58	2	Next Face node ID number	□
Int	60	2	Unit multiplier (always 1)	□
Int	62	1	Vertex coordinate units	□
			0 = Meters	□
			1 = Kilometers	□
			4 = Feet	□
			5 = Inches	□
			8 = Nautical miles	□
Int	63	1	if TRUE set texwhite on new faces	□
Int	64	4	Flags (bits, from left to right)	□
			0 = Save vertex normals	□
			1 = Packed Color mode	□
			2 = CAD View mode	□
			3-31 = Spare	□

Int	68	4*6	Reserved	□
Int	92	4	Projection type	□
			0 = Flat earth	□
			1 = Trapezoidal	□
			2 = Round earth	□
			3 = Lambert	□
			4 = UTM	□
			5 = Geodetic	□
			6 = Geocentric	□
Int	96	4*7	Reserved	□
Int	124	2	Next DOF node ID number	□
Int	126	2	Vertex storage type	□
			1 = Double precision float - should always be 1	□

Table 3. Header Record (Continued)

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	128	4	Database origin	□
			100 = OpenFlight	□
			200 = DIG I/DIG II	□
			300 = Evans and Sutherland CT5A/CT6	□
			400 = PSP DIG	□
			600 = General Electric CIV/CV/PT2000	□
			700 = Evans and Sutherland GDF	□
Double	132	8	Southwest database coordinate x	□
Double	140	8	Southwest database coordinate y	□

Double	148	8	Delta x to place database	□
Double	156	8	Delta y to place database	□
Int	164	2	Next sound node ID number	□
Int	166	2	Next path node ID number	□
Int	168	4*2	Reserved	□
Int	176	2	Next Clip node ID number	□
Int	178	2	Next Text node ID number	□
Int	180	2	Next BSP node ID number	□
Int	182	2	Next Switch node ID number	□
Int	184	4	Reserved	□
Double	188	8	Southwest corner latitude	□
Double	196	8	Southwest corner longitude	□
Double	204	8	Northeast corner latitude	□
Double	212	8	Northeast corner longitude	□
Double	220	8	Origin latitude	□
Double	228	8	Origin longitude	□
Double	236	8	Lambert upper latitude	□
Double	244	8	Lambert lower latitude	□
Int	252	2	Next Light source node ID number	□
Int	254	2	Next Light point node ID number	□
Int	256	2	Next Road node ID number	□

Int	258	2	Next CAT node ID number	□
Int	260	2	Reserved	□
Int	262	2	Reserved	□
Int	264	2	Reserved	□
Int	266	2	Reserved	□
Int	268	4	Earth ellipsoid model	□
			0 = WGS 1984	□
			1 = WGS 1972	□
			2 = Bessel	□
			3 = Clarke 1866	□
			4 = NAD 1927	□
			-1 = User defined ellipsoid	□
Int	272	2	Next Adaptive node ID number	□
Int	274	2	Next Curve node ID number	□
Int	276	2	UTM zone (for UTM projections - negative value means Southern hemisphere)	□
Char	278	6	Reserved	□

Header Record (Continued)

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Double	284	8	Delta z to place database (used in conjunction with existing Delta x and Delta y values)	□
Double	292	8	Radius (distance from database origin to farthest corner)	□

Unsigned int	300	2	Next Mesh node ID number	□
Unsigned int	302	2	Next Light Point System ID number	□
Int	304	4	Reserved	□
Double	308	8	Earth major axis (for user defined ellipsoid) in meters	□
Double	316	8	Earth minor axis (for user defined ellipsoid) in meters	□

Group Record

The group record is the primary record of the group node. Groups are the most generic hierarchical node present in the database tree. Attributes within the group record provide bounding volumes that encompass the group's children and real-time control flags.

Relative priority specifies a fixed ordering of the group relative to its sibling nodes. Ordering is from left (lesser values) to right (higher values). Nodes of equal priority may be arbitrarily ordered. All nodes have an implicit (default) relative priority value of zero.

A group can represent an animation sequence in which case each immediate child of the group represents one frame of the sequence. An animation sequence is made of one or more loops.

For a group with N children, both forward and backward loops consist of N frames. The frames of forward and backward loops are:

Direction	Frame 1	Frame 2	Frame 3	...	Frame N
Forward	Child 1	Child 2	Child 3	...	Child N
Backward	Child N	Child N-1	Child N-2	...	Child 1

Independent of the direction of the loop, a loop can optionally *swing*. A swing loop is one that plays its children in the primary direction and then plays them in the opposite direction. Note that as the loop swings from the current direction to the opposite direction, the last frame in the current direction is not repeated. Therefore, for a group with N children, the first loop of both forward swing and backward swing animations

consist of M frames where M equals $((2*N)-1)$ frames. Subsequent loops of swing animations consist of M-1 frames. The frames of the first loop of forward and backward swing animations are:

Direction	Frame 1	Frame 2	...	Frame N	Frame N+1	Frame N+2	...	Frame M
Forward	Child 1	Child 2	...	Child N	Child N-1	Child N-2	...	Child 1

Backward	Child N	Child N-1	...	Child 1	Child 2	Child 3	...	Child N
----------	---------	-----------	-----	---------	---------	---------	-----	---------

The frames of subsequent loops of forward and backward swing animations are:

Direction	Frame 1	Frame 2	...	Frame N	Frame N+1	Frame N+2	...	Frame M-1
Forward	Child 2	Child 3	...	Child N	Child N-1	Child N-2	...	Child 1
Backward	Child N-1	Child N-2	...	Child 1	Child 2	Child 3	...	Child N

The number of times an animation loop repeats within the sequence is specified by the loop count attribute. A loop count of 0 indicates that the loop is to repeat forever.

The duration of one loop within the sequence is specified by the loop duration attribute and is measured in seconds. A loop duration of 0 indicates that the loop is to play as fast as possible.

For finite animation sequences (those with positive, non-zero loop count values), the duration that the last frame of the last loop is extended after the sequence has finish is specified by the last frame duration attribute and is measured in seconds. A last frame duration of 0 indicates that the last frame is not displayed any longer after the sequence finishes.

Special effect ID1 and ID2 are application-defined attributes. Their values can be used to enhance the meaning of existing attributes, such as the animation flags, or extend the interpretation of the group node. Normally, the value of these attributes is zero.

Significance can be used to assist real-time culling and load balancing mechanisms, by defining the visual significance of this group with respect to other groups in the database. Normally the value of this attribute is zero.

Layer ID is used by the Instrumentation Tools in the modeling products to identify (for display) a collection of groups, independent of their locations in the hierarchy. Normally the value of this attribute is zero.

Table 4. Group Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Group Opcode 2	□
Unsigned Int	2	2	Length - length of the record	□
Char	4	8	7 char ASCII ID; 0 terminates	□
Int	12	2	Relative priority	□
Int	14	2	Reserved	□
Int	16	4	Flags (bits, from left to right)	□
			0 = Reserved	□

			1 = Forward animation	□
			2 = Swing animation	□
			3 = Bounding box follows	□
			4 = Freeze bounding box	□
			5 = Default parent	□
			6 = Backward animation	□
			7-31 = Spare	□
Int	20	2	Special effect ID1 - application defined	□
Int	22	2	Special effect ID2 - application defined	□
Int	24	2	Significance	□ per CDB convention
Int	26	1	Layer code	□
Int	27	1	Reserved	□
Int	28	4	Reserved	□
Int	32	4	Loop count	□
Float	36	4	Loop duration in seconds	□
Float	40	4	Last frame duration in seconds	□

Table 5. Group Animation Flags Examples

Forward Animation	Backward Animation	Swing Animation	Result
0	0	0	Group is not animated
1	0	0	Animation loop is forward, no swing.
0	1	0	Animation loop is backward, no swing.

1	0	1	Animation loop is forward with swing.
0	1	1	Animation loop is backward with swing.
1	1	Any	Undefined, must be either forward or backward (not both).

Here are some examples that show how the loop duration, loop count and last frame duration attributes affect the animation. Note that these values are independent of the animation flags from above.

Table 6. Group Animation Count Examples

Loop Duration	Loop Count	Last Frame Duration	Result
0	0	Any	Each loop plays as fast as possible. Loops are played forever. Last Frame Duration not applicable.
T	0	Any	Each loop lasts T seconds. Loops are played forever. Last Frame Duration not applicable.
0	N	0	Each loop plays as fast as possible. N loops are played. Last frame displayed as long as any other frame.
0	N	T	Each loop plays as fast as possible. N loops are played. Last frame of last (Nth) loop displayed T seconds longer than any other frame.

T1	N	0	Each loop lasts T1 seconds. N loops are played. Last frame of last (Nth) loop displayed as long as any other frame.
T1	N	T2	Each loop lasts T1 seconds. N loops are played. Last frame of last (Nth) loop displayed T2 seconds longer than any other frame.

Object Record

The object record is the primary record of the object node. Objects are low-level grouping nodes that contain attributes pertaining to the state of its child geometry. Only face and light point nodes may be the children of object nodes.

The time-of-day object flags can be used to inhibit the display of certain objects, depending on the current time of day.

The illumination flag, when set, makes an object self-illuminating, and is not subject to lighting calculations. In practice, geometric normals should be ignored.

The flat shading flag, when set, indicates that lighting calculations should produce a faceted appearance to the object's geometry. In practice, geometric normals should be constrained to face normals.

The shadow flag indicates the object represents the shadow of the rest of the group. When used as part of a moving model (e.g., an aircraft), the application can apply appropriate distortions, creating a realistic shadow on the terrain or runway.

Relative priority specifies a fixed ordering of the object relative to its sibling nodes. Ordering is from left (lesser values) to right (higher values). Nodes of equal priority may be arbitrarily ordered. All nodes have an implicit (default) value of zero.

When used, transparency applies to all an object's children (geometry). The value should be modulated with the transparency of the geometry and material alpha calculation, as described in the Face Record, Mesh Record and Material Record sections.

NOTE

The MultiGen-Paradigm, Inc. modeling environment does not use the object transparency value for rendering as described above.

However, when an object's transparency value is set in Creator, that value is set on all children faces of the object. Runtime applications may choose to use the transparency value at the object level at their discretion.

Table 7. Object Record

Data Type	Offset	Length	Description	OpenFlight CDB Reader
Int	0	2	Object Opcode 4	□
Unsigned Int	2	2	Length - length of the record	□
Char	4	8	7 char ASCII ID; 0 terminates	□
Int	12	4	Flags (bits from to right)	□
			0 = Don't display in daylight	□
			1 = Don't display at dusk	□
			2 = Don't display at night	□
			3 = Don't illuminate	□
			4 = Flat shaded	□
			5 = Group's shadow object	□
			6-31 = Spare	□
Int	16	2	Relative priority	□
Unsigned Int	18	2	Transparency	□
			0 = Opaque	□
			65535 = Totally clear	□
Int	20	2	Special effect ID1 - application defined	□
Int	22	2	Special effect ID2 - application defined	□
Int	24	2	Significance	□ Per CDB conventions
Int	26	2	Reserved	□

Face Record

The face record is the primary record of the face node. A face contains attributes describing the

visual state of its child vertices. Only vertex and morph vertex nodes may be children of faces. This should not be confused with the fact that faces may have subfaces.

If a face contains a non-negative material index, its apparent color is a combination of the face color and material color, as described in [Material Palette Record](#). If a face contains a nonaddictive material with an alpha component and the transparency field is set, the total transparency is the product of the material alpha and face transparency.

NOTE As mentioned in [Object Record](#), the object transparency is not used in the MultiGen-Paradigm, Inc. modeling environment to determine the actual transparency value of a face.

If a face is a unidirectional or bidirectional light point, the face record is followed by a vector record (Vector Opcode 50) that contains the unit vector indicating the direction in which the primary color is displayed. For bidirectional light points, the alternate color is displayed in the opposite direction (180 degrees opposed).

NOTE This method of defining light points is obsolete after OpenFlight version 15.2. Such light point faces will be turned into the new light point record when it is read into MultiGen II v1.4 or later.

Relative priority specifies a fixed ordering of the face relative to its sibling nodes. Ordering is from left (lesser values) to right (higher values). Nodes of equal priority may be arbitrarily ordered. All nodes have an implicit (default) value of zero.

Table 8. Face Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Face Opcode 5	□
Unsigned Int	2	2	Length - length of the record	□
Char	4	8	7 char ASCII ID; 0 terminates	□
Int	12	4	IR color code	□
Int	16	2	Relative priority	□
Int	18	1	Draw type	□
			0 = Draw solid with backface culling (front side only)	□

			1 = Draw solid, no backface culling (both sides visible)	<input type="checkbox"/>
			2 = Draw wireframe and close	<input type="checkbox"/>
			3 = Draw wireframe	<input type="checkbox"/>
			4 = Surround with wireframe in alternate color	<input type="checkbox"/>
			8 = Omnidirectional light	<input type="checkbox"/>
			9 = Unidirectional light	<input type="checkbox"/>
			10 = Bidirectional light	<input type="checkbox"/>
Int	19	1	Texture white = if TRUE, draw textured face white	<input type="checkbox"/>
Unsigned Int	20	2	Color name index	<input type="checkbox"/>
Unsigned Int	22	2	Alternate color name index	<input type="checkbox"/>
Int	24	1	Reserved	<input type="checkbox"/>
Int	25	1	Template (billboard)	<input type="checkbox"/>
			0 = Fixed, no alpha blending	<input type="checkbox"/>
			1 = Fixed, alpha blending	<input type="checkbox"/>
			2 = Axial rotate with alpha blending	<input type="checkbox"/>
			4 = Point rotate with alpha blending	<input type="checkbox"/>

Int	26	2	Detail texture pattern index, -1 if none	□ Note: Detail textures are IRIS GL specific
Int	28	2	Texture pattern index, -1 if none	□
Int	30	2	Material index, -1 if none	□
Int	32	2	Surface material code (for DFAD)	□ (tentative)
Int	34	2	Feature ID (for DFAD)	□
Int	36	4	IR material code	□
Unsigned Int	40	2	Transparency	□
			0 = Opaque	
			65535 = Totally clear	
Unsigned Int	42	1	LOD generation control	□
Unsigned Int	43	1	Line style index	□
Int	44	4	Flags (bits from left to right)	□
			0 = Terrain	□
			1 = No color	□
			2 = No alternate color	□
			3 = Packed color	□
			4 = Terrain culture cutout (footprint)	□
			5 = Hidden, not drawn	□
			6 = Roofline	□
			7-31 = Spare	□
Unsigned Int	48	1	Light mode	□
			0 = Use face color, not illuminated	□
			1 = Use vertex colors, not illuminated	□

			2 = Use face color and vertex normals	<input type="checkbox"/>
			3 = Use vertex colors and vertex normals	<input type="checkbox"/>
Char	49	7	Reserved	<input type="checkbox"/>
Unsigned Int	56	4	Packed color, primary (a, b, g, r) - only b, g, r used	<input type="checkbox"/>
Unsigned Int	60	4	Packed color, alternate (a, b, g, r) - only b, g, r used	<input type="checkbox"/>
Int	64	2	Texture mapping index	<input type="checkbox"/>
Int	66	2	Reserved	<input type="checkbox"/>
Unsigned Int	68	4	Primary color index	<input type="checkbox"/>
Unsigned Int	72	4	Alternate color index	<input type="checkbox"/>
Int	76	2	Reserved	<input type="checkbox"/>
Int	78	2	Shader index, -1 if none	<input type="checkbox"/>

B.12.3. Mesh Nodes

A mesh node is defined by three distinct record types:

- *Local Vertex Pool Record* - defines the set of vertices that are referenced by the geometric primitives of the mesh.
- *Mesh Primitive Record* - defines a geometric primitive (triangle-strip, triangle-fan, quadrilateral-strip or indexed face set) for the mesh.

A mesh node consists of one mesh record, one local vertex pool record, and one or more mesh primitive records. The mesh primitive records are delimited by push and pop control records as shown in the following example:

```

MESH
LOCAL VERTEX POOL
PUSH
MESH PRIMITIVE
MESH PRIMITIVE
...
MESH PRIMITIVE
POP

```

B.12.4. Mesh Record

The mesh record is the primary record of a mesh node and defines the common “face-like” attributes associated to all geometric primitives of the mesh. These attributes are identical to those of the face record. See [Face Record](#).

Table 9. Mesh Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Mesh Opcode 84	□
Unsigned Int	2	2	Length - length of the record	□
Char	4	8	7 char ASCII ID; 0 terminates	□
Int	4	4	Reserved	□
Int	16	4	IR color code	□
Int	20	2	Relative priority	□
Int	22	1	Draw type	□
			0 = Draw solid with backface culling (front side only)	□
			1 = Draw solid, no backface culling (both sides visible)	□
			2 = Draw wireframe and close	□
			3 = Draw wireframe	□

			4 = Surround with wireframe in alternate color	<input type="checkbox"/>
			8 = Omnidirectional light	<input type="checkbox"/>
			9 = Unidirectional light	<input type="checkbox"/>
			10 = Bidirectional light	<input type="checkbox"/>
Int	23	1	Texture white = if TRUE, draw textured face white	<input type="checkbox"/>
Unsigned Int	24	2	Color name index	<input type="checkbox"/>
Unsigned Int	26	2	Alternate color name index	<input type="checkbox"/>
Int	28	1	Reserved	<input type="checkbox"/>
Int	29	1	Template (billboard)	<input type="checkbox"/>
			0 = Fixed, no alpha blending	<input type="checkbox"/>
			1 = Fixed, alpha blending	<input type="checkbox"/>
			2 = Axial rotate with alpha blending	<input type="checkbox"/>
			4 = Point rotate with alpha blending	<input type="checkbox"/>
Int	30	2	Detail texture pattern index, -1 if none	<input type="checkbox"/> Note: Detail textures are IRIS GL specific
Int	32	2	Texture pattern index, -1 if none	<input type="checkbox"/>
Int	34	2	Material index, -1 if none	<input type="checkbox"/>
Int	36	2	Surface material code (for DFAD)	<input type="checkbox"/> (tentative)
Int	38	2	Feature ID (for DFAD)	<input type="checkbox"/>

Int	40	4	IR material code	□
Unsigned Int	44	2	Transparency	□
			0 = Opaque	
			65535 = Totally clear	
Unsigned Int	46	1	LOD generation control	□
Unsigned Int	47	1	Line style index	□
Int	48	4	Flags (bits from left to right)	□
			0 = Terrain	□
			1 = No color	□
			2 = No alternate color	□
			3 = Packed color	□
			4 = Terrain culture cutout (footprint)	□
			5 = Hidden, not drawn	□
			6 = Roofline	□
			7-31 = Spare	□
Unsigned Int	52	1	Light mode	□
			0 = Use mesh color, not illuminated	□
			1 = Use vertex colors, not illuminated	□
			2 = Use mesh color and vertex normals	□
			3 = Use vertex colors and vertex normals	□
Char	53	7	Reserved	□
Unsigned Int	60	4	Packed color, primary (a, b, g, r) - only b, g, r used	□

Unsigned Int	64	4	Packed color, alternate (a, b, g, r) - only b, g, r used	□
Int	68	2	Texture mapping index	□
Int	70	2	Reserved	□
Unsigned Int	72	4	Primary color index	□
Unsigned Int	76	4	Alternate color index	□
Int	80	2	Reserved	□
Int	82	2	Shader index, -1 if none	□

B.12.5. Local Vertex Pool Record

This record defines a set of vertices that is referenced by the geometry (primitives) of the mesh.

NOTE

Currently the Local Vertex Pool is used exclusively in the context of mesh nodes, but it is designed in a general way so that it may appear in other contexts in future versions of the OpenFlight Scene Description.

Table 10. Local Vertex Pool Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Local Vertex Pool Opcode 85	□

Unsigned Int	2	2	Length - length of the record Note: Since the length of this record is represented by an unsigned short, the maximum length of the vertex pool is 216-1 (or 65535 bytes). If the entire vertex pool cannot fit into this size, one or more continuation records will follow. (See Continuation Record .)	□
Unsigned Int	4	4	Number of vertices - number of vertices in the local vertex pool	□
Unsigned Int	8	4	Attribute mask - Bit mask indicating what kind of vertex information is specified for each vertex in the local vertex pool. Bits are ordered from left to right as follows:	□
			Bit #Description	
			0 Has Position - if set, data for each vertex in will include x, y, and z coordinates (3 doubles)	□

			1 Has Color Index - if set, data for each vertex will include a color value that specifies a color table index as well as an alpha value	<input type="checkbox"/>
			2 Has RGBA Color - if set, data for each vertex will include a color value that is a packed RGBA color value	<input type="checkbox"/>
			Note: Bits 1 and 2 are mutually exclusive - a vertex can have either color index or RGB color value or neither, but not both.	<input type="checkbox"/>
			3 Has Normal - if set, data for each vertex will include a normal (3 floats)	<input type="checkbox"/>
			4 Has Base UV - if set, data for each vertex will include uv texture coordinates for the base texture (2 floats)	<input type="checkbox"/>
			5 Has UV Layer 1 - if set, data for each vertex will include uv texture coordinates for layer 1 (2 floats)	<input type="checkbox"/>

			6 Has UV Layer 2 - if set, data for each vertex will include uv texture coordinates for layer 2 (2 floats)	□
			7 Has UV Layer 3 - if set, data for each vertex will include uv texture coordinates for layer 3 (2 floats)	□
			8 Has UV Layer 4 - if set, data for each vertex will include uv texture coordinates for layer 4 (2 floats)	□
			9 Has UV Layer 5 - if set, data for each vertex will include uv texture coordinates for layer 5 (2 floats)	□
			10 Has UV Layer 6 - if set, data for each vertex will include uv texture coordinates for layer 6 (2 floats)	□
			11 Has UV Layer 7 - if set, data for each vertex will include uv texture coordinates for layer 7 (2 floats)	□
			12-31 Spare	□

Then beginning at offset 12, the following fields are repeated for each vertex in the local vertex pool, depending on the bits set in the Attribute mask field above. In the fields listed below, N ranges from 0 to Number of vertices - 1.

Table 11. Local Vertex Pool Record (Continued)

Data Type	Offset	Length	Description	CDB OpenFlight Reader

Double	Varies	8*3	CoordinateN - Coordinate of vertex N (x, y, z) - present if Attribute mask includes Has Position.	□
Unsigned Int	Varies	4	<p>colorN - Color for vertex N - present if Attribute mask includes Has Color Index or Has RGBA Color.</p> <p>If Has Color Index, lower 3 bytes specify color table index, upper 1 byte is Alpha.</p> <p>If Has RGBA Color, 4 bytes specify (a, b, g, r) values.</p>	□
Float	Varies	4*3	normalN - Normal for vertex N (i, j, k) - present if Attribute mask includes Has Normal.	□
Float	Varies	4*2	uvBaseN - Texture coordinates (u, v) for base texture layer of vertex N - present if Attribute mask includes Has Base UV.	□
Float	Varies	4*2	uv1N - Texture coordinates (u, v) for layer 1 of vertex N - present if Attribute mask includes Has UV Layer 1.	□

Float	Varies	4*2	uv2N - Texture coordinates (u, v) for layer 2 of vertex N - present if Attribute mask includes Has UV Layer 2.	□
Float	Varies	4*2	uv3N - Texture coordinates (u, v) for layer 3 of vertex N - present if Attribute mask includes Has UV Layer 3.	□
Float	Varies	4*2	uv4N - Texture coordinates (u, v) for layer 4 of vertex N - present if Attribute mask includes Has UV Layer 4.	□
Float	Varies	4*2	uv5N - Texture coordinates (u, v) for layer 5 of vertex N - present if Attribute mask includes Has UV Layer 5.	□
Float	Varies	4*2	uv6N - Texture coordinates (u, v) for layer 6 of vertex N - present if Attribute mask includes Has UV Layer 6.	□
Float	Varies	4*2	uv7N - Texture coordinates (u, v) for layer 7 of vertex N - present if Attribute mask includes Has UV Layer 7.	□

Mesh Primitive Record

This record defines a geometric primitive (triangle strip, triangle fan, quadrilateral strip, or indexed polygon) for a mesh.

Table 12. Mesh Primitive Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Mesh Primitive Opcode 86	□
Unsigned Int	2	2	Length - length of the record	□
Int	4	2	Primitive Type - specifies how the vertices of the primitive are interpreted	□
			1 = Triangle Strip	□
			2 = Triangle Fan	□
			3 = Quadrilateral Strip	□
			4 = Indexed Polygon	□
Unsigned Int	6	2	Index Size - specifies the length (in bytes) of each of the vertex indices that follow - will be either 1, 2, or 4	□
Unsigned Int	8	4	Vertex Count - number of vertices contained in this primitive.	□
The following field is repeated for each vertex referenced by the mesh primitive. These vertices are interpreted according to Primitive Type. In the field below, N ranges from 0 to Vertex Count - 1.				□
Int	12+(N*Index Size)	Index Size	IndexN - Index of vertex N of the mesh primitive.	□

Each mesh primitive is represented using the Mesh Primitive record above. The following descriptions explain how the vertices of each primitive type are interpreted as geometry:

- **Indexed Polygon** -This mesh primitive defines a single polygon in the context of the enclosing mesh. This primitive is similar to the other mesh primitives in that it also shares the polygon attributes of the enclosing mesh. It is different from the other mesh primitive types in that while triangle strips/fans and quadrilateral strips describe a set of connected triangles/quadrilaterals, the indexed polygon defines a single polygon. This primitive contains a sequence of indices that reference vertices from the local vertex pool. One polygon is defined by the sequence of vertices in this record. N vertices represent 1 N-sided closed polygon or 1 (N-1)-sided unclosed polygon.

B.12.6. Light Point Nodes

The OpenFlight format supports two kinds of light point records, indexed and inline. In indexed light point records, the attributes are stored in two palettes; the light point appearance palette and the light point animation palette. The indexed light point record simply stores indices into these two palettes. In inline light point records, all the attributes are stored directly in the light point record itself. This section describes both of these records.

Indexed Light Point Record

The indexed light point record is one of the records that can represent a light point node.

The appearance index specifies an entry in the light point appearance palette that contains the visual attributes of the light point.

The animation index specifies an entry in the light point animation palette that contains the behavioral attributes of the light point.

The palette entries referenced by the indexed light point record describe the visual state of the light point's child vertices. Only vertex nodes may be children of light point nodes.

Table 13. Indexed Light Point Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Indexed Light Point Record Opcode 130	□
Unsigned Int	2	2	Length - length of the record	□
Char	4	8	7 char ASCII ID; 0 terminates	□
Int	12	4	Appearance index	□
Int	16	20	Animation index	□
Int	24	4	Draw order (for calligraphic lights)	□
Int	28	4	Reserved	□

Light Point Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider Light Point Records.

The light point record is one of the records that can represent a light point node. The light point record contains attributes describing the visual state of its child vertices. Only vertex nodes may be children of light point nodes.

Light points are geometric points that represent real world light sources such as runway lights, vehicle lights, street lights, and rotating beacons. Light points differ from light sources in that they do not illuminate the scene around them. They are primarily used to model important visual cues without incurring the tremendous rendering overhead associated with light sources.

Most light point attributes are specific to these unique requirements. Light points can be displayed on special purpose calligraphic imaging systems, the more familiar raster variety, or even hybrid raster/calligraphic (RASCAL) systems.

Table 14. Light Point Record

Data Type	Offset	Length	Description
Int	0	2	Light Point Record Opcode 111
Unsigned Int	2	2	Length - length of the record
Char	4	8	7 char ASCII ID; 0 terminates
Int	12	2	Surface material code
Int	14	2	Feature ID
Unsigned Int	16	4	Back color for bidirectional points
Int	20	4	Display mode 0 = RASTER 1 = CALLIGRAPHIC 2 = EITHER
Float	24	4	Intensity - scalar for front colors
Float	28	4	Back intensity - scalar for back color
Float	32	4	Minimum defocus - (0.0 - 1.0) for calligraphic points
Float	36	4	Maximum defocus - (0.0 - 1.0) for calligraphic points

Int	40	4	Fading mode 0 = Enable perspective fading calculations 1 = Disable calculations
Int	44	4	Fog Punch mode 0 = Enable fog punch through calculations 1 = Disable calculations
Int	48	4	Directional mode 0 = Enable directional calculations 1 = Disable calculations
Int	52	4	Range mode 0 = Use depth (Z) buffer calculation 1 = Use slant range calculation
Float	56	4	Min pixel size - minimum diameter of points in pixels
Float	60	4	Max pixel size - maximum diameter of points in pixels
Float	64	4	Actual size - actual diameter of points in database units
Float	68	4	Transparent falloff pixel size - diameter in pixels when points become transparent
Float	72	4	Transparent falloff exponent ≥ 0 - falloff multiplier exponent
			1.0 - linear falloff
Float	76	4	Transparent falloff scalar > 0 - falloff multiplier scale factor

Float	80	4	Transparent falloff clamp - minimum permissible falloff multiplier result
Float	84	4	Fog scalar >= 0 - adjusts range of points for punch threw effect.
Float	88	4	Reserved
Float	92	4	Size difference threshold - point size transition hint to renderer

Table 15. Light Point Record (Continued)

Data Type	Offset	Length	Description
Int	96	4	Directionality 0 = OMNIDIRECTIONAL 1 = UNIDIRECTIONAL 2 = BIDIRECTIONAL
Float	100	4	Horizontal lobe angle - total angle in degrees
Float	104	4	Vertical lobe angle - total angle in degrees
Float	108	4	Lobe roll angle - rotation of lobe about local Y axis in degrees
Float	112	4	Directional falloff exponent >= 0 - falloff multiplier exponent 1.0 - linear falloff
Float	116	4	Directional ambient intensity - of points viewed off axis
Float	120	4	Animation period in seconds
Float	124	4	Animation phase delay in seconds - from start of period

Float	128	4	Animation enabled period in seconds
Float	132	4	Significance - drop out priority for RASCAL lights (0.0 - 1.0)
Int	136	4	Calligraphic draw order - for rendering consistency
Int	140	4	Flags (bits, from left to right)
		0	0 = reserved
		1	1 = No back color
		2	TRUE = don't use back color for bidirectional points
		3	FALSE = use back color for bidirectional points
		4	2 = reserved
		5	3 = Calligraphic proximity occulting (Debunching)
		6	4 = Reflective, non-emissive point
		7	5-7 = Randomize intensity
		8	0 = never
		9	1 = low
		10	2 = medium
		11	3 = high
		12	8 = Perspective mode
		13	9 = Flashing
		14	10 = Rotating
		15	11 = Rotate Counter Clockwise
		16	Direction of rotation about local Z axis
		17	12 = reserved
		18	13-14 = Quality

			0 = Low
			1 = Medium
			2 = High
			3 = Undefined
			15 = Visible during day
			16 = Visible during dusk
			17 = Visible during night
			18-31 = Spare
Float	144	4*3	Axis of rotation for rotating animation (i, j, k)

Light Point System Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider Light Point System Records.

The light point system record enables you to collect a set of light points and enable/disable or brighten/dim them as a group.

Table 16. Light Point System Record

Data Type	Offset	Length	Description
Int	0	2	Light Point System Record Opcode 130
Unsigned Int	2	2	Length - length of the record
Char	4	8	7 char ASCII ID; 0 terminates
Float	12	4	Intensity
Int	16	4	Animation state
			0 = On
			1 = Off
			2 = Random
Int	20	4	Flags (bits, from left to right)
			0 = Enabled
			1-31 = Spare

Degree of Freedom Record

The degree of freedom (DOF) record is the primary record of the DOF node. The DOF node specifies a local coordinate system and the range allowed for translation, rotation, and scale with respect to that coordinate system.

The DOF record can be viewed as a series of applied transformations consisting of the following elements:

[PTTTRRRSSSP]

where “P” denotes “put,” “T” denotes “translate,” “R” denotes “rotate,” and “S” denotes “scale.”

It is important to understand the order in which these transformations are applied to the geometry. A pre-multiplication is assumed, so the sequence of transformations must be read from right to left, in order to describe its effect on the geometry contained below the DOF. In this manner, a DOF is interpreted as a Put followed by three Scales, three Rotates, three Translates, and a Put.

Taking the transformations in right to left order, they represent:

1. A Put (3 point to 3 point transformation). This matrix brings the DOF coordinate system to the world origin, with its x-axis aligned along the world x-axis and its y-axis in the world x-y plane. Testing against the DOF’s constraints is performed in this standard position. This matrix is therefore the inverse of the last (see step 11 below).
2. Scale in x.
3. Scale in y.
4. Scale in z.
5. Rotation about z (yaw).
6. Rotation about y (roll).
7. Rotation about x (pitch).
8. Translation in x.
9. Translation in y.
10. Translation in z.
11. A final Put. This matrix moves the DOF coordinate system back to its original position in the scene.

The DOF record specifies the minimum, maximum, and current values for each transformation. Only the current value affects the actual transformation applied to the geometry. The increment value specifies discrete allowable values within the range of legal values represented by the DOF.

Table 17. Degree of Freedom Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
-----------	--------	--------	-------------	-----------------------

Int	0	2	Degree-of-Freedom Opcode 14	🔗
Unsigned Int	2	2	Length - length of the record	🔗
Char	4	8	7 char ASCII ID; 0 terminates	🔗
Int	12	4	Reserved	🔗
Double	16	8*3	Origin of DOF's local coordinate system (x, y, z)	🔗
Double	40	8*3	Point on x axis of DOF's local coordinate system (x, y, z)	🔗
Double	64	8*3	Point in xy plane of DOF's local coordinate system (x, y, z)	🔗
Double	88	8	Min z value with respect to local coordinate system	🔗
Double	96	8	Max z value with respect to local coordinate system	🔗
Double	104	8	Current z value with respect to local coordinate system	🔗
Double	112	8	Increment in z	🔗
Double	120	8	Min y value with respect to local coordinate system	🔗
Double	128	8	Max y value with respect to the local coordinate system	🔗
Double	136	8	Current y value with respect to local coordinate system	🔗
Double	144	8	Increment in y	🔗

Double	152	8	Min x value with respect to local coordinate system	□
Double	160	8	Max x value with respect to local coordinate system	□
Double	168	8	Current x value with respect to local coordinate system	□
Double	176	8	Increment in x	□
Double	184	8	Min pitch (rotation about the x axis)	□
Double	192	8	Max pitch	□
Double	200	8	Current pitch	□
Double	208	8	Increment in pitch	□
Double	216	8	Min roll (rotation about the y axis)	□
Double	224	8	Max roll	□
Double	232	8	Current roll	□
Double	240	8	Increment in roll	□
Double	248	8	Min yaw (rotation about the z axis)	□
Double	256	8	Max yaw	□
Double	264	8	Current yaw	□
Double	272	8	Increment in yaw	□
Double	280	8	Min z scale (about local origin)	□
Double	288	8	Max z scale (about local origin)	□

Table 18. Degree of Freedom Record (Continued)

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Double	296	8	Current z scale (about local origin)	□
Double	304	8	Increment for scale in z	□

Double	312	8	Min y scale (about local origin)	🔗
Double	320	8	Max y scale (about local origin)	🔗
Double	328	8	Current y scale (about local origin)	🔗
Double	336	8	Increment for scale in y	🔗
Double	344	8	Min x scale (about local origin)	🔗
Double	352	8	Max x scale (about local origin)	🔗
Double	360	8	Current x scale (about local origin)	🔗
Double	368	8	Increment for scale in x	🔗
Int	376	4	Flags (bits, from left to right)	🔗
			0 = x translation is limited	🔗
			1 = y translation is limited	🔗
			2 = z translation is limited	🔗
			3 = Pitch rotation is limited	🔗
			4 = Roll rotation is limited	🔗
			5 = Yaw rotation is limited	🔗
			6 = x scale is limited	🔗
			7 = y scale is limited	🔗
			8 = z scale is limited	🔗
			9 = Reserved	🔗
			10 = Reserved	🔗

			11-31 = Spare	□
Int	380	4	Reserved	□

Vertex List Record

A vertex list record is the primary record of a vertex node. Each record references one or more vertices in the vertex palette. See [Vertex Palette Records](#). A vertex node is a leaf node in the database and therefore cannot have any children.

Table 19. Vertex List Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Vertex List Opcode 72	□
Unsigned Int	2	2	Length - length of the record	□

The following field is repeated for each vertex contained in the vertex list record.

In the field below, N ranges from 0 to Number of vertices - 1, where Number of vertices = (Length - 4) / 4

Int	4+(N*4)	4	OffsetN - Byte offset into vertex palette of the actual vertex for vertex N.	□
-----	---------	---	--	---

Morph Vertex List Record

A morph vertex list record is the primary record of a morph vertex node. Like the vertex list record, each morph vertex list record references one or more vertices in the vertex palette. See: [Vertex Palette Records](#). A morph vertex node is a leaf node in the database and therefore cannot have any children.

Each record references one or more pairs of vertices (weights) in the vertex palette. One weight is the 0 percent morph attributes and the other weight is the 100 percent morph attributes. Since each weight references a vertex, all vertex attributes including color, normal, and texture coordinates may be morphed.

When the eyepoint approaches the switch-in distance, the vertex attributes displayed are 100 percent morphed. When the eyepoint reaches the distance computed by LOD switch-in distance minus LOD transition range, the vertex attributes displayed are 0 percent morphed. Within the LOD transition range, the vertex attributes displayed are interpolated between the two known vertex attributes.

Geometric morphing is controlled by the parent LOD node. Only morph vertex nodes are affected. Both morphing and static geometry (vertices) may exist within the same branch of the database hierarchy.

Table 20. Morph Vertex List Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Morph Vertex List Opcode 89	□
Unsigned Int	2	2	Length - length of the record	□
The following fields are repeated for each vertex contained in the morph vertex list record.				□
In the fields below, N ranges from 0 to Number of vertices - 1, where Number of vertices = (Length - 4) / 8				
Int	4+(N*8)	4	Offset 0N - Offset into vertex palette of Nth 0% vertex.	□
Int	8+(N*8)	4	Offset 100N - Offset into vertex palette of Nth 100% vertex.	□

Binary Separating Plane Record

The (BSP) record is the primary record of the BSP node. A BSP allows you to model 3D databases without depth (Z) buffer support.

An application uses this information to cull portions of the database according to which side of the plane the subtree is situated on with regard to eyepoint position and viewing direction.

This record contains an equation $ax + by + cz + d = 0$ that describes the separating plane.

Table 21. Binary Separating Plane Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Binary Separating Plane (BSP) Opcode 55	□
Unsigned Int	2	2	Length - length of the record	□
Char	4	8	7 char ASCII ID; 0 terminates	□
Int	12	4	Reserved	□
Double	16	8*4	Plane equation coefficients (a, b, c, d)	□

External Reference Record

The external reference record is the primary record of the external reference node. External references allow one database to reference, or instance, a node in another database (or an entire database). At the point of reference, the referenced node/database is copied (or possibly shared) as a child of the current parent node.

The override flags allow the referencing (parent) database to control use of the referenced (child) node/database palettes. If an override flag (e.g., material) is set, the child node/database uses its own (material) palette. Otherwise, the child node/database uses the current (parent's) palette. The override flags are hierarchical and may affect references made by the child node/database.

The view as bounding box field is used by the MultiGen-Paradigm, Inc. modeling environment and is not expected to be used by runtime applications.

Table 22. External Reference Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	External Reference Opcode 63	□
Unsigned Int	2	2	Length - length of the record	□
Char	4	200	199-char ASCII path; 0 terminates Format of this string is: filename<node name> if <node name> absent, references entire file	□
Int	204	4	Reserved	□
Int	208	4	Flags (bits, from left to right)	□
			0 = Color palette override	□
			1 = Material palette override	□
			2 = Texture and texture mapping palette override	□
			3 = Line style palette override	□

			4 = Sound palette override	<input type="checkbox"/>
			5 = Light source palette override	<input type="checkbox"/>
			6 = Light point palette override	<input type="checkbox"/>
			7 = Shader palette override	<input type="checkbox"/>
			8-31 = Spare	<input type="checkbox"/>
Int	212	2	View as bounding box	<input type="checkbox"/>
			0 = View external reference normally	<input type="checkbox"/>
			1 = View external reference as bounding box	<input type="checkbox"/>
Int	214	2	Reserved	<input type="checkbox"/>

Level of Detail Record

The level of detail (LOD) record is the primary record of the LOD node. LOD's are perhaps the most important hierarchical node present in the database tree. Proper use of level-of-detail modeling concepts can vastly improve real-time playback of large databases. Attributes within the LOD record provide switching and transition distances for real-time culling and load management mechanisms.

The center coordinate can be used by a real-time application to calculate the slant range distance from the eyepoint to the LOD. Based upon the result of this calculation, a real-time application can choose not to display the LOD's children and thus reduce system load. The center of the LOD is generally the transformed center of the geometry of the LOD's children. This should include the effects of instancing and (parent) group replication as well.

The use previous slant range flag indicates that the slant range for this LOD is the same as the previous (sibling) LOD, implying the center coordinate is also the same. The real-time application can reuse the previous slant range calculation when evaluating this LOD, thereby improving performance.

If the freeze center flag is not set, the MultiGen-Paradigm, Inc. modeling environment as well as OpenFlight API based applications will recalculate the center point of the LOD when the OpenFlight file is saved.

Transition range specifies the range over which real-time smoothing effects should be employed while switching from one LOD to another. Smoothing effects include geometric morphing and image blending. The smoothing effect is active between: switch-in distance minus transition range

(near), and switch-in distance (far). The center distance of the effect is therefore switch-in distance minus one half the transition range.

Significant size is a value used to calculate switch in and out distances based on viewing parameters of your simulation display system. This value is used internally by MultiGen-Paradigm and will be enhanced in future versions of OpenFlight.

Table 23. Level of Detail Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Level-of-Detail Opcode 73	□
Unsigned Int	2	2	Length - length of the record	□
Char	4	8	7 char ASCII ID; 0 terminates	□
Int	12	4	Reserved	□
Double	16	8	Switch-in distance	□
Double	24	8	Switch-out distance	□
Int	32	2	Special effect ID1 - application defined	□
Int	34	2	Special effect ID2 - application defined	□
Int	36	4	Flags (bits, from left to right)	□
			0 = Use previous slant range	□
			1 = Reserved	□
			2 = Freeze center (don't recalculate)	□
			3-31 = Spare	□
Double	40	8	Center coordinate x of LOD	□
Double	48	8	Center coordinate y of LOD	□
Double	56	8	Center coordinate z of LOD	□
Double	64	8	Transition range	□

Double	72	8	Significant size	□
--------	----	---	------------------	---

Sound Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider Sound Records.

The sound record is the primary record of the sound node. A sound node represents the position and orientation of a sound emitter in the database.

Amplitude and pitch blend are relative to the amplitude in the waveform file. Falloff defines how amplitude falls off when approaching the edge of the sound lobe, with maximum amplitude at the center of the lobe.

Priority determines which sounds are played when more emitters populate a scene than the sound system can play simultaneously.

Width defines the half angle of the sound lobe. Direction sets the type of sound lobe.

Doppler, absorption, and delay flags enable or disable the modeling of Doppler, atmospheric absorption, and propagation delay in the sound environment.

Active indicates a sound is to be activated when read in to the modeling environment.

Table 24. Sound Record

Data Type	Offset	Length	Description
Int	0	2	Sound Node Opcode 91
Unsigned Int	2	2	Length - length of the record
Char	4	8	7 char ASCII ID; 0 terminates
Int	12	4	Reserved
Int	16	4	Index into sound palette
Int	20	4	Reserved
Double	24	8*3	Coordinate of offset from local origin (x, y, z)
Float	48	4*3	Sound direction (vector) wrt local coordinate axes (i, j, k)
Float	60	4	Amplitude of sound
Float	64	4	Pitch bend of sound
Float	68	4	Priority of sound
Float	72	4	Falloff of sound

Float	76	4	Width of sound lobe
Int	80	4	Flags (bits, from left to right)
			0 = Doppler
			1 = Atmospheric absorption
			2 = Delay
			3-4 = Direction:
			0 = Omnidirectional
			1 = Unidirectional
			2 = Bidirectional
			5 = Active
			6-31 = Spare
Int	84	4	Reserved

Light Source Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider Light Source Records.

The light source record is the primary record of the light source node. Light sources illuminate the database. They contain position and rotation data (overriding any information stored in the light palette), an index into the light palette, and information on how the light behaves within the hierarchy.

The enabled flag indicates whether the light is turned on and, therefore, a factor of the lighting (rendering) model.

The global flag specifies whether the light shines on the entire database or only on its children (for example, the cabin light in a car).

Table 25. Light Source Record

Data Type	Offset	Length	Description
Int	0	2	Light Source Record Opcode 101
Unsigned Int	2	2	Length - length of the record
Char	4	8	7 char ASCII ID; 0 terminates
Int	12	4	Reserved
Int	16	4	Index into light palette

Int	20	4	Reserved
Int	24	4	Flags (bits, from left to right)
			0 = Enabled
			1 = Global
			2 = Reserved
			3 = Export
			4 = Reserved
			5-31 = Spare
Int	28	4	Reserved
Double	32	8*3	Position (for Local or Spot lights only) (x, y, z)
Float	56	4	Yaw (azimuth for Infinite or Spot lights only)
Float	60	4	Pitch (elevation for Infinite or Spot lights only)

Road Segment Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider Road Segment Records.

A road segment record is the primary record of a road segment node. It stores the attributes used to create and modify a road segment. The children of the road node represent the geometry and paths of the road and should not be manually edited. Any modification invalidates the road segment.

Table 26. Road Segment Record

Data Type	Offset	Length	Description
Int	0	2	Road Segment Opcode 87
Unsigned Int	2	2	Length of record
Char	4	8	7 char ASCII ID; 0 terminates

Road Construction Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider Road Construction Records.

A road construction record is the primary record of a road construction node. It supersedes the Road Segment Record described previously. It is created by the Pathfinder option of MultiGen II Pro

v1.5 as well as the Road Tool option beginning with Creator v2.1. It stores the parameters defining the road path construction for one road section. In practice, the children of the road construction node usually represent the geometry and paths of the road section. Although every field in the road construction record may be modified, this data makes the most sense when it is kept in sync with the geometry that is created from it. Therefore, typical usage will be read-only access from applications able to analyze the road surface from this given data.

The Road type field dictates how the following fields define the current road section. For all road types, the Entry and Exit control points lie on the boundaries of the road section. The Alignment control point is only necessary for the Curve type as it defines a horizontal tangent with the other control points.

Other fields particular to the Curve type are the horizontal curve parameters. The horizontal components of the Curve type start and end with spiral transitions of specified lengths. An Arc Radius length is used to define the constant curve area. The Superelevation is specified in a rise over run slope measured laterally across the road for the maximum banking which is used throughout the constant curve component. The banking transitions along the spiral sections in one of three ways defined by the Spiral type field.

Both the Curve and Hill types may have a vertical curve component defined by the remaining fields. Slopes are given at both the entry and exit of the section. If the given slopes don't intersect within the road segment then two vertical parabolas are constructed instead of one, and the Additional vertical parabola flag is set. Note that this flag's value is only valid when the Road Tools version field is 3 or later. This flag may also be set when convergence of the slopes creates a vertical curve length less than Minimum curve length. Otherwise, Vertical curve length is used to define the horizontal distance covered by the single parabola vertical curve.

Table 27. Road Construction Record

Data Type	Offset	Length	Description
Int	0	2	Road Construction Opcode 127
Unsigned Int	2	2	Length of record
Char	4	8	7 char ASCII ID; 0 terminates
Char	12	4	Reserved
Int	16	4	Road type
			0 = Curve
			1 = Hill
			2 = Straight
Int	20	4	Road Tools version
Double	24	8*3	Entry control point (x, y, z)
Double	48	8*3	Alignment control point (x, y, z)

Double	72	8*3	Exit control point (x, y, z)
Double	96	8	Arc radius
Double	104	8	Entry spiral length
Double	112	8	Exit spiral length

Table 28. Road Construction Record (Continued)

Data Type	Offset	Length	Description
Double	120	8	Superelevation
Int	128	4	Spiral type
			0 = Linear with length
			1 = Linear with angle
			2 = Cosine with length
Int	132	4	Additional vertical parabola flag
Double	136	8	Vertical curve length
Double	144	8	Minimum curve length
Double	152	8	Entry slope
Double	160	8	Exit slope

Road Path Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider Road Path Records.

A road path record is the primary record of a road path node. A road path node is a child of a road segment node. It describes a lane of the parent road segment. The child of a road path node is a face node whose vertices provide the coordinates of the center of the lane.

Road path record attributes may also be written to an ASCII file for easy access by the application. The format of the file is described in [Road Path Files](#).

Table 29. Road Path Record

Data Type	Offset	Length	Description
Int	0	2	Road Path Opcode 92
Unsigned Int	2	2	Length of record
Char	4	8	7 char ASCII ID; 0 terminates
Int	12	4	Reserved
Char	16	120	Path name; 0 terminates

Double	136	8	Speed limit
Boolean	144	4	No passing
Int	148	4	Vertex normal type
			0 = Up-vector
			1 = Heading, Pitch, Roll
Int	152	480	Reserved

Clip Region Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider Clip Region Records.

A clip region record is the primary record of a clip node. It defines those regions in 3D space in which drawing occurs. Clip regions only clip the geometry below the clip node in the hierarchy.

The coordinates create a four-sided face that defines the clip region in space. Planes are formed along the edges of the four-sided face normal to the face; a fifth plane clips the back side of the face.

Table 30. Clip Region Record

Data Type	Offset	Length	Description
Int	0	2	Clip Region Opcode 98
Unsigned Int	2	2	Length of record
Char	4	8	7 char ASCII ID; 0 terminates
Int	12	4	Reserved
Int	16	2	Reserved
Char	18	5	Flags for enabling the individual clip planes
			Char 0 is flag for edge defined by coordinate 0 and 1 Char 1 is flag for edge defined by coordinate 1 and 2 Char 2 is flag for edge defined by coordinate 2 and 3 Char 3 is flag for edge defined by coordinate 3 and 0 Char 5 is flag for plane that clips the half space behind the clip region
Char	23	1	Reserved

Double	24	8*3	1st coordinate defining the clip region (x, y, z)
Double	48	8*3	2nd coordinate defining the clip region (x, y, z)
Double	72	8*3	3rd coordinate defining the clip region (x, y, z)
Double	96	8*3	4th coordinate defining the clip region (x, y, z)
Double	120	8*20	Five plane equation coefficients (ax + by +cz + d) Coefficients are ordered: a0, a1, a2, a3, a4 b0, b1, b2, b3, b4 c0, c1, c2, c3, c4 d0, d1, d2, d3, d4

Text Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider Text Records.

The text record is the primary record of the text node. Text draws a string of data using a specified font. The record specifies the visual characteristics of the text and formatting information.

The actual string for the text is stored in the comment record immediately following. The format of the text record is:

Table 31. Text Record

Data Type	Offset	Length	Description
Int	0	2	Text Opcode 95
Unsigned Int	2	2	Length of record
Char	4	8	7 char ASCII ID; 0 terminates
Int	12	4	Reserved
Int	16	4	Reserved
Int	20	4	Type
			-1 = Static
			0 = Text String
			1 = Float

			2 = Integer
Int	24	4	Draw type
			0 = Solid
			1 = Wireframe and close
			2 = Wireframe
			3 = Surround with wireframe in alternate color
Int	28	4	Justification
			0 = Left
			1 = Right
			2 = Center
Double	32	8	Floating point value
Int	40	4	Integer value
Int	44	4*5	Reserved
Int	64	4	Flags (bits, from left to right)
			0 = Boxable (Unused)
			1-31 = Spare
Int	68	4	Color
Int	72	4	Color 2 (Unused)
Int	76	4	Material
Int	80	4	Reserved
Int	84	4	Maximum number of lines (Unused)
Int	88	4	Maximum number of characters
Int	92	4	Current length of text (Unused)
Int	96	4	Next line number available (Unused)
Int	100	4	Line number at top of display (Unused)
Int	104	4*2	Low/high values for integers

Double	112	8*2	Low/high values for floats
Double	128	8*3	Lower-left corner of rectangle around text (x, y, z)
Double	152	8*3	Upper-right corner of rectangle around text (x, y, z)
Char	176	120	Font name
Int	296	4	Draw vertical
Int	300	4	Draw italic
Int	304	4	Draw bold
Int	308	4	Draw underline
Int	312	4	Line style
Int	316	4	Reserved

Switch Record

A switch record is the primary record of a switch node. A switch represents a set of masks that control the display of the switch's children.

Each mask contains one bit for each child of the switch. Each mask bit indicates that the corresponding child is selected (1) or deselected (0). Each mask selects some, none, or all of the children for display according to the state of the mask bits.

Both the switch children and mask bits begin counting from 0. Therefore the selection state, for a particular switch child is derived from a given mask with the following calculation:

```
mask_bit = 1 << (child_num % 32)
```

```
mask_word = mask_words [mask_num * num_words + child_num / 32]
```

```
child_selected = mask_word & mask_bit
```

The current mask value is an index into the set of masks and indicates the selected mask.

The masks of a switch node can be named. These names are stored in the ancillary record, indexed string record. See [Indexed String Record](#).

Table 32. Switch Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Switch Opcode 96	□
Unsigned Int	2	2	Length of record	□

Char	4	8	7 char ASCII ID; 0 terminates	
Int	12	4	Reserved	
Int	16	4	Current mask	
Int	20	4	Number of masks	
Int	24	4	Number of words per mask - the number of 32 bit words required for each mask, calculated as follows: (number of children / 32) + X where X equals: 0 if (number of children modulo 32) is zero 1 if (number of children modulo 32) is nonzero	
Unsigned Int	28	Variable	Mask words. The length (in bytes) can be calculated as follows: Number of words per mask * Number of masks * 4 bytes	

CAT Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider CAT Records.



A continuously adaptive terrain (CAT) record is the primary record of a CAT node. A continuously adaptive terrain skin is a hierarchical triangle mesh designed for high fidelity, real-time viewing. A CAT skin is represented in OpenFlight by a record stream consisting of: a CAT record, a set of CAT data records, a push record, the CAT hierarchy and geometry, and a pop record. CAT hierarchy and geometry is represented by standard OpenFlight constructs of LOD's, groups, external references, faces, and vertices.

Table 33. CAT Record

Data Type	Offset	Length	Description
Int	0	2	CAT Opcode 115
Unsigned Int	2	2	Length - length of the record
Char	4	8	7 char ASCII ID; 0 terminates
Int	12	4	Reserved
Int	16	4	IR color code
Int	20	1	Draw type
			0 = Hidden, don't draw
			1 = Draw solid, no backface
			2 = Draw wireframe
Int	21	1	Texture white = if TRUE, draw textured face white
Int	22	2	Reserved
Unsigned Int	24	2	Color name index

Unsigned Int	26	2	Alternate color name index
Int	28	2	Detail texture pattern index, -1 if none
Int	30	2	Texture pattern index, -1 if none
Int	32	2	Material index, -1 if none
Int	34	2	Surface material code (for DFAD)
Int	36	4	IR material code
Int	40	4*2	Reserved
Int	48	2	Texture mapping index
Int	50	2	Reserved
Unsigned Int	52	4	Primary color index
Unsigned Int	56	4	Alternate color index

Table 34. CAT Record (Continued)

Data Type	Offset	Length	Description
Int	60	4	Reserved
Double	64	8	Reserved
Int	72	4	Flags (bits, from left to right)
			0 = No color
			1 = No alternate color
			2-31 = Spare
Int	76	4	Reserved

Extension Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider Extension Records.

An extension node record is the primary record of an extension node. It introduces a user defined node type that is defined by an extension site that utilizes the extensibility of the OpenFlight format. It specifies the site information for a third party record which contains additional data that is not represented by the standard OpenFlight records. The content of the data itself is transparent to users other than the extension site. The data can be accessed by the combination of the OpenFlight API and the data dictionary defined by the extension site.

The relationship of an extension node relative to other hierarchical nodes is defined by the standard push and pop control records. For more information about extensions, please refer to

the “OpenFlight API User’s Guide, Level 3: Extensions”.

The extension record (Opcode 100) may also introduce new attributes to existing nodes (See [Extension Attribute Record](#).)

Table 35. Extension Record

Data Type	Offset	Length	Description
Int	0	2	Extension Opcode 100
Unsigned Int	2	2	Length of the total extension record
Char	4	8	7 char ASCII ID; 0 terminates
Char	12	8	Site ID - Unique site name. 7 char ASCII ID; 0 terminates
Int	20	1	Reserved
Int	21	1	Revision - site specific
Unsigned Int	22	2	Record code - site specific
Char	24	Varies	Extended data - site specific

Curve Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider Curve Records.

A curve record is the primary record of a curve node. A curve node represents one or more curve segments joined together with at least G0 continuity. Let a curve segment be defined by 4 geometric constraints. We will call these geometric constraints control points in the curve record. The way the control points are grouped and used will be discussed below.

Let each control point be a double precision 3D coordinate, $P = (x, y, z)$.

Let the group of control points be (P_0, P_1, \dots, P_k) .

The currently defined curve types are B-spline, Cardinal, and Bezier.

If the curve type is Bezier, P_0, P_1, P_2 , and P_3 form the first curve segment. P_3, P_4, P_5 , and P_6 form the next segment, and so on. Notice that the last control point in the first segment becomes the first control point in the second segment.

If the curve type is either B-spline or Cardinal, P_0, P_1, P_2 , and P_3 form the first curve segment. P_1, P_2, P_3 , and P_4 from the next segment, and so on. Notice that the second control point in the first segment becomes the first control point in the second segment.

Note that the smoothness of the curve depends on how many times your renderer samples the

curve equation into piece-wise linear elements. In the MultiGen-Paradigm, Inc. modeling environment, each curve segment is evenly sampled 11 times to produce 10 lines per curve segment.

Table 36. Curve Record

Data Type	Offset	Length	Description
Int	0	2	Curve Opcode 126
Unsigned Int	2	2	Length of the total curve record
Char	4	8	7 char ASCII ID; 0 terminates
Int	12	4	Reserved
Int	16	4	Curve type
			4 = B-spline
			5 = Cardinal
			6 = Bezier
Int	20	4	Number of control points
Char	24	8	Reserved
Double	32	Variable	Coordinates of control points. Each coordinate is (x, y, z)
			Coordinates are ordered: cp0x, cp0y, cp0z, cp1x, cp1y, cp1z, ... cpNx, cpNy, cpNz where N is Number of control points - 1
			(Length = Number of control points * 3 * 8 bytes.)

Ancillary Records

Ancillary records follow node primary records. They contain supplementary attribute data for the node they follow. Ancillary records are optional but must precede any control record, following the node primary record, when present, as shown in this example:

```

GROUP
COMMENT
LONG ID
PUSH
...
POP

```

In this example, the comment and long ID ancillary records apply to the group record. There is no order dependency between ancillary records. The comment could appear before or after the long ID record in the example above, but must appear before any control record.

Comment Record

A comment record is an ancillary record that contains text data that belongs to the preceding node primary record. The text description is a variable length ASCII string terminated by a <nil> character.

Table 37. Comment Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Comment Opcode 31	□
Unsigned Int	2	2	Length - length of the record	□
Char	4	Length - 4	Text description of node; 0 terminates	□

Long ID Record

A long ID record is an ancillary record that contains the full name of the preceding node. It is present only when the name exceeds eight characters (seven characters plus a terminating <nil> character).

Note that the ID field found in third field of every primary OpenField record must be unique. The ID itself can be in Short or Long form. In Short form, the ID is limited to a 7 char ASCII string. In Long form, the ID can be of up to (64K – 5) characters in length. The Long ID record, when present, replaces the 7 char string found in the third of the primary record.

Table 38. Long ID Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Long ID Opcode 33	□
Unsigned Int	2	2	Length - length of the record	□

Char	4	Length - 4	ASCII ID of node; 0 terminates	
------	---	------------	--------------------------------	--

Indexed String Record

An indexed string record is an ancillary record that contains an integer index followed by a variable length character string. In this way, arbitrary strings can be associated to indices in a general way.

Currently, indexed string records are only used in the context of switch nodes, for which they represent the names of the masks contained in the switch node. The index specifies the mask number for which the string specifies the name. Mask numbers start at 0. Not all masks are required to have names.

Table 39. Indexed String Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Indexed string Opcode 132	
Unsigned Int	2	2	Length - length of the record	
Unsigned Int	4	4	Index	
Char	8	Length - 8	ASCII string; 0 terminates	

B.12.7. Multitexture

OpenFlight supports eight textures per polygon or mesh as well as eight uv values per vertex. The current texture information stored on the polygon is referred to as “the base texture” or “texture layer 0”. Each additional texture is referred to as “texture layer N”. Therefore, to support eight textures per polygon, a base texture is required as well as seven additional texture layers. Not all layers are required. Nor is any mandate set forth requiring that layers be contiguous after the base layer. The additional texture layers for each polygon, mesh, and vertex are represented in ancillary records at the face, mesh and vertex primary node level as shown in the following example:

```
FACE
MULTITEXTURE
PUSH
VERTEX LIST
UV LIST
POP
```

The records that are used to represent multitexture in the OpenFlight file are described in the following sections.

Multitexture Record

The multitexture record is an ancillary record of face and mesh nodes. It specifies the texture layer information for the face or mesh.

Table 40. Multitexture Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Unsigned Int	0	2	Multitexture Opcode 52	□
Unsigned Int	2	2	Length - length of the record	□
Int	4	4	Attribute mask - Bit mask indicating what kind of multi-texture information is present in this record. Bits are ordered from left to right and have the following definitions:	□
			Bit # Description	□
			0 Has Layer 1 - if set, multitexture information for texture layer 1 is present.	□
			1 Has Layer 2 - if set, multitexture information for texture layer 2 is present.	□
			2 Has Layer 3 - if set, multitexture information for texture layer 3 is present.	□
			3 Has Layer 4 - if set, multitexture information for texture layer 4 is present.	□

			4 Has Layer 5 - if set, multitexture information for texture layer 5 is present.	<input type="checkbox"/>
			5 Has Layer 6 - if set, multitexture information for texture layer 6 is present.	<input type="checkbox"/>
			6 Has Layer 7 - if set, multitexture information for texture layer 7 is present.	<input type="checkbox"/>
			7-31 Spare	<input type="checkbox"/>
Unsigned Int	Varies	2	textureN - Texture index for texture layer N	<input type="checkbox"/>
Unsigned Int	Varies	2	effectN - Multitexture effect for texture layer N	<input type="checkbox"/>
			0 = Texture environment	<input type="checkbox"/>
			1 = Bump map	<input type="checkbox"/>
			2-100 = Reserved by MultiGen-Paradigm, Inc.	<input type="checkbox"/>
			>100 = user (runtime) defined	<input type="checkbox"/>
Unsigned Int	Varies	2	mappingN - Texture mapping index for texture layer N	<input type="checkbox"/>

Unsigned Int	Varies	2	dataN - Texture data for layer N. This is user defined. For example, it may be used as a blend percentage or color or any other data needed by the runtime to describe texture layer N	
--------------	--------	---	---	--

UV List Record

The uv list record is an ancillary record of vertex nodes. This record (if present) always follows the vertex list or morph vertex list record and contains texture layer information for the vertices represented in the vertex list record it follows.

Table 41. UV List Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Unsigned Int	0	2	UV List Opcode 53	
Unsigned Int	2	2	Length - length of the record	
Int	4	4	Attribute mask - Bit mask indicating what kind of multi-texture information is present in this record. Bits are ordered from left to right as follows:	
			Bit # Description	
			0 Has Layer 1 - if set, uvs for layer 1 are present	
			1 Has Layer 2 - if set, uvs for layer 2 are present	

			2 Has Layer 3 - if set, uvs for layer 3 are present	<input type="checkbox"/>
			3 Has Layer 4 - if set, uvs for layer 4 are present	<input type="checkbox"/>
			4 Has Layer 5 - if set, uvs for layer 5 are present	<input type="checkbox"/>
			5 Has Layer 6 - if set, uvs for layer 6 are present	<input type="checkbox"/>
			6 Has Layer 7 - if set, uvs for layer 7 are present	<input type="checkbox"/>
			7-31 Spare	<input type="checkbox"/>

The following fields are repeated for each vertex contained in the corresponding vertex list or morph vertex list record.

If this uv list record follows a vertex list record, the following fields are repeated for each layer present (as specified by the bits set in Attribute mask).

Data Type	Offset	Length
Float	4	ui, N - Texture coordinate U for vertex i, layer N
Float	4	vi, N - Texture coordinate V for vertex i, layer N

The number of vertices represented in the uv list record that follows a vertex list record must be identical to the number of vertices contained in that vertex list record. This number can also be calculated as follows:

Number of vertices = (Length - 8) / (8 * X), where X is the number of bits set in Attribute mask.

If this uv list record follows a morph vertex list record, the following fields are repeated for each layer present (as specified by the bits set in Attribute mask).

Data Type	Offset	Length
Float	4	u0i, N - Texture U for the 0% vertex i, layer N
Float	4	v0i, N - Texture V for the 0% vertex i, layer N

Float	4	u100i, N - Texture U for the 100% vertex i, layer N
Float	4	v100i, N - Texture V for the 100% vertex i, layer N

Again, the number of vertices represented in the uv list record that follows a morph vertex list record must be identical to the number of vertices contained in that morph vertex list record. This number can also be calculated as follows:

Number of vertices = (Length - 8) / (16 * X), where X is the number of bits set in Attribute mask.

Example

Consider a triangular face (3 vertices) that contains morph vertex information and has texture layers 1 and 3 defined. The following example shows the contents of the uv list record corresponding to the morph vertex list record representing this triangle:

Data Type	Offset	Length	Description
opcode	Unsigned Int	2	53 (UV List opcode).
length	Unsigned Int	2	200 (Length of the record)
uvmask	Unsigned Int	4	1010 0000 0000 0000 (layers 1and 3 ON, others OFF)
u0 1,1	Float	8	Texture U for the 0% vertex 1, layer 1.
v0 1,1	Float	8	Texture V for the 0% vertex 1, layer 1.
u100 1,1	Float	8	Texture U for the 100% vertex 1, layer 1.
v100 1,1	Float	8	Texture V for the 100% vertex 1, layer 1.
u0 1,3	Float	8	Texture U for the 0% vertex 1, layer 3.
v0 1,3	Float	8	Texture V for the 0% vertex 1, layer 3.
u100 1,3	Float	8	Texture U for the 100% vertex 1, layer 3.
v100 1,3	Float	8	Texture V for the 100% vertex 1, layer 3.
u0 2,1	Float	8	Texture U for the 0% vertex 2, layer 1.

v0 2,1	Float	8	Texture V for the 0% vertex 2, layer 1.
u100 2,1	Float	8	Texture U for the 100% vertex 2, layer 1.
v100 2,1	Float	8	Texture V for the 100% vertex 2, layer 1.
u0 2,3	Float	8	Texture U for the 0% vertex 2, layer 3.
v0 2,3	Float	8	Texture V for the 0% vertex 2, layer 3.
u100 2,3	Float	8	Texture U for the 100% vertex 2, layer 3.
v100 2,3	Float	8	Texture V for the 100% vertex 2, layer 3.
u0 3,1	Float	8	Texture U for the 0% vertex 3, layer 1.
v0 3,1	Float	8	Texture V for the 0% vertex 3, layer 1.
u100 3,1	Float	8	Texture U for the 100% vertex 3, layer 1.
v100 3,1	Float	8	Texture V for the 100% vertex 3, layer 1.
u0 3,3	Float	8	Texture U for the 0% vertex 3, layer 3.
v0 3,3	Float	8	Texture V for the 0% vertex 3, layer 3.
u100 3,3	Float	8	Texture U for the 100% vertex 3, layer 3.
v100 3,3	Float	8	Texture V for the 100% vertex 3, layer 3

Replicate Record

A replicate record is an ancillary record of group, face, and light (string) point nodes. It indicates the number of times the group, face, or light (string) point is instantiated. An ancillary transformation record must also be present. The transformation is iteratively applied to each instance to uniquely place it in the database.

Table 42. Replicate Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader

Int	0	2	Replicate Opcode 60	□
Unsigned Int	2	2	Length - length of the record	□
Int	4	2	Number of replications	□
Int	6	2	Reserved	□

Road Zone Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider Road Zone Records.

The road zone record is an ancillary record of the header node. It references a road zone file that contains gridded elevation data. The format of the file is described in [Road Zone Files](#).

Table 43. Road Zone Record

Data Type	Offset	Length	Description
Int	0	2	Road Path Opcode 88
Unsigned Int	2	2	Length - length of the record
Char	4	120	Zone file name; 0 terminates
Int	124	4	Reserved
Double	128	8	Lower-left x coordinate
Double	136	8	Lower-left y coordinate
Double	144	8	Upper-right x coordinate
Double	152	8	Upper-right y coordinate
Double	160	8	Grid interval
Int	168	4	Number of posts along x axis
Int	172	4	Number of posts along y axis

B.12.8. Transformation Records

CDB OpenFlight Readers: CDB-compliant OpenFlight readers consider only the Matrix Transformation Records. The Rotate About Edge Record, Translate Record, Scale Record, Rotate About Point Record, Rotate and/or Scale to Point Record, Put Record and General Matrix Record are specific to the MultiGen Creator tool; as a result, CDB OpenFlight readers do not consider them.

Transformation records may be ancillary records of most nodes. All hierarchical nodes may be transformed except the header node. Some nodes may only be transformed implicitly, as part of some other operation, such as point replication within a light point string.

There are several distinct types of transformation records. For a transformation applied to any node, a matrix record is always present and represents the final (composite) transformation. When present, the transformation records that follow a matrix record represent the individual transformations applied to the node. If an application only needs the final transformation, the matrix record is sufficient and the transformation records that follow the matrix record can be ignored. The records following the matrix record are only needed by the application if it needs to decompose the transformation. The MultiGen-Paradigm, Inc. modeling environment uses these records in order to allow the modeler to modify any of the discrete transformations applied to a node.

Again, each record that follows the matrix record represents a discrete transformation that has been concatenated to form the composite matrix. Concatenation is done in the order that the records are encountered, using pre-multiplication.

NOTE

The final and general matrices are only single-precision, while the discrete transformations are double-precision.

Table 44. Matrix Record

Data Type	Offset	Length	Description
Int	0	2	Matrix Opcode 49
Unsigned Int	2	2	Length - length of the record
Float	4	4*16	4x4 matrix, row major order

Table 45. Rotate About Edge Record

Data Type	Offset	Length	Description
Int	0	2	Rotate About Edge Opcode 76
Unsigned Int	2	2	Length - length of the record
Int	4	4	Reserved
Double	8	8*3	First point on edge (x, y, z)
Double	32	8*3	Second point on edge (x, y, z)
Float	56	4	Angle by which to rotate
Int	60	4	Reserved

Table 46. Translate Record

Data Type	Offset	Length	Description
Int	0	2	Translate Opcode 78
Unsigned Int	2	2	Length - length of the record
Int	4	4	Reserved
Double	8	8*3	From point (x, y, z)
Double	32	8*3	Delta to translate (x, y, z)

Table 47. Scale Record

Data Type	Offset	Length	Description
Int	0	2	Scale Opcode 79
Unsigned Int	2	2	Length - length of the record
Int	4	4	Reserved
Double	8	8*3	Scale center (x, y, z)
Float	32	4	x scale factor
Float	36	4	y scale factor
Float	40	4	z scale factor
Int	44	4	Reserved

Table 48. Rotate About Point Record

Data Type	Offset	Length	Description
Int	0	2	Rotate About Point Opcode 80
Unsigned Int	2	2	Length - length of the record
Int	4	4	Reserved
Double	8	8*3	Rotation center point (x, y, z)
Float	32	4	i, axis of rotation
Float	36	4	j, axis of rotation
Float	40	4	k, axis of rotation
Float	44	4	Angle by which to rotate

Table 49. Rotate and/or Scale to Point Record

Data Type	Offset	Length	Description
Int	0	2	Rotate and/or Scale Opcode 81
Unsigned Int	2	2	Length - length of the record
Int	4	4	Reserved
Double	8	8*3	Scale center (x, y, z)
Double	32	8*3	Reference point (x, y, z)
Double	56	8*3	To point (x, y, z)
Float	80	4	Overall scale factor
Float	84	4	Scale factor in direction of axis
Float	88	4	Angle by which to rotate
Int	92	4	Reserved

Table 50. Put Record

Data Type	Offset	Length	Description
Int	0	2	Put Opcode 82
Unsigned Int	2	2	Length - length of the record
Int	4	4	Reserved
Double	8	8*3	From point origin (x, y, z)
Double	32	8*3	From point align (x, y, z)
Double	56	8*3	From point track (x, y, z)
Double	80	8*3	To point origin (x, y, z)
Double	104	8*3	To point align (x, y, z)
Double	128	8*3	To point track (x, y, z)

Table 51. General Matrix Record

Data Type	Offset	Length	Description
Int	0	2	General Matrix Opcode 82
Unsigned Int	2	2	Length - length of the record

Float	4	4*16	4x4 matrix, row major order
-------	---	------	-----------------------------

Vector Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider Vector Records.

A vector record is an ancillary record of the (pre v15.4) face node. Its only use is to provide the direction vector for old-style unidirectional and bidirectional light point faces.

Table 52. Vector Record

Data Type	Offset	Length	Description
Int	0	2	Vector Opcode 50
Unsigned Int	2	2	Length - length of the record
Float	4	4	i component
Float	8	4	j component
Float	12	4	k component

B.12.9. Bounding Volume Records

Bounding volumes are ancillary records for group nodes only. They generally encompass all the geometry of a group's children. A bounding volume may describe a box, sphere, cylinder, convex hull or histogram.

The center coordinate of a bounding volume is stored as a separate record. The orientation of a bounding volume is also stored as a separate record. The convex hull data is represented by a sequence of triangles forming the convex hull around the group geometry.

Applications may use the bounding volume information with culling and collision detection algorithms.

Table 53. Bounding Box Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Bounding Box Opcode 74	□
Unsigned Int	2	2	Length - length of the record	□
Int	4	4	Reserved	□
Double	8	8	x coordinate of lowest corner	□
Double	16	8	y coordinate of lowest corner	□

Double	24	8	z coordinate of lowest corner	□
Double	32	8	x coordinate of highest corner	□
Double	40	8	y coordinate of highest corner	□
Double	48	8	z coordinate of highest corner	□

Table 54. Bounding Sphere Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Bounding Sphere Opcode 105	□
Unsigned Int	2	2	Length - length of the record	□
Int	4	4	Reserved	□
Double	8	8	Radius of the sphere	□

Table 55. Bounding Cylinder Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Bounding Cylinder Opcode 106	□
Unsigned Int	2	2	Length - length of the record	□
Int	4	4	Reserved	□
Double	8	8	Radius of the cylinder base	□
Double	16	8	Height of the cylinder	□

Table 56. Bounding Convex Hull Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Bounding Convex Hull Opcode 107	□
Unsigned Int	2	2	Length - length of the record	□

Int	4	4	Number of triangles	□
The following fields are repeated for each triangle represented in the convex hull data.				□
In the fields listed below, N ranges from 0 to Number of triangles-1.				
Double	8+(N*72)	8	x coordinate of vertex 1 of triangleN	□
Double	16+(N*72)	8	y coordinate of vertex 1 of triangleN	□
Double	24+(N*72)	8	z coordinate of vertex 1 of triangleN	□
Double	32+(N*72)	8	x coordinate of vertex 2 of triangleN	□
Double	40+(N*72)	8	y coordinate of vertex 2 of triangleN	□
Double	48+(N*72)	8	z coordinate of vertex 2 of triangleN	□
Double	56+(N*72)	8	x coordinate of vertex 3 of triangleN	□
Double	64+(N*72)	8	y coordinate of vertex 3 of triangleN	□
Double	72+(N*72)	8	z coordinate of vertex 3 of triangleN	□

Table 57. Bounding Histogram Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Bounding Histogram Opcode 119	□
Unsigned Int	2	2	Length - length of the record	□

Char	4	Variable	The contents of this record is reserved for use by MultiGen-Paradigm.	
------	---	----------	---	--

Table 58. Bounding Volume Center Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Bounding Volume Center Opcode 108	
Unsigned Int	2	2	Length - length of the record	
Int	4	4	Reserved	
Double	8	8	x coordinate of center	
Double	16	8	y coordinate of center	
Double	24	8	z coordinate of center	

Table 59. Bounding Volume Orientation Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Bounding Volume Orientation Opcode 109	
Unsigned Int	2	2	Length - length of the record	
Int	4	4	Reserved	
Double	8	8	Yaw angle	
Double	16	8	Pitch angle	
Double	24	8	Roll angle	

CAT Data Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider CAT Records.

The CAT data records contain the information needed to reconstruct a Continuously Adaptive Terrain skin from its OpenFlight representation. They provide the information which links faces between levels of detail within a CAT skin. CAT data is stored as a key table with an opcode of 116. For specific detail please refer to [Key Table Records](#).

Each CAT data record describes how a face within a CAT skin is related to faces at the next finer level of detail. The coarsest level of detail is level zero. The next finer level of detail is one, and so forth. Each data record is stored in the key table using an ordinal key, counting up from zero. The face node ID is stored in the data record, thereby providing the cross reference to the OpenFlight face node that represents it.

In OpenFlight, each CAT level of detail is parented by a LOD node. Each CAT triangle strip is parented by a group node.

Table 60. CAT Data Header Record

Data Type	Offset	Length	Description
Int	0	2	CAT Data Opcode 116
Unsigned Int	2	2	Length - length of the record
Int	4	4	Subtype
			1 = indicates this record is a key table header
Int	8	4	Max number - maximum number of face keys
Int	12	4	Actual number - actual number of face keys

Table 61. CAT Data Header Record (Continued)

Data Type	Offset	Length	Description
Int	16	4	Total length of packed face data
Int	20	4	Reserved
Int	24	4	Reserved
Int	28	4	Reserved

The following fields are repeated for each face record represented in the CAT data.

In the fields listed below, N ranges from 0 to Actual number - 1.

Int	$32 + (N * 12)$	4	Face indexN - index of face N
Int	$36 + (N * 12)$	4	ReservedN - reserved space for face N

Int	$40 + (N * 12)$	4	<p>Face data offsetN - offset for face data record N in the CAT data.</p> <p>Note: This offset is measured relative to the Packed face data field in the CAT data face record described below.</p>
-----	-----------------	---	--

Table 62. CAT Data Face Record

Data Type	Offset	Length	Description
Int	0	2	CAT Data Opcode 116
Unsigned Int	2	2	Length - length of the record
Int	4	4	Subtype
			2 = indicates this record is a key data record
Int	8	4	Total length of all packed face records

The following fields constitute one face record and are repeated for each face record represented in the CAT data. In the fields listed below, N ranges from 0 to Actual number - 1. Actual number is from the CAT data header record.

Int	Varies	4	LODN - Level of detail to which this face N belongs.
Int	Varies	4	Child index 1N - The 1st child (within this table) of face N, or -1 for no face.
Int	Varies	4	Child index 2N - The 2nd child (within this table) of face N, or -1 for no face.
Int	Varies	4	Child index 3N - The 3rd child index (within this table) of face N, or -1 for no face.

Int	Varies	4	Child index 4N - The 4th child index (within this table) of face N, or -1 for no face.
Int	Varies	4	ID LengthN - length of face node ID string which follows
Char	Varies	Varies	IDN - ASCII ID of the face to which this record applies.

Extension Attribute Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider Extension Attribute Records.

The extension attribute record is an ancillary record defined by an extension site that utilizes the extensibility of the OpenFlight format. It specifies the site information of a third party extended record which describes additional data that is not represented by the standard OpenFlight records. The data itself is transparent to users other than the extension site. The data can be accessed by the combination of the OpenFlight API and the data dictionary defined by the extension site.

Any hierarchical node can contain extension attribute records. Extension attributes are introduced by an push extension control record and concluded by a pop extension control record.

The extension record (Opcode 100) may also introduce a new node type (See [Extension Record](#).)

Table 63. Extension Attribute Record

Data Type	Offset	Length	Description
Int	0	2	Extension Opcode 100
Unsigned Int	2	2	Length of the total extension record
Char	4	8	7 char ASCII ID; 0 terminates
Char	12	8	Site ID - Unique site name
Int	20	1	Reserved
Int	21	1	Revision - site specific
Unsigned Int	22	2	Record code - site specific
Char	24	Variable	Extended data - site specific

Continuation Record

All OpenFlight records begin with a 4 byte sequence. The first two bytes identify the record (opcode) and the second two bytes specify the length of the record. Given this regular record structure, the length of all OpenFlight records is limited to the largest value that can be encoded with 2 bytes or 16 bits (65535). For fixed-size records, this maximum size is sufficient. For variable-size records, this limitation is addressed with the continuation record which is described in this section.

The continuation record accommodates variable size records in the OpenFlight Scene Description. The continuation record is used to “continue” a record in the OpenFlight file stream. It appears in the stream immediately following the record that it “continues” (the record that is being continued will be referred to as the “original” record). In this way, the continuation record is an ancillary record to any other record type. The data contained in the continuation record is defined by the original record and is assumed to be directly appended onto the content of the original record.

NOTE

Multiple continuation records may follow a record, in which case all continuation records would be appended (in sequence) to the original record.

Table 64. Continuation Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Unsigned Int	0	2	Continuation Record Opcode 23	□
Unsigned Int	2	2	Length - length of the record	□
Varies	4	Length-4	Depends on the original record. The contents of this field are to be appended directly to the end of the original record contents (before the original record contents are parsed)	□

In theory, any OpenFlight record may be “continued”, but in practice only variable length records, whose length is likely to exceed 65535 bytes, are. Following is a list of the variable length OpenFlight record types to which the continuation record is likely to apply:

- [Extension Record](#)
- [Name Table Record](#)
- [Local Vertex Pool Record](#)
- [Mesh Primitive Record](#)

Example: In the following example, the color name table is “too” big to fit in 65535 bytes so the primary palette record, NAME TABLE, is followed by one (or more) CONTINUATION records. The contents of each of the continuation records is appended to the contents of the name table record before the name table data is parsed.

```
NAME TABLE  
CONTINUATION  
CONTINUATION
```

B.12.10. Palette Records

Palette records are ancillary records of the header node. They contain attribute data globally shared by other nodes in the database. Other nodes, such as face nodes, reference the palette data by index.

Individual palettes contain resources such as vertex, material, light source, texture pattern, and line style definitions.

Vertex Palette Records

Double precision vertex records are stored in a vertex palette for the entire database. Vertices shared by one or more geometric entities are written only one time in the vertex palette. This reduces the overall size of the OpenFlight file by writing only “unique” vertices. Vertex palette records are referenced by faces and light points via vertex list and morph vertex list records. See [Vertex List Record](#) and “Morph Vertex List Record” for more information. Double precision vertex records are stored in a vertex palette for the entire database. Vertices shared by one or more geometric entities are written only one time in the vertex palette. This reduces the overall size of the OpenFlight file by writing only “unique” vertices. Vertex palette records are referenced by faces and light points via vertex list and morph vertex list records. See “[Vertex List Record](#)” and [Morph Vertex List Record](#) for more information.

NOTE

The vertices referenced by mesh nodes are not contained in vertex palette records. Instead, they are contained in local vertex pool records. See [Local Vertex Pool Record](#). The vertex palette record signifies the start of the vertex palette. It contains a one word entry specifying the total length of the vertex palette, which is equal to the length of this header record plus the length of the following vertex records. The individual vertex records follow this header, each starting with its own opcode. The length field in the vertex palette record makes it possible to skip over vertex records until the data is actually needed.

As stated above, vertices may be shared, and are accessed through the vertex and morph vertex list records following each face record. A face may contain all morph vertices, all non-morph vertices, or a mixture of both. Thus there can be one or more list records following each face. Consecutive vertices with the same type are grouped together within a list record. The length of each list record is determined by the number of consecutive vertices of each type. For each vertex, there is a one word field pointing to its vertex record in the vertex palette. Since this offset includes the length of the vertex palette record, the value of the first pointer is 8.

Table 65. Vertex Palette Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Vertex Palette Opcode 67	□
Unsigned Int	2	2	Length - length of the record	□
Int	4	4	Length of this record plus length of the vertex palette	□

The vertex palette record is immediately followed by vertex records. Each vertex record contains all the attributes of a vertex that has been referenced one or more times in the database.

The Color name index references a name in the color name palette.

The Hard edge flag indicates this vertex starts an edge that is to be preserved by polygon reduction or decimation algorithms.

The Normal frozen flag indicates the normal is not to be updated by shading or lighting algorithms.

The No color flag indicates the vertex does not have a color. If set, neither the Packed color or Vertex color index fields are defined.

When a vertex has a color (the No color flag is not set), the Packed color field is always specified (regardless of the value of the Packed color flag) and contains the red, green, blue and alpha color components. For alpha, 0 represents fully transparent, 255 fully opaque. If the Packed color flag is set, the Vertex color index field will be undefined.

Here are some examples that show how vertex palette records can represent vertex colors:

PackedColor Flag	PackedColor	VertexColorIndex	Result
0	a, g, b, r	N	Vertex color index and Packed color attributes are both specified. a, b, g, r specify the vertex color components. g, b, r components match those of color index N in palette.

1	a, g, b, r	Not defined	Vertex color index attribute is not specified, only packed color. a, b, g, r specify the vertex color components.
---	------------	-------------	--

Table 66. Vertex with Color Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Vertex with Color Opcode 68	□
Unsigned Int	2	2	Length - length of the record	□
Unsigned Int	4	2	Color name index	□
Int	6	2	Flags (bits, from left to right)	□
			0 = Start hard edge	□
			1 = Normal frozen	□
			2 = No color	□
			3 = Packed color	□
			4-15 = Spare	□
Double	8	8*3	Vertex coordinate (x, y, z)	□
Int	32	4	Packed color (a, b, g, r) - always specified when the vertex has color	□
Unsigned Int	36	4	Vertex color index - valid only if vertex has color and Packed color flag is not set	□

Table 67. Vertex with Color and Normal Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Vertex with Color and Normal Opcode 69	□

Unsigned Int	2	2	Length - length of the record	
Unsigned Int	4	2	Color name index	
Int	6	2	Flags (bits, from left to right)	
			0 = Start hard edge	
			1 = Normal frozen	
			2 = No color	
			3 = Packed color	
			4-15 = Spare	
Double	8	8*3	Vertex coordinate (x, y, z)	
Float	32	4*3	Vertex normal (i, j, k)	
Int	44	4	Packed color (a, b, g, r) - always specified when the vertex has color	
Unsigned Int	48	4	Vertex color index - valid only if vertex has color and Packed color flag is not set	
Int	52	4	Reserved	

Table 68. Vertex with Color and UV Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Vertex with Color and UV Opcode 71	
Unsigned Int	2	2	Length - length of the record	
Unsigned Int	4	2	Color name index	
Int	6	2	Flags (bits, from left to right)	
			0 = Start hard edge	
			1 = Normal frozen	
			2 = No color	
			3 = Packed color	

			4-15 = Spare	□
Double	8	8*3	Vertex coordinate (x, y, z)	□
Float	32	4*2	Texture coordinate (u, v)	□
Int	40	4	Packed color (a, b, g, r) - always specified when the vertex has color	□
Unsigned Int	44	4	Vertex color index - valid only if vertex has color and Packed color flag is not set	□

Table 69. Vertex with Color, Normal and UV Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Vertex with Color, Normal and UV Opcode 70	□
Unsigned Int	2	2	Length - length of the record	□
Unsigned Int	4	2	Color name index	□
Int	6	2	Flags (bits, from left to right)	□
			0 = Start hard edge	□
			1 = Normal frozen	□
			2 = No color	□
			3 = Packed color	□
			4-15 = Spare	□
Double	8	8*3	Vertex coordinate (x, y, z)	□
Float	32	4*3	Vertex normal (i, j, k)	□
Float	44	4*2	Texture coordinate (u, v)	□

Int	52	4	Packed color (a, b, g, r) - always specified when the vertex has color	🔗
Unsigned Int	56	4	Vertex color index - valid only if vertex has color and Packed color flag is not set	🔗
Int	60	4	Reserved	🔗

Color Palette Record

The color palette record contains all colors indexed by face and vertex nodes in the database.

The color record is divided into two sections: one for color entries and one for color names. All color entries are in 32-bit packed format (a, b, g, r). Each color consists of red, green, and blue components of 8 bits each, plus 8 bits reserved for alpha (future). Currently alpha is always 0xff (fully opaque). The color entry section consists of 1024 ramped colors of 128 intensities each.

The color name section may or may not be included. If the length of the color palette record is greater than 4228, then you can assume that the color name section is included. When it is present, the color name section consists of a header followed by 0 or more color name entries. The header contains the number of names in the palette. If this value is 0, there are no names following in the palette. Each color name entry contains the name string, pointer to the associated color entry, and other reserved information. The name field is a variable-length, null-terminated ASCII string, with a maximum of 80 bytes.

Table 70. Color Palette Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Color Palette Opcode 32	🔗
Unsigned Int	2	2	Length - length of the record	🔗
Char	4	128	Reserved	🔗
Int	132	4	Brightest RGB of color 0, intensity 127 (a, b, g, r)	🔗
Int	136	4	Brightest RGB of color 1, intensity 127 (a, b, g, r)	🔗
etc.		🔗

Int	4224	4	Brightest RGB of color 1023, intensity 127 (a, b, g, r)	□
As stated above, if the length of the color palette record is greater than 4228, then it also contains a color name section as shown below:				□
Int	4228	4	Number of color names	□
The following fields are repeated for each color name entry present in the color palette record.				
In the fields listed below, N ranges from 0 to Number of color names - 1.				
Unsigned Int	Varies	2	LengthN - length of color name subrecord N. This length is the total length of this field plus the length of the next 3 fields plus the length of the Color nameN field.	□
Int	Varies	2	ReservedN - reserved space for color name N	□
Int	Varies	2	Color indexN - index of color in palette corresponding to color name N	□
Int	Varies	2	ReservedN - reserved space for color name N	□
Char	Varies	LengthN-8	Color nameN - color name N; 0 terminates, max 80 bytes	□

Name Table Record

The name table contains a lookup table of names referenced within the database. These names are typically used as attributes (e.g., color name index in the face record). The primary benefit of the name table is to allow name referencing, so each name string is only stored once. Each name entry in the name table contains fields for its length, index, and string. The name index is used by the

database to reference names within the table. The name string is a variable-length, null-terminated ASCII string, with a maximum of 80 bytes.

Table 71. Name Table Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Name Table Opcode 114	□
Unsigned Int	2	2	Length - length of the record	□
Int	4	4	Number of names	□
Unsigned Int	8	2	Next available name index	□
Name Table Entry 0				
Int	10	4	Length0 - length of entry 0	□
Unsigned Int	14	2	Name index0 - index corresponding to entry 0	□
Char	16	Varies	Name string0 - name for entry 0; 0 terminates. Variable length, maximum of 80 chars	□
Name Table Entry 1				
Int	Varies	4	Length1 - length of entry 1	□
Unsigned Int	Varies	2	Name index1 - index corresponding to entry 1	□
Char	Varies	Varies	Name string1 - name for entry 1; 0 terminates. Variable length, maximum of 80 chars	□
...	
Name Table Entry N, where N is Number of names - 1				

Int	Varies	4	LengthN - length of entry N	□
Unsigned Int	Varies	2	Name indexN - index corresponding to entry N	□
Char	Varies	Varies	Name stringN - name for entry N; 0 terminates. Variable length, maximum of 80 chars	□

Material Palette Record

The material palette contains descriptions of materials used while drawing geometry. It is composed of an arbitrary number of material palette records. The material palette records must follow the header record and precede the first push.

The appearance of a face or mesh in OpenFlight is a combination of the geometry (face or mesh) color and the material properties. The geometry color is factored into the material properties as follows:

Ambient:

The displayed material's ambient component is the product of the ambient component of the material and the geometry color:

$$\begin{aligned} \text{Displayed ambient (red)} &= \text{Material ambient (red)} * \text{geometry color (red)} \\ \text{Displayed ambient (green)} &= \text{Material ambient (green)} * \text{geometry color (green)} \\ \text{Displayed ambient (blue)} &= \text{Material ambient (blue)} * \text{geometry color (blue)} \end{aligned}$$

For example, suppose the material has an ambient component of {1.0,.5,.5} and the geometry color is {100, 100, 100}. The displayed material has as its ambient color {100, 50, 50}.

Diffuse:

As with the ambient component, the diffuse component is the product of the diffuse component of the material and the geometry color:

$$\begin{aligned} \text{Displayed diffuse (red)} &= \text{Material diffuse (red)} * \text{geometry color (red)} \\ \text{Displayed diffuse (green)} &= \text{Material diffuse (green)} * \text{geometry color (green)} \\ \text{Displayed diffuse (blue)} &= \text{Material diffuse (blue)} * \text{geometry color (blue)} \end{aligned}$$

Specular:

Unlike ambient and diffuse components, the displayed specular component is taken directly from the material:

Displayed specular (red) = Material specular (red)
 Displayed specular (green) = Material specular (green)
 Displayed specular (blue) = Material specular (blue)

Emissive:

The displayed emissive component is taken directly from the material:

Displayed emissive (red) = Material emissive (red)
 Displayed emissive (green) = Material emissive (green)
 Displayed emissive (blue) = Material emissive (blue)

Shininess:

The MultiGen-Paradigm, Inc. modeling environment uses the shininess directly from the material. Specular highlights are tighter, with higher shininess values.

Alpha:

An alpha of 1.0 is fully opaque, while 0.0 is fully transparent. The final alpha applied is a combination of the transparency value of the geometry (face or mesh) with the alpha value of the material record. The final alpha value is a floating point number between 0.0 (transparent) and 1.0 (opaque), and is computed as follows:

```
Final alpha = material alpha * (1.0 - (geometry transparency / 65535))
```

Table 72. Material Palette Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Material Palette Opcode 113	□
Int	2	2	Length - length of the record	□
Int	4	4	Material index	□
Char	8	12	Material name	□
Int	20	4	Flags	□
			0 = Material is used	□
			1-31 = Spare	□
Float	24	4*3	Ambient component of material (r, g, b) *	□

Float	36	4*3	Diffuse component of material (r, g, b) *	□
Float	48	4*3	Specular component of material (r, g, b) *	□
Float	60	4*3	Emissive component of material (r, g, b) *	□
Float	72	4	Shininess - (0.0-128.0)	□
Float	76	4	Alpha - (0.0-1.0) where 1.0 is opaque	□
Int	80	4	Reserved	□

- normalized values between 0.0 and 1.0, inclusive.

Texture Palette Record

There is one record for each texture pattern referenced in the database. These records must follow the header record and precede the first push.

A palette and pattern system can be used to reference the texture patterns. A texture palette is made up of 256 patterns. The pattern index for the first palette is 0 - 255, for the second palette 256 - 511, etc. Note: If less than 256 patterns exist on a palette, several pattern indices are unused. The x and y palette locations are used to store offset locations in the palette for display.

Table 73. Texture Palette Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Texture Palette Opcode 64	□
Unsigned Int	2	2	Length - length of the record	□
Char	4	200	File name of texture pattern	□
Int	204	4	Texture pattern index	□
Int	208	4*2	Location in the texture palette (x, y)	□

Eyepoint and Trackplane Palette Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider the Eyepoint and Trackplane Palette Records.

Table 74. Eyepoint and Trackplane Palette Record

Data Type	Offset	Length	Description
Int	0	2	Eyepoint and Trackplane Palette Opcode 83
Unsigned Int	2	2	Length - length of the record
Int	4	4	Reserved

The following fields are repeated for 10 eyepoints

Eyepoint 0 - 272 bytes

Double	8	8*3	Rotation center (x, y, z)
Float	32	4*3	Yaw, pitch, and roll angles
Float	44	16*4	4x4 rotation matrix, row major order
Float	108	4	Field of view
Float	112	4	Scale
Float	116	4	Near clipping plane
Float	120	4	Far clipping plane
Float	124	16*4	4x4 fly-through matrix, row major order
Float	188	3*4	Eyepoint position (x, y, z)
Float	200	4	Yaw of fly-through
Float	204	4	Pitch of fly-through
Float	208	3*4	Eyepoint direction vector (i, j, k)
Int	220	4	No fly through - 1 if no fly-through
Int	224	4	Ortho view - 1 if ortho drawing mode
Int	228	4	Valid eyepoint - 1 if this is a valid eyepoint
Int	232	4	Image offset x

Int	236	4	Image offset y
Int	240	4	Image zoom
Int	244	4*8	Reserved
Int	276	4	Reserved
Eyepoint 1	280	272	Eyepoint 1 - the fields listed above are repeated here.
Eyepoint 2	552	272	Eyepoint 2 - the fields listed above are repeated here.
Eyepoint 3	824	272	Eyepoint 3 - the fields listed above are repeated here.
Eyepoint 4	1096	272	Eyepoint 4 - the fields listed above are repeated here.
Eyepoint 5	1368	272	Eyepoint 5 - the fields listed above are repeated here.
Eyepoint 6	1640	272	Eyepoint 6 - the fields listed above are repeated here.
Eyepoint 7	1912	272	Eyepoint 7 - the fields listed above are repeated here.
Eyepoint 8	2184	272	Eyepoint 8 - the fields listed above are repeated here.
Eyepoint 9	2456	272	Eyepoint 9 - the fields listed above are repeated here.

Table 75. Eyepoint and Trackplane Palette Record (Continued)

Data Type	Offset	Length	Description
The following fields are repeated for 10 trackplanes			
Trackplane 0 - 128 bytes			
Int	2728	4	Valid trackplane - 1 if this is a valid trackplane
Int	2732	4	Reserved

Double	2736	8*3	Trackplane origin coordinate (x, y, z)
Double	2760	8*3	Trackplane alignment coordinate (x, y, z)
Double	2784	8*3	Trackplane plane coordinate (x, y, z)
Boolean	2808	1	Grid visible - 1 if grid is visible
Int	2809	1	Grid type flag 0 = rectangular grid 1 = radial grid
Int	2810	1	Grid under flag 0 = draw grid over scene 1 = draw grid under scene 2 = draw grid depth buffered
Int	2811	1	Reserved
Float	2812	4	Grid angle for radial grid
Double	2816	8	Grid spacing in X. Radius if radial grid.
Double	2824	8	Grid spacing in Y
Int	2832	1	Radial grid spacing direction control
Int	2833	1	Rectangular grid spacing direction control
Boolean	2834	1	Snap cursor to grid - 1 if snap cursor to grid is on
Int	2835	1	Reserved
Int	2836	4	Reserved
Double	2840	8	Grid size (a power of 2)
Boolean	2848	4	Mask of visible grid quadrants
Int	2852	4	Reserved

Trackplane 1	2856	128	Trackplane 1 - the fields listed above are repeated here.
Trackplane 2	2984	128	Trackplane 2 - the fields listed above are repeated here.
Trackplane 3	3112	128	Trackplane 3 - the fields listed above are repeated here.
Trackplane 4	3240	128	Trackplane 4 - the fields listed above are repeated here.
Trackplane 5	3368	128	Trackplane 5 - the fields listed above are repeated here.
Trackplane 6	3496	128	Trackplane 6 - the fields listed above are repeated here.
Trackplane 7	3624	128	Trackplane 7 - the fields listed above are repeated here.
Trackplane 8	3752	128	Trackplane 8 - the fields listed above are repeated here.
Trackplane 9	3880	128	Trackplane 9 - the fields listed above are repeated here.

Key Table Records

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider the Key Table Records.

Key table records store variable length data records and their identifiers. The linkage editor, sound palette, and CAT Data are stored as key table records. The first key table record contains the key table header and a set of keys. If all the keys cannot fit into the first record, additional key records are written. This is followed by one or more key table data records.



A key table consists of: For an example of the use of key table records, see [Sound Palette Record](#).

Table 76. Key Table Header Record

Data Type	Offset	Length	Description
Int	0	2	Opcode - opcode of record using key table for storage
Unsigned Int	2	2	Length - length of the record
Int	4	4	Subtype 1 = indicates this record is a key table header
Int	8	4	Max number - maximum number of entries
Int	12	4	Actual number - actual number of entries
Int	16	4	Total length of packed data
Int	20	4*3	Reserved
The following fields are repeated for each key in the key table. In the fields listed below, N ranges from 0 to Actual number - 1.	Int	32+(N*12)	4
Key valueN - key value N	Int	36+(N*12)	4
ReservedN - reserved space for key N, defined by record using key table for storage	Int	40+(N*12)	4

Table 77. Key Table Data

Data Type	Offset	Length	Description
Int	0	2	Opcode - opcode of record using key table for storage
Unsigned Int	2	2	Length - length of the record
Int	4	4	Subtype
			2 = indicates this record is a key table data record
Int	8	4	Data length
Char	12	Data length	Packed data Data is always 4 byte aligned, with unused bytes set to 0. Data length can be calculated as follows: Length - 12

Linkage Palette Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider the Linkage Palette Records.

Database linkages use key table records. Linkage data consists of two different constructs: nodes and arcs. Nodes usually contain data pertaining to database entities such as DOFs. In addition, the nodes may represent modeling driver functions and code nodes. The arcs contain information on how all the nodes are connected to each other. For most nodes, the value of the node is contained in the following Entity name subrecord. For example, this node value can be a node name, when the node represents a database entity, or a math formula as a string, in the case of a formula node. Names are stored as null-terminated ASCII strings.

See [Linkage Editor Parameter IDs](#) for parameter ID values and descriptions.

Table 78. Linkage Palette Header Record

Data Type	Offset	Length	Description
Int	0	2	Linkage Palette Opcode 90
Unsigned Int	2	2	Length - length of the record
Int	4	4	Subtype

			1 = indicates this record is a key table header
Int	8	4	Max number - maximum number of entries. Each entry is either a node, arc, or entity name.
Int	12	4	Actual number - actual number of entries. Each entry is either a node, arc, or entity name.
Int	16	4	Total length of data
Int	20	4*3	Reserved

The following fields are repeated for each key in the key table.

In the fields listed below, N ranges from 0 to Actual number - 1.

Int	32+(N*12)	4	Key valueN - key value N
Int	36+(N*12)	4	Data typeN - data type for key N
			0x12120001 = Node data
			0x12120002 = Arc data
			0x12120004 = Database entity name
Int	40+(N*12)	4	Data offsetN - offset for data corresponding to key N. Note: This offset is measured relative to the Packed data field in the linkage palette data record described below.

Table 79. Linkage Palette Data Record

Data Type	Offset	Length	Description
Int	0	2	Linkage Palette Opcode 90
Unsigned Int	2	2	Length - length of the record
Int	4	4	Subtype

			2 = indicates this record is a key data record
Int	8	4	Data length
Char	12	Data length	Packed data. Each packed data item is either a node data subrecord, arc data subrecord or entity name subrecord. Node data subrecords can be either general nodes, formula nodes, or driver nodes. All these subrecords are described in the following sections. Data length can be calculated as follows: Length - 12

The offsets listed in the following subrecords are measured from the start of the subrecord, not from the start of the linkage palette data record that contains this packed data.

Table 80. General Node Data Subrecord

Data Type	Offset	Length	Description
Int	0	4	Identifier
Int	4	4	Reserved
Int	8	4	Node type
			0x12120003 = Header node
			0x12120005 = Database entity node
Int	12	4*4	Reserved
Int	28	4	Sinks
Int	32	4	Sources
Int	36	4	Next node identifier
Int	40	4	Previous node identifier
Int	44	4	Arc source identifier
Int	48	4	Arc sink identifier

Table 81. Formula Node Data Subrecord

Data Type	Offset	Length	Description
Int	0	4	Identifier
Int	4	4	Reserved
Int	8	4	Data type
			0x12150000 = Formula node
Int	12	4	Reserved
Int	16	4	Reserved
Int	20	4	Reserved
Int	24	4	Reserved
Int	28	4	Sinks
Int	32	4	Sources
Int	36	4	Next node identifier
Int	40	4	Previous node identifier
Int	44	4	Arc source identifier
Int	48	4	Arc sink identifier
Int	52	4	Reserved

Table 82. Formula Node Data Subrecord (Continued)

Data Type	Offset	Length	Description
Int	56	4	Reserved
Int	60	4	Reserved
Int	64	4	Reserved
Int	68	4	Reserved
Int	72	4	Reserved
Int	76	4	Reserved
Int	80	4	Reserved

Table 83. Driver Node Data Subrecord

Data Type	Offset	Length	Description
Int	0	4	Identifier
Int	4	4	Reserved
Int	8	4	Node type
			0x12140001 = Ramp driver node

			0x12140004 = Variable driver node
			0x12140005 = External file driver node
Int	12	4	Reserved
Int	16	4	Reserved
Int	20	4	Reserved
Int	24	4	Reserved
Int	28	4	Sinks
Int	32	4	Sources
Int	36	4	Next node identifier
Int	40	4	Previous node identifier
Int	44	4	Arc source identifier
Int	48	4	Arc sink identifier
Float	52	4	Current value
Float	56	4	Min amplitude
Float	60	4	Max amplitude
Float	64	4	Wave offset
Float	68	4	Min time
Float	72	4	Max time
Float	76	4	Time steps
Int	80	4	Reserved
Int	84	4	Reserved
Int	88	4	Reserved
Int	92	4	Reserved

Table 84. Arc Data Subrecord

Data Type	Offset	Length	Description
Int	0	4	Identifier
Int	4	4	Reserved
Int	8	4	Data type
			0x12120002 = Arc data subrecord
Int	12	4	Reserved
Int	16	4	Reserved

Int	20	4	Priority
-----	----	---	----------

Table 85. Arc Data Subrecord (Continued)

Data Type	Offset	Length	Description
Int	24	4	Source parameter - parameter ID if source node is a node
Int	28	4	Sink parameter - parameter ID if sink node is a node
			number (0...7) for variables (x1...x8)
			Only valid if sink node is a formula
Int	32	4	Reserved
Int	36	4	Next source identifier
Int	40	4	Next sink identifier
Int	44	4	Node source identifier
Int	48	4	Node sink identifier

Table 86. Entity Name Subrecord

Data Type	Offset	Length	Description
Char	0	Variable	ASCII string; 0 terminates

Sound Palette Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider the Sound Palette Records.

The sound palette uses key table records to store the sound index and file name. The index is the key value, and the file name is the data record, formatted as a null-terminated ASCII string. The sound palette header record indicates the number of sounds associated with the database.

Table 87. Sound Palette Header Record

Data Type	Offset	Length	Description
Int	0	2	Sound Palette Opcode 93
Unsigned Int	2	2	Length - length of the record
Int	4	4	Subtype

			1 = indicates this record is a key table header
Int	8	4	Max number - the maximum number of sounds in palette
Int	12	4	Actual number - the actual number of sounds in palette
Int	16	4	Total length - total length of the sound file names contained in the sound palette key data record, which follows this record and is described below
Int	20	4*3	Reserved

The following fields are repeated for each sound represented in the palette.

In the fields listed below, N ranges from 0 to Actual number - 1.

Int	32+(N*12)	4	Sound indexN - index of sound N in the palette
Int	36+(N+12)	4	ReservedN - reserved space for sound N in the palette
Int	40+(N*12)	4	File name offsetN - starting offset for file name of sound N in the palette. This offset is measured relative to the Packed file names field in the sound palette data record described below.

Table 88. Sound Palette Data Record

Data Type	Offset	Length	Description
Int	0	2	Sound Palette Opcode 93
Unsigned Int	2	2	Length - length of the record
Int	4	4	Subtype

			2 = indicates this record is a key data record
Int	8	4	Total length of all packed sound file names
Char	12	Data length	Packed file names. Use File name offsets contained in sound palette key table header to locate individual names in this data blocks. Data length can be calculated as follows: Length - 12

Light Source Palette Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider the Light Source Records.

These records represent entries in the light source palette. Entries are referenced by light source nodes using the palette index. Lights can be flagged as modeling lights, which illuminate a scene without being stored as part of the hierarchy. A modeling light is always positioned at the eye; its direction is stored in the palette. A light referenced by a node obtains its position and direction from the node. In this case, the palette yaw and pitch components are ignored.

Table 89. Light Source Palette Record

Data Type	Offset	Length	Description
Int	0	2	Light Source Palette Opcode 102
Unsigned Int	2	2	Length - length of the record
Int	4	4	Light source index
Int	8	2*4	Reserved
Char	16	20	Light source name; 0 terminates
Int	36	4	Reserved
Float	40	4*4	Ambient component of light source (r, g, b, a) - alpha unused
Float	56	4*4	Diffuse component of light source (r, g, b, a) - alpha unused

Float	72	4*4	Specular component of light source (r, g, b, a) - alpha unused
Int	88	4	Light type
			0 = Infinite
			1 = Local
			2 = Spot
Int	92	4*10	Reserved
Float	132	4	Spot exponential drop-off term
Float	136	4	Spot cutoff angle (in degrees)
Float	140	4	Yaw
Float	144	4	Pitch
Float	148	4	Constant attenuation coefficient
Float	152	4	Linear attenuation coefficient
Float	156	4	Quadratic attenuation coefficient
Int	160	4	Modeling light
			0 = Light source is not active during modeling
			1 = Light source is active during modeling
Int	164	4*19	Reserved

Light Point Appearance Palette Record

The light point appearance palette record defines the visual attributes of light points.

Table 90. Light Point Appearance Palette Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Light Point Appearance Palette Opcode 128	□
Unsigned Int	2	2	Length - length of the record	□
Int	4	4	Reserved	□

Char	8	256	Light Point Type Name 0 terminates	🔗
Int	264	4	Appearance	🔗
Int	268	2	Surface material code	🔗
Int	270	2	Feature ID	🔗
Unsigned Int	272	4	Back color for bidirectional points	🔗
Int	276	4	Display mode	🔗
			0 = RASTER	🔗
			1 = CALLIGRAPHIC	🔗
			2 = EITHER	🔗
Float	280	4	Intensity - scalar for front colors	🔗
Float	284	4	Back intensity - scalar for back color	🔗
Float	288	4	Minimum defocus - (0.0 - 1.0) for calligraphic points	🔗
Float	292	4	Maximum defocus - (0.0 - 1.0) for calligraphic points	🔗
Int	296	4	Fading mode	🔗
			0 = Enable perspective fading calculations	🔗
			1 = Disable calculations	🔗
Int	300	4	Fog Punch mode	🔗
			0 = Enable fog punch through calculations	🔗
			1 = Disable calculations	🔗
Int	304	4	Directional mode	🔗

			0 = Enable directional calculations	🔗
			1 = Disable calculations	🔗
Int	308	4	Range mode	🔗
			0 = Use depth (Z) buffer calculation	🔗
			1 = Use slant range calculation	🔗
Float	312	4	Min pixel size - minimum diameter of points in pixels	🔗
Float	316	4	Max pixel size - maximum diameter of points in pixels	🔗
Float	320	4	Actual size - actual diameter of points in database units	🔗
Float	324	4	Transparent falloff pixel size - diameter in pixels when points become transparent	🔗
Float	328	4	Transparent falloff exponent	🔗
			≥ 0 - falloff multiplier exponent	🔗
			1.0 - linear falloff	🔗
Float	332	4	Transparent falloff scalar	🔗
			> 0 - falloff multiplier scale factor	🔗
Float	336	4	Transparent falloff clamp - minimum permissible falloff multiplier result	🔗

Float	340	4	Fog scalar >= 0 - adjusts range of points for punch threw effect.	🔗
Float	344	4	Fog intensity	🔗
Float	348	4	Size difference threshold - point size transition hint to renderer	🔗
Int	352	4	Directionality 0 = OMNIDIRECTIONAL 1 = UNIDIRECTIONAL 2 = BIDIRECTIONAL	🔗
Float	356	4	Horizontal lobe angle - total angle in degrees	🔗
Float	360	4	Vertical lobe angle - total angle in degrees	🔗
Float	364	4	Lobe roll angle - rotation of lobe about local Y axis in degrees	🔗
Float	368	4	Directional falloff exponent >= 0 - falloff multiplier exponent	🔗
			1.0 - linear falloff	🔗
Float	372	4	Directional ambient intensity - of points viewed off axis	🔗

Float	376	4	Significance - drop out priority for RASCAL lights (0.0 - 1.0)	<input type="checkbox"/>
Int	380	4	Flags (bits, from left to right)	<input type="checkbox"/>
			0 = reserved	<input type="checkbox"/>
			1 = No back color	<input type="checkbox"/>
			TRUE = don't use back color for bidirectional points	<input type="checkbox"/>
			FALSE = use back color for bidirectional points	<input type="checkbox"/>
			2 = reserved	<input type="checkbox"/>
			3 = Calligraphic proximity occulting (Debunching)	<input type="checkbox"/>
			4 = Reflective, non-emissive point	<input type="checkbox"/>
			5-7 = Randomize intensity	<input type="checkbox"/>
			0 = never	<input type="checkbox"/>
			1 = low	<input type="checkbox"/>
			2 = medium	<input type="checkbox"/>
			3 = high	<input type="checkbox"/>
			8 = Perspective mode	<input type="checkbox"/>
			9 = Flashing	<input type="checkbox"/>
			10 = Rotating	<input type="checkbox"/>
			11 = Rotate Counter Clockwise	<input type="checkbox"/>
			Direction of rotation about local Z axis	<input type="checkbox"/>
			12 = reserved	<input type="checkbox"/>
			13-14 = Quality	<input type="checkbox"/>

			0 = Low	<input type="checkbox"/>
			1 = Medium	<input type="checkbox"/>
			2 = High	<input type="checkbox"/>
			3 = Undefined	<input type="checkbox"/>
			15 = Visible during day	<input type="checkbox"/>
			16 = Visible during dusk	<input type="checkbox"/>
			17 = Visible during night	<input type="checkbox"/>
			18-31 = Spare	<input type="checkbox"/>
Float	384	4	Visibility range (> 0.0)	<input type="checkbox"/>
Float	388	4	Fade range ratio - percentage of total range at which light points start to fade (0.0 - 1.0)	<input type="checkbox"/>
Float	392	4	Fade in duration - time it takes (seconds) light point to fade in when turned on	<input type="checkbox"/>
Float	396	4	Fade out duration - time it takes (seconds) light point to fade out when turned off	<input type="checkbox"/>
Float	400	4	LOD range ratio - percentage of total range at which light points LODs are active (0.0 - 1.0)	<input type="checkbox"/>
Float	404	4	LOD scale - size of light point LOD polygon relative to light point diameter	<input type="checkbox"/>
Int	408	2	Texture pattern index, -1 if none	<input type="checkbox"/>

Int	410	2	Reserved	□
-----	-----	---	----------	---

Light Point Animation Palette Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider the Light Point Animation Palette Records.

The light point animation palette record defines the behavioral attributes of light points.

Table 91. Light Point Animation Palette Record

Data Type	Offset	Length	Description
Int	0	2	Light Point Animation Opcode 129
Unsigned Int	2	2	Length - length of the record
Int	4	4	Reserved
char	8	256	Animation name; 0 terminates
Int	264	4	Animation index
Float	268	4	Animation period in seconds. Note: Rate = 1/Period
Float	272	4	Animation phase delay in seconds - from start of period
Float	276	4	Animation enabled period (time on) in seconds
Float	280	4*3	Axis of rotation for rotating animation (i, j, k)
Int	292	4	Flags (bits, from left to right)
			0 = Flashing
			1 = Rotating
			2 = Rotate counter clockwise
			3-31 = Spare
Int	296	4	Animation type
			0 = Flashing sequence
			1 = Rotating

			2 = Strobe
			3 = Morse code
Int	300	4	Morse code timing
			0 = Standard timing
			1 = Farnsworth timing
Int	304	4	Word rate (for Farnsworth timing)
Int	308	4	Character rate (for Farnsworth timing)
char	312	1024	Morse code string
Int	1336	4	Number of sequences (for Flashing sequence)

The following fields are repeated for each sequence represented in the light point animation palette entry.

In the fields listed below, N ranges from 0 to Number of sequences - 1.

Unsigned Int	1340+(N*12)	4	Sequence StateN - state of sequence N
			0 = On
			1 = Off
			2 = Color change
Float	1344+(N*12)	4	Sequence DurationN - duration of sequence N in seconds
Unsigned Int	1348+(N*12)	4	Sequence ColorN - color for sequence N. Defined if Sequence state is On or Color change

Line Style Palette Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider the Line Style Palette Records.

Line style records define the outline displayed around faces in wireframe or wireframe-over-solid mode. The Pattern field defines a mask to control the display of segments of the line. For example, if all the bits of the mask are set, the line is drawn as a solid line. If every other bit is on, the line is displayed as a dashed line. The Line Width field controls the width of the line in pixels. Line style 0 is the default. Faces are assigned line styles in the Line Style field of the face record. One of these records appears for each line style defined in the OpenFlight file.

Table 92. Line Style Palette Record

Data Type	Offset	Length	Description
Int	0	2	Line Style Palette Record Opcode 97
Int	2	2	Length of record
Int	4	2	Line style index
Int	6	2	Pattern mask
Int	8	4	Line width

Texture Mapping Palette Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider any of the Texture Mapping Palette Records.

The texture mapping palette record defines methods and parameters used to map textures onto geometry. One record is created for each texture mapping reference in the palette. These records must follow the header record and precede the first push.

Table 93. Texture Mapping Palette Record

Data Type	Offset	Length	Description
Int	0	2	Texture Mapping Palette Opcode 112
Int	2	2	Length - length of the record
Int	4	4	Reserved
Int	8	4	Texture mapping index
Char	12	20	Texture mapping name
Int	32	4	Texture mapping type
			0 = None
			1 = Put
			2 = 4 Point Put
			3 = Reserved
			4 = Spherical Project
			5 = Radial Project
			6 = Reserved
Int	36	4	Warped flag; if TRUE, 8 point warp applied
Double	40	8*16	4x4 Transformation matrix (for types 1, 2, 4 & 5), row major

The parameters for put texture mapping will appear immediately following the texture mapping palette record if Texture mapping type is 1.

Table 94. Parameters for Put Texture Mapping (Type 1)

Data Type	Offset	Length	Description
Int	168	4	State of Put Texture tool
			0 = Start state - no points entered
			1 = One point entered
			2 = Two points entered
			3 = Three points entered
Int	172	4	Active geometry point
			1 = Origin point
			2 = Alignment point
			3 = Shear point
Double	176	8*3	Lower-left corner of bounding box for geometry using this mapping when mapping was created (x, y, z)
Double	200	8*3	Upper-right corner of bounding box for geometry using this mapping when mapping was created (x, y, z)
Int	224	4*3	Use real world size flags for each of the put points
Int	236	4	Reserved
Double	240	8*3	Texture origin point (x, y, z)
Double	264	8*3	Texture alignment point (x, y, z)
Double	288	8*3	Texture shear point (x, y, z)
Double	312	8*3	Geometry origin point (x, y, z)

Double	336	8*3	Geometry alignment point (x, y, z)
Double	360	8*3	Geometry shear point (x, y, z)
Int	384	4	Active texture point 1 = Origin point 2 = Alignment point 3 = Shear point
Int	388	4	UV display type 1 = XY 2 = UV
Float	392	4	U Repetition
Float	396	4	V Repetition

The parameters for 4 point put texture mapping will appear immediately following the texture mapping palette record if Texture mapping type is 2

Table 95. Parameters for 4 Point Put Texture Mapping (Type 2)

Data Type	Offset	Length	Description
Int	168	4	State of 4 Point Put Texture tool 0 = Start state - no points entered 1 = One point entered 2 = Two points entered 3 = Three points entered 4 = Four points entered
Int	172	4	Active geometry point 1 = Origin point 2 = Alignment point 3 = Shear point 4 = Perspective point
Double	176	8*3	Lower-left corner of bounding box for geometry using this mapping when mapping was created (x, y, z)

Double	200	8*3	Upper-right corner of bounding box for geometry using this mapping when mapping was created (x, y, z)
Int	224	3*4	Use real world size flags for each of the put points
Int	236	4	Reserved
Double	240	8*3	Texture origin point (x, y, z)
Double	264	8*3	Texture alignment point (x, y, z)
Double	288	8*3	Texture shear point (x, y, z)
Double	312	8*3	Texture perspective point (x, y, z)
Double	336	8*3	Geometry origin point (x, y, z)
Double	360	8*3	Geometry alignment point (x, y, z)
Double	384	8*3	Geometry shear point (x, y, z)
Double	408	8*3	Geometry perspective point (x, y, z)
Int	432	4	Active texture point
			1 = Origin point
			2 = Alignment point
			3 = Shear point
			4 = Perspective point
Int	436	4	UV display type
			1 = XY
			2 = UV
Float	440	4	Depth scale factor
Int	444	4	Reserved

Double	448	8*16	4x4 Transformation matrix for the 4 point projection plane, row major order
Float	576	4	U Repetition
Float	580	4	V Repetition

The parameters for spherical project mapping will appear immediately following the texture mapping palette record if Texture mapping type is 4.

Table 96. Parameters for Spherical Project Mapping (Type 4)

Data Type	Offset	Length	Description
Float	168	4	Scale
Double	172	8*3	Center of the projection sphere (x, y, z)
Float	196	4	Scale / (maximum dimension of the mapped geometry bounding box)
Float	200	4	Maximum dimension of the mapped geometry bounding box when mapping was created

The parameters for radial project mapping will appear immediately following the texture mapping palette record if Texture mapping type is 5.

Table 97. Parameters for Radial Project Mapping (Type 5)

Data Type	Offset	Length	Description
Int	168	4	Active geometry point
			1 = End point 1 of cylinder center line
			2 = End point 2 of cylinder center line
Int	172	4	Reserved
Float	176	4	Radial scale
Float	180	4	Scale along length of cylinder

Double	184	8*16	4x4 Trackplane to XY plane transformation matrix, row major order
Double	312	8*3	End point 1 of cylinder center line (x, y, z)
Double	336	8*3	End point 2 of cylinder center line (x, y, z)

The parameters for warped mapping will be included if the Warped flag is set in the texture mapping palette record. This parameter block will appear immediately following the texture mapping parameter block to which the warp applies. In the offset fields below, X is equal to the size of the texture mapping palette record plus the size of the texture mapping parameter block to which the warp applies.

Table 98. Parameters for Warped Mapping (Warped Flag Set)

Data Type	Offset	Length	Description
Int	X+0	4	Active geometry point
			0 = First warp FROM point
			1 = Second warp FROM point
			2 = Third warp FROM point
			3 = Fourth warp FROM point
			4 = Fifth warp FROM point
			5 = Sixth warp FROM point
			6 = Seventh warp FROM point
			7 = Eighth warp FROM point
			8 = First warp TO point
			9 = Second warp TO point
			10 = Third warp TO point
			11 = Fourth warp TO point

			12 = Fifth warp TO point
			13 = Sixth warp TO point
			14 = Seventh warp TO point
			15 = Eighth warp TO point
Int	X+4	4	Warp tool state
			0 = Start state - no points entered
			1 = One FROM point entered
			2 = Two FROM point entered
			3 = Three FROM point entered
			4 = Four FROM point entered
			5 = Five FROM point entered
			6 = Six FROM point entered
			7 = Seven FROM point entered
			8 = All FROM point entered
Int	X+8	8	Reserved
Double	X+16	8*8*2	FROM points transformed to XY plane by above matrix. 8 FROM points are ordered 1, 2, ... 8. Each point is (x, y)
Double	X+144	8*8*2	TO points transformed to XY plane by above matrix. 8 TO points are ordered 1, 2, ... 8. Each point is (x, y)

Shader Palette Record

The shader palette contains descriptions of shaders used while drawing geometry. It is composed of an arbitrary number of shader palette records. The shader palette records must follow the header record and precede the first push.

Table 99. Shader Palette Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Shader Opcode 133	□
Unsigned Int	2	2	Length - length of the record	□
Int	4	4	Shader index	□
Int	8	4	Shader type	□
			0 = Cg	□
			1 = CgFX	□
			2 = OpenGL Shading Language	□
char	12	1024	Shader name; 0 terminates	□
char	1036	1024	Vertex program file name; 0 terminates (Cg Shader type specific)	□
char	2060	1024	Fragment program file name; 0 terminates (Cg Shader type specific)	□
Int	3084	4	Vertex program profile (Cg Shader type specific)	□
Int	3088	4	Fragment program profile (Cg Shader type specific)	□
char	3092	256	Vertex program entry point (Cg Shader type specific)	□

char	3348	256	Fragment program entry point (Cg Shader type specific)	□
------	------	-----	---	---

B.13. Texture Files

B.13.1. Texture Pattern Files

OpenFlight does not have its own texture pattern format, but rather uses existing texture formats and references patterns by file name. See [Texture Palette Record](#).

File formats currently supported include:

•AT&T® image 8 format (8-bit color lookup)	CDB OpenFlight Readers
•AT&T image 8 template format	□
•SGI intensity modulation (*.int)	□ [32]
•SGI intensity modulation with alpha (*.inta)	□ 8
•SGI RGB (*.rgb)	□
•SGI RGB with alpha (*.rgba)	□ 8
•GIF	□
•JPEG/JFIF (*.jpg)	□
•TIFF (*.tif)	□
•IFF/ILBM	□
•BMP/DIB	□
•PCX	□
•PNG	□
•PPM	□
•Sun™ Raster	□
•Direct Draw Surface (DDS)	□
•Targa™	□
•Alias™ Pix	□
•SGI clip texture	□

The format of the file is determined by the file name extension, the magic numbers within the file, or the texture attribute file, as described in the following section.

B.13.2. Texture Attribute Files

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider texture attribute

(*.attr) files.

A corresponding attribute file is created for each texture pattern, with the name of the attribute file the same as the texture file, followed by the extension “.attr”. These attribute files are used by the modeling software, and may not be necessary for the application using the database.

The attribute file contains information specifying how to parse the texture pattern file, set the texture hardware and software environment for the texture pattern, or position the image in a database.

The format of the texture attribute file is described in this section.

Table 100. Texture Attribute File Format

Data Type	Offset	Length	Description
Int	0	4	Number of texels in u direction
Int	4	4	Number of texels in v direction
Int	8	4	Real world size u direction (obsolete - not used)
Int	12	4	Real world size v direction (obsolete - not used)
Int	16	4	x component of up vector
Int	20	4	y component of up vector
Int	24	4	File format type
			0 = AT&T image 8 pattern
			1 = AT&T image 8 template
			2 = SGI intensity modulation
			3 = SGI intensity w/alpha
			4 = SGI RGB
			5 = SGI RGB w/alpha
Int	28	4	Minification filter type
			0 = Point
			1 = Bilinear

			2 = Mipmap (obsolete)
			3 = Mipmap Point
			4 = Mipmap linear
			5 = Mipmap bilinear
			6 = Mipmap trilinear
			7 = None
			8 = Bicubic
			9 = Bilinear greater/equal
			10 = Bilinear less/equal
			11 = Bicubic greater/equal
			12 = Bicubic less/equal
Int	32	4	Magnification filter type
			0 = Point
			1 = Bilinear
			2 = None
			3 = Bicubic
			4 = Sharpen
			5 = Add Detail
			6 = Modulate Detail
			7 = Bilinear greater/equal
			8 = Bilinear less/equal
			9 = Bicubic greater/equal
			10 = Bicubic less/equal
Int	36	4	Wrap method u,v - only used when either Wrap method u or Wrap method v is set to None
			0 = Repeat
			1 = Clamp
			4 = Mirrored Repeat
Int	40	4	Wrap method u

			0 = Repeat
			1 = Clamp
			3 = None - use Wrap method u,v
			4 = Mirrored Repeat
Int	44	4	Wrap method v
			0 = Repeat
			1 = Clamp
			3 = None - use Wrap method u,v
			4 = Mirrored Repeat
Int	48	4	Modified flag - for internal use only
Int	52	4	x pivot point for rotating textures
Int	56	4	y pivot point for rotating textures
Int	60	4	Environment type
			0 = Modulate
			1 = Blend
			2 = Decal
			3 = Replace
			4 = Add
Int	64	4	TRUE if intensity pattern to be loaded in alpha with white in color
Int	68	4*8	Reserved
Double	100	8	Real world size u direction
Double	108	8	Real world size v direction
Int	116	4	Code for origin of imported texture
Int	120	4	Kernel version number
Int	124	4	Internal format type
			0 = Default

			1 = TX_I_12A_4
			2 = TX_IA_8
			3 = TX_RGB_5
			4 = TX_RGBA_4
			5 = TX_IA_12
			6 = TX_RGBA_8
			7 = TX_RGBA_12
			8 = TX_I_16 (shadow mode only)
			9 = TX_RGB_12
Int	128	4	External format type
			0 = Default
			1 = TX_PACK_8
			2 = TX_PACK_16
Int	132	4	TRUE if using following 8 floats for MIPMAP kernel
Float	136	4*8	8 floats for kernel of separable symmetric filter
Int	168	4	if TRUE send:
Float	172	4	LOD0 for TX_CONTROL_POINT
Float	176	4	SCALE0 for TX_CONTROL_POINT
Float	180	4	LOD1 for TX_CONTROL_POINT
Float	184	4	SCALE1 for TX_CONTROL_POINT
Float	188	4	LOD2 for TX_CONTROL_POINT
Float	192	4	SCALE2 for TX_CONTROL_POINT
Float	196	4	LOD3 for TX_CONTROL_POINT
Float	200	4	SCALE3 for TX_CONTROL_POINT

Float	204	4	LOD4 for TX_CONTROL_POINT
Float	208	4	SCALE4 for TX_CONTROL_POINT
Float	212	4	LOD5 for TX_CONTROL_POINT
Float	216	4	SCALE5 for TX_CONTROL_POINT
Float	220	4	LOD6 for TX_CONTROL_POINT
Float	224	4	SCALE6 for TX_CONTROL_POINT
Float	228	4	LOD7 for TX_CONTROL_POINT
Float	232	4	SCALE7 for TX_CONTROL_POINT
Float	236	4	Control Clamp
Int	240	4	Magnification filter type for alpha
			0 = Point
			1 = Bilinear
			2 = None
			3 = Bicubic
			4 = Sharpen
			5 = Add Detail
			6 = Modulate Detail
			7 = Bilinear greater/equal
			8 = Bilinear less/equal
			9 = Bicubic greater/equal
			10 = Bicubic less/equal
Int	244	4	Magnification filter type for color
			0 = Point
			1 = Bilinear
			2 = None
			3 = Bicubic

			4 = Sharpen
			5 = Add Detail
			6 = Modulate Detail
			7 = Bilinear greater/equal
			8 = Bilinear less/equal
			9 = Bicubic greater/equal
			10 = Bicubic less/equal
Float	248	4	Reserved
Float	252	4*8	Reserved
Double	284	8	Lambert conic projection central meridian
Double	292	8	Lambert conic projection upper latitude
Double	300	8	Lambert conic projection lower latitude
Double	308	8	Reserved
Float	316	4*5	Reserved
Int	336	4	TRUE if using next 5 integers for TX_DETAIL
Int	340	4	J argument for TX_DETAIL
Int	344	4	K argument for TX_DETAIL
Int	348	4	M argument for TX_DETAIL
Int	352	4	N argument for TX_DETAIL
Int	356	4	Scramble argument for TX_DETAIL
Int	360	4	TRUE if using next 4 floats for TX_TILE
Float	364	4	Lower-left u value for TX_TILE

Float	368	4	Lower-left v value for TX_TILE
Float	372	4	Upper-right u value for TX_TILE
Float	376	4	Upper-right v value for TX_TILE
Int	380	4	Projection
			0 = Flat earth
			3 = Lambert conic
			4 = UTM
			7 = Undefined projection
Int	384	4	Earth model
			0 = WGS84
			1 = WGS72
			2 = Bessel
			3 = Clark 1866
			4 = NAD27
Int	388	4	Reserved
Int	392	4	UTM zone
Int	396	4	Image origin
			0 = Lower left
			1 = Upper left
Int	400	4	Geospecific points units
			0 = Degrees
			1 = Meters
			2 = Pixels
Int	404	4	Reserved
Int	408	4	Reserved
Int	412	4	Hemisphere for geospecific points units
			0 = Southern
			1 = Northern
Int	416	4	Reserved
Int	420	4	Reserved
Int	424	149*4	Reserved

Char	1020	512	Comments; 0 terminates
Int	1538	13*4	Reserved
Int	1584	4	Attribute file version number
Int	1588	4	Number of geospecific control points

If the value of the Number of geospecific control points field is greater than 0, the following fields are also contained in the attribute file:

Table 101. Geospecific Control Point subrecord

Data Type	Offset	Length
Int	4	Reserved

The following fields are repeated for each geospecific control point in the texture attribute file.

Note: In the fields below, N ranges from 0 to Number of geospecific control points – 1.

The earth coordinates depend on the projection, earth model, and geospecific points units.

Double	8	Texel uN - texel u of control point
Double	8	Texel vN - texel v of control point
Double	8	Earth coordinate xN - earth x coordinate of control point.
Double	8	Earth coordinate yN - earth y coordinate of control point.

If the value of the Number of geospecific control points field is greater than 0, the following fields are also contained in the attribute file:

Data Type	Offset	Length
Int	4	Number of subtextures

If the value of the Number of subtextures field is greater than 0, the following fields are repeated for each subtexture in the texture attribute file.

In the fields below, N ranges from 0 to Number of subtextures - 1.

The Left, Bottom, Right and Top fields are all measured in texels.

Table 102. Subtexture subrecord

Data Type	Offset	Length

Char	32	NameN - name of subtexture N; 0 terminates
Int	4	LeftN - Coordinate of left edge of subtexture N
Int	4	BottomN - Coordinate of bottom edge of subtexture N
Int	4	RightN - Coordinate of right edge of subtexture N
Int	4	TopN - Coordinate of top edge of subtexture N

B.14. Road Path Files

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider the Road Path Files.

A road path file contains the attributes of a road path node in ASCII format. The name of the file is user defined. Each attribute is denoted by a keyword, a literal colon, a space, and the value(s). Boolean values are denoted by the string literals “TRUE” and “FALSE”. For the “POINT” keyword its values consist of an XYZ coordinate and an orientation vector, separated by spaces. The orientation vector is specified as either a normal up-vector, or in degrees of heading, pitch, and roll. The “STORE_HPR” keyword specifies which method is used. For path nodes that define the road’s centerline path, construction information for the correlated road section is also stored with additional keywords. Here’s an example:

ROAD_ID: 2.0	This is the first section (a curve) in Road #2 of the database. Section numbers start at zero.
ROAD_TYPE: Curve	Road ID also appears on database node.
ARC_RADIUS: 175.000000	
SPIRAL_LEN1: 80.000000	
SPIRAL_LEN2: 80.000000	
SUPERELEVATION: 0.080000	
CONTROL_POINT: 0.000000 300.000000 0.000000	
VCURVE_LEN: 400.000000	
VCURVE_MIN: 20.000000	
SLOPE1: 0.000000	
SLOPE2: 0.000000	
WIDTH: 12.000000	Road width and centerline placement
CENTER2LEFT: 6.000000	
NUM_LANES: 2	
LANE_OFFSET: 1.825000	Lane information for the road section
LANE_OFFSET: -1.825000	
PROFILE_NAME: /usr/people/db/road/crown.flt	
PROFILE_POINT: 12.000000 0.000000	
PROFILE_POINT: 9.823453 0.300000	
PROFILE_POINT: 6.000000 0.500000	
PROFILE_POINT: 3.200000 0.300000	
PROFILE_POINT: 0.000000 0.000000	
SPEED: 70.000000	Passing lane flag (path attribute page)
NO_PASSING: TRUE	
STORE_HPR: TRUE	Heading, Pitch and Roll data will be stored and reported
NUM_POINTS: 12	
POINT: 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000	
POINT: 0.000000 83.548590 0.000000 0.000000 0.000000 0.000000	
POINT: 2.906056 145.953096 0.000000 8.000000 0.000000 3.574879	
POINT: 6.072530 163.131640 0.000000 13.096178 0.000000 4.573921	
POINT: 13.249899 186.467582 0.000000 21.096178 0.000000 4.573921	
POINT: 36.936741 229.031118 0.000000 37.096180 0.000000 4.573921	
POINT: 71.438096 263.416837 0.000000 53.096180 0.000000 4.573921	
POINT: 114.080915 286.960649 0.000000 69.096176 0.000000 4.573921	
POINT: 136.868360 293.927470 0.000000 76.903824 0.000000 4.573921	
POINT: 166.586342 298.521248 0.000000 84.903824 0.000000 2.853243	
POINT: 216.451410 300.000000 0.000000 90.000000 0.000000 0.000000	
POINT: 300.000000 300.000000 0.000000 90.000000 0.000000 0.000000	

B.15. Road Zone Files

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider the Road Zone Files.

Zone files are gridded posts files containing elevation and attribute data for a road. The zone data is followed immediately by a series of:

(Number of data points in x) * (Number of data points in y) elevation data points.

The elevation data points are followed immediately by a series of:

(Number of data points in x) * (Number of data points in y) surface types corresponding to each of the elevation data points above.

The elevation data points as well as the surface types begin at the lower-left corner. Values are ordered from bottom to top, then in columns from left to right.

Table 103. Road Zone File Format

Data Type	Offset	Length	Data Type
Int	0	4	Version - road tools format version
Int	4	4	Reserved

Double	8	8*3	Lower left corner (x, y, z)
Double	32	8*3	Upper right corner (x, y, z)
Double	56	8	Grid interval - spacing between data points
Int	64	4	Number of data points in x
Int	68	4	Number of data points in y
Float	72	4	Low z elevation data point
Float	76	4	High z elevation data point
Char	80	440	Reserved

The following field is repeated for each data point in the road zone file.

In this field, N ranges from 0 to Number of data points - 1, where

Number of data points = Number of data points in x * Number of data points in y.

Table 104. Elevation Data Point subrecord

Data Type	Offset	Length	Data Type
Float	520+(N*4)	4	ZN - elevation value for data point N

The following field is repeated for each data point in the road zone file.

In this field, N ranges from 0 to Number of data points - 1, where

Number of data points = Number of data points in x * Number of data points in y and M is equal to Number of data points.

Table 105. Surface Type subrecord

Data Type	Offset	Length	Data Type
Char	520+(M*4)+N	1	Surface typeN - user defined surface type for data point N

B.16. Linkage Editor Parameter IDs

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider Linkage Editor Parameter IDs.

Table 106. Vertex Node Parameters

ID	Description
----	-------------

258	X coordinate
259	Y coordinate
260	Z coordinate
261	Texture U coordinate
262	Texture V coordinate
265	Color
266	Hard edge flag
267	Freeze normal flag
269	Normal I component
270	Normal J component
271	Normal K component

Table 107. Face Node Parameters

ID	Description
514	Color
515	Polygon drawing
516	Lighting mode
518	Relative priority
519	Draw both sides flag
520	Texture index
521	Template
522	Infrared
523	Terrain flag
525	Material index
526	Feature ID
527	Surface material code
529	Draw textured faces white
530	IR material
534	Detail texture index
535	Transparency
536	Alternate color
537	LOD control
538	Line style index
539	Light point directional mode
540	Texture mapping

Table 108. Object Node Parameters

ID	Description
770	Relative priority
771	Inhibit during day flag
772	Inhibit during dusk flag
773	Inhibit during night flag
774	No illumination flag
775	Flat shading flag
776	Shadow flag
777	Transparency
778	Special #1
779	Special #2
782	Significance

Table 109. LOD Node Parameters

ID	Description
1026	Switch-in distance
1027	Switch-out distance
1028	Special #1
1029	Special #2
1030	Use previous range flag
1031	Center X coordinate
1032	Center Y coordinate
1033	Freeze center flag
1034	Center Z coordinate
1036	Additive LOD's below flag
1037	Transition distance

Table 110. Group Node Parameters

ID	Description
1282	Relative priority
1284	Animation type
1286	Bounding volume type
1287	Special #1
1288	Special #2
1289	Replication count

1290	Significance
1291	Layer

Table 111. DOF Node Parameters

ID	Description
1538	Current Z
1539	Minimum Z
1540	Maximum Z
1542	Current Y
1543	Minimum Y
1544	Maximum Y
1546	Current X
1547	Minimum X
1548	Maximum X
1550	Current pitch
1551	Minimum pitch
1552	Maximum pitch
1554	Current roll
1555	Minimum roll
1556	Maximum roll
1558	Current yaw
1559	Minimum yaw
1560	Maximum yaw
1562	Current Z scale
1563	Minimum Z scale
1564	Maximum Z scale
1566	Current Y scale
1567	Minimum Y scale
1568	Maximum Y scale
1570	Current X scale
1571	Minimum X scale
1572	Maximum X scale
1574	X constrained motion flag
1575	Y constrained motion flag
1576	Z constrained motion flag

1577	Pitch constrained motion flag
1578	Roll constrained motion flag
1579	Yaw constrained motion flag
1580	X scale constrained motion flag
1581	Y scale constrained motion flag
1582	Z scale constrained motion flag
1583	Repeating texture flag
1584	Membrane mode flag

Table 112. Sound Node Parameters

ID	Description
1796	Amplitude
1797	Pitch bend
1798	Priority
1799	Falloff
1800	Width
1801	Doppler
1802	Absorption
1803	Delay
1804	Directivity
1805	X coordinate
1806	Y coordinate
1807	Z coordinate
1808	Direction vector I component
1809	Direction vector J component
1810	Direction vector K component
1812	Active flag

Table 113. Switch Node Parameters

ID	Description
2050	Current mask index

Table 114. Text Node Parameters

ID	Description
2307	Text type
2308	Draw type

2310	Color
2311	Alternate color
2312	Material index
2315	Integer value minimum
2316	Integer value maximum
2317	Float value minimum
2318	Float value maximum
2325	Current integer value
2326	Current float value
2327	Decimal places for float value
2329	Line style index
2330	Justification type
2331	Vertical flag
2332	Bold flag
2333	Italic flag
2334	Underline flag

Table 115. Light Source Node Parameters

ID	Description
2819	Enabled flag
2820	Global flag
2821	X coordinate
2822	Y coordinate
2823	Z coordinate
2824	Yaw
2825	Pitch

Table 116. Clip Node Parameters

ID	Description
3074	Plane 0 enable
3075	Plane 1 enable
3076	Plane 2 enable
3077	Plane 3 enable
3078	Plane 4 enable

B.17. OpenFlight Opcodes

Table 117. Valid Opcodes

Opcode	Record Type	For more information, see...	CDB OpenFlight Reader
1	Header	Header Record	□
2	Group	Group Record	□
4	Object	Object Record	□
5	Face	Face Record	□
10	Push Level	Push Level Record	□
11	Pop Level	Pop Level Record	□
14	Degree of Freedom	Degree of Freedom Record	□
19	Push Subface	Push Subface Record	□
20	Pop Subface	Pop Subface Record	□
21	Push Extension	Push Extension Record	□
22	Pop Extension	Pop Extension Record	□
23	Continuation	Continuation Record	□
31	Comment	Comment Record	□
32	Color Palette	Color Palette Record	□
33	Long ID	Long ID Record	□
50	Vector	Vector Record	□
52	Multitexture	Multitexture Record	□
53	UV List	UV List Record	□
55	Binary Separating Plane	Binary Separating Plane Record	□
60	Replicate	Replicate Record	□
61	Instance Reference	Instance Reference Record	□
62	Instance Definition	Instance Definition Record	□
63	External Reference	External Reference Record	□
64	Texture Palette	Texture Palette Record	□
72	Vertex List	Vertex List Record	□
73	Level of Detail	Level of Detail Record	□

83	Eyepoint and Trackplane Palette	Eyepoint and Trackplane Palette Record	🔗
84	Mesh	Mesh Record	🔗
85	Local Vertex Pool	Local Vertex Pool Record	🔗
86	Mesh Primitive	Mesh Primitive Record	🔗
87	Road Segment	Road Segment Record	🔗
88	Road Zone	Road Zone Record	🔗
89	Morph Vertex List	Morph Vertex List Record	🔗
90	Linkage Palette	Linkage Palette Record	🔗
91	Sound	Sound Record	🔗
92	Road Path	Road Path Record	🔗
93	Sound Palette	Sound Palette Record	🔗
95	Text	Text Record	🔗
96	Switch	Switch Record	🔗
98	Clip Region	Clip Region Record	🔗
100	Extension	Extension Record	🔗
101	Light Source	Light Source Record	🔗
102	Light Source Palette	Light Source Palette Record	🔗
103	Reserved		🔗
104	Reserved		🔗
110	Reserved		🔗
111	Light Point	Light Point Record	🔗
112	Texture Mapping Palette	Texture Mapping Palette Record	🔗
113	Material Palette	Material Palette Record	🔗
114	Name Table	Name Table Record	🔗
115	Continuously Adaptive Terrain (CAT)	CAT Record	🔗
116	CAT Data	CAT Data Record	🔗
117	Reserved		🔗
118	Reserved		🔗
120	Reserved		🔗

121	Reserved		□
122	Push Attribute	Push Attribute Record	□
123	Pop Attribute	Pop Attribute Record	□
124	Reserved		□
125	Reserved		□
126	Curve	Curve Record	□
127	Road Construction	Road Construction Record	□
128	Light Point Appearance Palette	Light Point Appearance Palette Record	□
129	Light Point Animation Palette	Light Point Animation Palette Record	□
130	Indexed Light Point	Indexed Light Point Record	□
131	Light Point System	Light Point System Record	□
132	Indexed String	Indexed String Record	□
133	Shader Palette	Shader Palette Record	□

B.17.1. Obsolete Opcodes

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider the following obsolete OpenFlight opcodes.

Opcode	Record Type
3	Level of Detail (single precision floating point, replaced by Opcode 73)
6	Vertex with ID (scaled integer coordinates, replaced by Opcodes 68-71)
7	Short Vertex w/o ID (scaled integer coordinates, replaced by Opcodes 68-71)
8	Vertex with Color (scaled integer coordinates, replaced by Opcodes 68-71)
9	Vertex with Color and Normal (scaled integer coordinates, replaced by Opcodes 68-71)
12	Translate (replaced by Opcode 78)
13	Degree of Freedom (scaled integer coordinates, replaced by Opcode 14)
16	Instance Reference (replaced by Opcode 61)

17	Instance Definition (replaced by Opcode 62)
40	Translate (replaced by Opcode 78)
41	Rotate about Point (replaced by Opcode 80)
42	Rotate about Edge (replaced by Opcode 76)
43	Scale (replaced by Opcode 79)
44	Translate (replaced by Opcode 78)
45	Scale nonuniform (replaced by Opcode 79)
46	Rotate about Point (replaced by Opcode 80)
47	Rotate and/or Scale to Point (replaced by Opcode 81)
48	Put (replaced by Opcode 82)
51	Bounding Box (replaced by Opcode 74)
65	Eyepoint Palette (only eyepoints, replaced by Opcode 83)
66	Material Palette (fixed size 64 entries, replaced by Opcode 80)
77	Scale (replaced by Opcode 79)

[32] The SGI format is fully supported by the CDB standard but a single file extension used, *.rgb. Consequently, all image formats (int, inta, rgb, and rgba) are stored in .rgb files regardless of the number of channels in the image.

Annex C: Summary of Changes Version 15.7

CDB OpenFlight Readers: This section is not applicable to CDB-compliant OpenFlight readers. The first version of the CDB standard is based on OpenFlight v16.0.

C.1. C - Overview

This section describes the changes in the OpenFlight Scene Description between versions 15.6 and 15.7. OpenFlight version 15.7 coincides with MultiGen Creator versions 2.4 through 2.5.1 and the OpenFlight API versions 2.4 through 2.5.1. The changes made for this version are:

[Continuation Record](#)

[Header Record](#)

[Mesh Record](#)

[Local Vertex Pool Record](#)

[Mesh Primitive Record](#)

[Multitexture Record](#)

[UV List Record](#)

[Texture Attribute File](#)

C.2. Format Changes

C.2.1. Continuation Record

All OpenFlight records begin with a 4 byte sequence. The first two bytes identify the record (opcode) and the second two bytes specify the length of the record. Given this regular record structure, the length of all OpenFlight records is limited to the largest value that can be encoded with 2 bytes or 16 bits (65535). In most cases, this maximum size is sufficient but there are cases where it is not. For fixed size records, this is not a problem. For variable size records, this limitation is being addressed with this version.

A new record, called the continuation record is introduced in this version to accommodate variable size records in the OpenFlight Scene Description. The continuation record is used to “continue” a record in the OpenFlight Scene Description file stream. It would appear in the stream immediately following the record that it “continues” (the record that is being continued will be referred to as the “original” record). The data contained in the continuation record is defined by the original record and is assumed to be directly appended onto the content of the original record.

NOTE

Multiple continuation records may follow a record, in which case all continuation records would be appended (in sequence) to the original record.

Table 118. Continuation Record - New record for OpenFlight 15.7

Data Type	Offset	Length	Description
Unsigned Int	0	2	Continuation Record Opcode 23
Unsigned Int	2	2	Length - length of the record
Varies	4	Length-4	Depends on the original record. The contents of this field are to be appended directly to the end of the original record contents (before the original record contents are parsed)

C.2.2. Header Record

New attributes have been appended to the end of the existing header record (see [Header Record](#)). The following fields were added at the specified offsets in the header record.

Table 119. Header Record changes for OpenFlight 15.7 - New Fields

Data Type	Offset	Length	Description
Double	284	8	Delta z to place database (used in conjunction with existing Delta x and Delta y values)
Double	292	8	Radius (distance from database origin to farthest corner)
Unsigned Int	300	2	Next Mesh node ID number
Unsigned Int	302	2	Reserved

C.3. Mesh Nodes

A mesh node defines a set of geometric primitives that share attributes and vertices. In previous versions of OpenFlight, the fundamental geometric construct was the polygon. Each polygon has a unique set of attributes and vertices. Meshes are used to represent “sets” of related polygons, each sharing common attributes and vertices. Using a mesh, related polygons can be represented in a much more compact format. Each mesh will share one set of “polygon” attributes (color, material, texture, etc.), a common “vertex pool” and one or more geometric primitives that use the shared attributes and vertices. Using a mesh you can represent triangle strips, triangle fans, quadrilateral strips and indexed face sets.

A mesh node is defined by three distinct record types:

- *Mesh Record* - defines the “polygon” attributes associated to all geometric primitives of the mesh.
- *Local Vertex Pool Record* - defines the set of vertices that are referenced by the geometric primitives of the mesh.
- *Mesh Primitive Record* - defines a geometric primitive (triangle-strip, triangle-fan, quadrilateral-strip or indexed face set) for the mesh.

A mesh node consists of one mesh record, one local vertex pool record, and one or more mesh primitive records. The mesh primitive records are delimited by push and pop control records as shown in the following example:

```
MESH
LOCAL VERTEX POOL
PUSH MESH
PRIMITIVE
MESH PRIMITIVE
...
MESH PRIMITIVE
POP
```

C.4. Mesh Record

The mesh record is the primary record of a mesh node and defines the common “face-like” attributes associated to all geometric primitives of the mesh. These attributes are identical to those of the face record. See [Face Record](#).

Table 120. Mesh Record- New record for OpenFlight 15.7

Data Type	Offset	Length	Description
Int	0	2	Mesh Opcode 84
Unsigned Int	2	2	Length - length of the record
Char	4	8	7 char ASCII ID; 0 terminates
Int	12	4	IR color code
Int	16	2	Relative priority
Int	18	1	Draw type
			0 = Draw solid with backface culling
			1 = Draw solid, no backface culling
			2 = Draw wireframe

			3 = Draw wireframe and close
			4 = Surround with wireframe in alternate color
			8 = Omnidirectional light
			9 = Unidirectional light
			10 = Bidirectional light
Int	19	1	Texture white = if TRUE, draw textured face white
Unsigned Int	20	2	Color name index
Unsigned Int	22	2	Alternate color name index
Int	24	1	Reserved

Table 121. Mesh Record - New record for OpenFlight 15.7 (Continued)

Data Type	Offset	Length	Description
Int	25	1	Template (billboard)
			0 = Fixed, no alpha blending
			1 = Fixed, alpha blending
			2 = Axial rotate with alpha blending
			4 = Point rotate with alpha blending
Int	26	2	Detail texture pattern index, -1 if none
Int	28	2	Texture pattern index, -1 if none
Int	30	2	Material index, -1 if none
Int	32	2	Surface material code (for DFAD)
Int	34	2	Feature ID (for DFAD)
Int	36	4	IR material code
Unsigned Int	40	2	Transparency

			0 = Opaque
			65535 = Totally clear
Unsigned Int	42	1	LOD generation control
Unsigned Int	43	1	Line style index
Int	44	4	Flags (bits from left to right)
			0 = Terrain
			1 = No color
			2 = No alternate color
			3 = Packed color
			4 = Terrain culture cutout (footprint)
			5 = Hidden, not drawn
			6-31 = Spare
Unsigned Int	48	1	Light mode
			0 = Use mesh color, not illuminated
			1 = Use vertex colors, not illuminated
			2 = Use mesh color and vertex normals
			3 = Use vertex colors and vertex normals
Char	49	7	Reserved
Unsigned Int	56	4	Packed color, primary (a, b, g, r)
Unsigned Int	60	4	Packed color, alternate (a, b, g, r)
Int	64	2	Texture mapping index
Int	66	2	Reserved
Unsigned Int	68	4	Primary color index
Unsigned Int	72	4	Alternate color index
Int	76	4	Reserved

C.5. Local Vertex Pool Record

This record defines a set of vertices that is referenced by the geometry (primitives) of the mesh.

NOTE

Currently the Local Vertex Pool is used exclusively in the context of mesh nodes, but it is designed in a general way so that it may appear in other contexts in future versions of the OpenFlight Scene Description.

Table 122. Local Vertex Pool Record- New record for OpenFlight 15.7

Data Type	Offset	Length	Description
Int	0	2	Local Vertex Pool Opcode 85
Unsigned Int	2	2	Length - length of the record Note: Since the length of this record is represented by an unsigned short, the maximum length of the vertex pool is 2 ¹⁶ - 1 (or 65535 bytes). If the entire vertex pool cannot fit into this size, one or more continuation records will follow. (See Continuation Record .)
Unsigned Int	4	4	Number of vertices - number of vertices in the local vertex pool
Unsigned Int	8	4	Attribute mask - Bit mask indicating what kind of vertex information is specified for each vertex in the local vertex pool. Bits are ordered from left to right as follows:
			Bit # Description
			0 Has Position - if set, data for each vertex in will include x, y, and z coordinates (3 doubles)
			1 Has Color Index - if set, data for each vertex will include a color value that is a color table index (1 int)
			2 Has RGB Color - if set, data for each vertex will include a color value that is a packed RGB color (1 int)

			Note: Bits 1and 2 are mutually exclusive - a vertex can have either color index or RGB color value or neither, but not both.
		3	Has Normal - if set, data for each vertex will include a normal (3 floats)
		4	Has Base UV - if set, data for each vertex will include uv texture coordinates for the base texture (2 floats)
		5	Has UV Layer 1 - if set, data for each vertex will include uv texture coordinates for layer 1 (2 floats)
		6	Has UV Layer 2 - if set, data for each vertex will include uv texture coordinates for layer 2 (2 floats)
		7	Has UV Layer 3 - if set, data for each vertex will include uv texture coordinates for layer 3 (2 floats)
		8	Has UV Layer 4 - if set, data for each vertex will include uv texture coordinates for layer 4 (2 floats)

			9	Has UV Layer 5 - if set, data for each vertex will include uv texture coordinates for layer 5 (2 floats)
			10	Has UV Layer 6 - if set, data for each vertex will include uv texture coordinates for layer 6 (2 floats)
			11	Has UV Layer 7 - if set, data for each vertex will include uv texture coordinates for layer 7 (2 floats)
		12-31		Spare

Table 123. Local Vertex Pool Record - New record for OpenFlight 15.7 (Continued)

Then beginning at offset 12, the following fields are repeated for each vertex in the local vertex pool, depending on the bits set in the Attribute mask field above:

In the fields listed below, N ranges from 0 to Number of vertices - 1.

Double	Varies	8*3	CoordinateN - Coordinate of vertex N (x, y, z) - present if Attribute mask includes Has Position.
Unsigned Int	Varies	4	colorN - Color for vertex N - present if Attribute mask includes Has Color Index or Has RGB Color. If Has Color Index, specifies color table index. If Has RGB Color, 4 bytes specify (a, b, g, r) values (alpha ignored).
Float	Varies	4*3	normalN - Normal for vertex N (i, j, k) - present if Attribute mask includes Has Normal.

Float	Varies	4*2	uvBaseN - Texture coordinates (u, v) for base texture layer of vertex N - present if Attribute mask includes Has Base UV.
Float	Varies	4*2	uv1N - Texture coordinates (u, v) for layer 1 of vertex N - present if Attribute mask includes Has UV Layer 1.
Float	Varies	4*2	uv2N - Texture coordinates (u, v) for layer 2 of vertex N - present if Attribute mask includes Has UV Layer 2.
Float	Varies	4*2	uv3N - Texture coordinates (u, v) for layer 3 of vertex N - present if Attribute mask includes Has UV Layer 3.
Float	Varies	4*2	uv4N - Texture coordinates (u, v) for layer 4 of vertex N - present if Attribute mask includes Has UV Layer 4.
Float	Varies	4*2	uv5N - Texture coordinates (u, v) for layer 5 of vertex N - present if Attribute mask includes Has UV Layer 5.
Float	Varies	4*2	uv6N - Texture coordinates (u, v) for layer 6 of vertex N - present if Attribute mask includes Has UV Layer 6.

Float	Varies	4*2	uv7N - Texture coordinates (u, v) for layer 7 of vertex N - present if Attribute mask includes Has UV Layer 7.
-------	--------	-----	--

C.6. Mesh Primitive Record

This record defines a geometric primitive (triangle strip, triangle fan, quadrilateral strip, or indexed polygon) for a mesh.

Table 124. Mesh Primitive Record - New record for OpenFlight 15.7

Data Type	Offset	Length	Description
Int	0	2	Mesh Primitive Opcode 86
Unsigned Int	2	2	Length - length of the record
Int	4	2	Primitive Type - specifies how the vertices of the primitive are interpreted
			1 = Triangle Strip
			2 = Triangle Fan
			3 = Quadrilateral Strip
			4 = Indexed Polygon
Unsigned Int	6	2	Index Size - specifies the length (in bytes) of each of the vertex indices that follow - will be either 1, 2, or 4
Unsigned Int	8	4	Vertex Count - number of vertices in this primitive.

The following field is repeated for each vertex referenced by the mesh primitive. These vertices are interpreted according to Primitive Type. In the field below, N ranges from 0 to Vertex Count - 1.

Int	12+(N*Index Size)	Index Size	IndexN - Index of vertex N of the mesh primitive.
-----	-------------------	------------	---

Each mesh primitive is represented using the Mesh Primitive record above. The following descriptions explain how the vertices of each primitive type are interpreted as geometry:

•**Triangle Strip** - This mesh primitive defines a connected group of triangles in the context of the enclosing mesh. Each triangle shares the “polygon” attributes defined by the enclosing mesh. This primitive contains a sequence of indices that reference vertices from the local vertex pool. One triangle is defined for each vertex presented after the first two vertices. For odd n, vertices n, n+1, and n+2 define triangle n. For even n, vertices n+1, n, and n+2 define triangle n. The first triangle is n=1. The first vertex in the vertex pool is n=1. N vertices represent N-2 triangles.

•**Triangle Fan** - Like the Triangle Strip, this mesh primitive also defines a connected group of triangles in the context of the enclosing mesh. Each triangle shares the “polygon” attributes defined by the enclosing mesh. This primitive contains a sequence of indices that reference vertices from the local vertex pool. One triangle is defined for each vertex presented after the first two vertices. Vertices 1, n+1, and n+2 define triangle n. The first triangle is n=1. The first vertex in the vertex pool is n=1. N vertices represent N-2 triangles.

•**Quadrilateral Strip** - This mesh primitive defines a connected group of quadrilaterals in the context of the enclosing mesh. Each quadrilateral shares the “polygon” attributes defined by the enclosing mesh. This primitive contains a sequence of indices that reference vertices from the local vertex pool. One quadrilateral is defined for each pair of vertices presented after the first pair. Vertices 2n-1, 2n, 2n+2, and 2n+1 define quadrilateral n. The first quadrilateral is n=1. The first vertex in the vertex pool is n=1. N vertices represent (N/2)-1 quadrilaterals.

•**Indexed Polygon** -This mesh primitive defines a single polygon in the context of the enclosing mesh. This primitive is similar to the other mesh primitives in that it also shares the polygon attributes of the enclosing mesh. It is different from the other mesh primitive types in that while triangle strips/fans and quadrilateral strips describe a set of connected triangles/quadrilaterals, the indexed polygon defines a single polygon. This primitive contains a sequence of indices that reference vertices from the local vertex pool. One polygon is defined by the sequence of vertices in this record. N vertices represent 1 N-sided closed polygon or 1 (N-1)-sided unclosed polygon.

C.7. Multitexture

OpenFlight supports 8 textures per polygon or mesh as well as 8 uv's per vertex. The current texture information stored on the polygon is referred to as “the base texture” or “texture layer 0”. Each additional texture is referred to as “texture layer N”. Therefore, to support 8 textures per polygon, a base texture is required as well as 7 additional texture layers. The additional texture layers for each polygon, mesh, and vertex will be represented in ancillary records at the Face, Mesh, and Vertex primary node levels as shown in the following example:

```
FACE
MULTITEXTURE
PUSH
VERTEX LIST
UV LIST
POP
```

The records that are used to represent multitexture in the OpenFlight file are described in the following sections.

C.7.1. Multitexture Record

The multitexture record is an ancillary record of face and mesh nodes. It specifies the texture layer information for the face or mesh.

Table 125. Multitexture Record - New record for OpenFlight 15.7

Data Type	Offset	Length	Description
Unsigned Int	0	2	Multitexture Opcode 52
Unsigned Int	2	2	Length - length of the record
Int	4	4	Attribute mask - Bit mask indicating what kind of multitexture information is present in this record. Bits are ordered from left to right as follows:
			Bit # Description
			0 Has Layer 1 - if this bit is set, multitexture information for texture layer 1 is present.
			1 Has Layer 2 - if this bit is set, multitexture information for texture layer 2 is present.
			2 Has Layer 3 - if this bit is set, multitexture information for texture layer 3 is present.
			3 Has Layer 4 - if this bit is set, multitexture information for texture layer 4 is present.
			4 Has Layer 5 - if this bit is set, multitexture information for texture layer 5 is present.
			5 Has Layer 6 - if this bit is set, multitexture information for texture layer 6 is present.

			6	Has Layer 7 - if this bit is set, multitexture information for texture layer 7 is present.
			7-31	Spare

The following fields are repeated for each multitexture layer that is specified as present by the bits set in the Attribute mask field above. This mechanism allows for “sparse” multitexture layer information to be present and does not require that the information present be contiguous.

Unsigned Int	Varies	2	textureN - Texture index for texture layer N
Unsigned Int	Varies	2	effectN - Multitexture effect for texture layer N
			0 = Texture environment
			1 = Bump map
			2-100 = Reserved by MultiGen-Paradigm
			>100 = user (runtime) defined
Unsigned Int	Varies	2	mappingN - Texture mapping index for texture layer N
Unsigned Int	Varies	2	dataN - Texture data for layer N. This is user defined. For example, it may be used as a blend percentage or color or any other data needed by the runtime to describe texture layer N

C.7.2. UV List Record

The uv list record is an ancillary record of vertex nodes. This record (if present) always follows the vertex list or morph vertex list record and contains texture layer information for the vertices represented in the vertex list record it follows.

Table 126. UV List Record - New record for OpenFlight 15.7

Data Type	Offset	Length	Description
Unsigned Int	0	2	UV List Opcode 53
Unsigned Int	2	2	Length - length of the record
Int	4	4	Attribute mask - Bit mask indicating what kind of multitexture information is present in this record. Bits are ordered from left to right as follows:

			Bit #	Description
			0	Has Layer 1 - if set, uvs for layer 1 are present
			1	Has Layer 2 - if set, uvs for layer 2 are present
			2	Has Layer 3 - if set, uvs for layer 3 are present
			3	Has Layer 4 - if set, uvs for layer 4 are present
			4	Has Layer 5 - if set, uvs for layer 5 are present
			5	Has Layer 6 - if set, uvs for layer 6 are present
			6	Has Layer 7 - if set, uvs for layer 7 are present
			7-31	Spare

The following fields are repeated for each vertex contained in the corresponding vertex list or morph vertex list record.

If this uv list record follows a vertex list record, the following fields are repeated for each layer present (as specified by the bits set in the Attribute mask field).

Data Type	Offset	Description
Float	4	ui, N - Texture U for vertex i, layer N
Float	4	vi, N - Texture V for vertex i, layer N

If this uv list record follows a morph vertex list record, the following fields are repeated for each layer present (as specified by the bits set in the Attribute mask field).

Data Type	Offset	Description
Float	4	u0i, N - Texture U for the 0% vertex i, layer N

Float	4	v0i, N - Texture V for the 0% vertex i, layer N
Float	4	u100i, N - Texture U for the 100% vertex i, layer N
Float	4	v100i, N - Texture V for the 100% vertex i, layer N

C.8. Texture Attribute File

C.8.1. Subtexture

Subtexture definitions have been added to the end of the Texture Attribute File (see [Texture Attribute Files](#)). After all the geospecific control points are listed, the following subtexture information now appears:

Table 127. Texture Attribute File Format changes for OpenFlight 15.7 - New Fields

Data Type	Length	Description
Int	4	Number of subtextures

If the value of the Number of subtextures field is greater than 0-, the following fields are repeated for each subtexture in the texture attribute file.

The Left, Bottom, Right and Top fields are all measured in texels.

Data Type	Length	Description
Char	32	NameN - name of subtexture N; 0 terminates
Int	4	LeftN - Coordinate of left edge of subtexture N
Int	4	BottomN - Coordinate of bottom edge of subtexture N
Int	4	RightN - Coordinate of right edge of subtexture N
Int	4	TopN - Coordinate of top edge of subtexture N

Annex D: Summary of Changes Version 15.8

D.1. D - Overview

This section describes the changes in the OpenFlight Scene Description between versions 15.7 and 15.8 as well as the errors contained in previous versions of this document that have been corrected in this version.

OpenFlight version 15.8 coincides with MultiGen Creator version 2.6 and the OpenFlight API version 2.6. The changes made for this version are:

[Header Record](#)

[Group Record](#)

[Level of Detail Record](#)

[External Reference Record](#)

[Indexed String Record](#) (for Switch nodes)

[Face Record](#)

[Mesh Record](#)

[Local Vertex Pool Record](#)

[Vertex Palette Records](#)

[Light Point Appearance Palette Record](#)

[Light Point Animation Record](#)

[Indexed Light Point Record](#)

[Light Point System Record](#)

[Texture Mapping Palette Record](#)

Also new in this version of the document is the addition of the “offset” column in the record format tables.

D.2. Document Corrections

The errors corrected in this version of the document are described in the sections that follow.

D.2.1. Text Record

The Reserved field, previously omitted in prior versions of this document, has been documented in the specification for OpenFlight 15.8. The offsets of fields following this field have been adjusted

accordingly.

Table 128. Text Record error corrected in OpenFlight 15.8 specification - Reserved field (documented)

Data Type	Offset	Length	Description
Int	16	4	Reserved

The Draw bold field, previously omitted in prior versions of this document, has been documented in the specification for OpenFlight 15.8. The offsets of fields following this field have been adjusted accordingly.

Table 129. Text Record error corrected in OpenFlight 15.8 specification - Draw bold field (documented)

Data Type	Offset	Length	Description
Int	304	4	Draw bold

For a complete description of the text record, see [Text Record](#).

D.2.2. CAT Record

The Relative priority field, included erroneously in the previous version of this document, has been removed from the specification for OpenFlight 15.8. The offsets of fields following this field have been adjusted accordingly.

Table 130. CAT Record error corrected in OpenFlight 15.8 specification - Relative priority field (removed)

Data Type	Offset	Length	Description
Int	20	2	Relative priority

The Feature ID field, included erroneously in the previous version of this document, has been removed from the specification for OpenFlight 15.8. The offsets of fields following this field have been adjusted accordingly.

Table 131. CAT Record error corrected in OpenFlight 15.8 specification - Feature ID field (removed)

Data Type	Offset	Length	Description
Int	38	2	Relative priority

The Reserved field, previously omitted in prior version of this document, has been documented in the specification for OpenFlight 15.8. The offsets of fields following this field have been adjusted accordingly.

Table 132. CAT Record error corrected in OpenFlight 15.8 specification - Reserved field (documented)

Data Type	Offset	Length	Description
Int	60	4	Reserved

D.3. Format Changes

D.3.1. Header Record

The header record has been modified to include additional projection attributes. New attributes have been appended to the end of the existing header record and some of the existing fields have new values possible.

The following fields were added to the end (at the specified offsets) of the header record.

Table 133. Header Record changes for OpenFlight15.8 - New Fields

Data Type	Offset	Length	Description
Unsigned Int	302	2	Next Light Point System ID number
Int	304	4	Reserved
Double	308	8	Earth major axis (for user defined ellipsoid) in meters
Double	316	8	Earth minor axis (for user defined ellipsoid) in meters

The Projection type field has been changed to include two new possible values, Geocentric and Geodetic as shown here. New values are shown in **bold font**:

Table 134. Header Record changes for OpenFlight15.8 - Projection type field

Data Type	Offset	Length 2	Description
Int	92	4	Projection type
			0 = Flat earth
			1 = Trapezoidal
			2 = Round earth
			3 = Lambert
			4 = UTM
			5 = Geocentric
			6 = Geodetic

The Earth ellipsoid model field has been changed to include one new possible value, User defined ellipsoid as shown here. This new value is shown in **bold font**.

Table 135. Header Record changes for OpenFlight15.8 - Earth ellipsoid field

Data Type	Offset	Length	Description
Int	268	4	Earth ellipsoid model
			0 = WGS 1984
			1 = WGS 1972

		2 = Bessel
		3 = Clarke 1866
		4 = NAD 1927
		5 = User defined ellipsoid

A field, previously labeled “Reserved” in prior versions of this document, has been described. It is the UTM zone for UTM projections and is shown here.

Table 136. Header Record changes for OpenFlight15.8 - UTM zone field

Data Type	Offset	Length	Description
Int	276	2	UTM zone (for UTM projections - negative value means Southern hemisphere)

For a complete description of the header record, see [Header Record](#).

D.3.2. Group Record

The group record has been modified to include additional animation attributes. New attributes have been appended to the end of the existing group record and some of the existing fields have new values possible.

The following fields were added to the end (at the specified offsets) of the group record.

Table 137. Group Record changes for OpenFlight15.8 - New Fields

Data Type	Offset	Length	Description
Int	32	4	Loop count
Float	36	4	Loop duration in seconds
Float	40	4	Last frame duration in seconds

The Flags field has been changed to include a new bit which can be used to specify backwards animations as shown here. The new bit is shown in **bold font**.

Table 138. Group Record changes for OpenFlight15.8 - Flags field

Data Type	Offset	Length	Description
Int	16	4	Flags (bits, from left to right)
			0 = Reserved
			1 = Forward animation

		2 = Swing animation
		3 = Bounding box follows
		4 = Freeze bounding box
		5 = Default parent
		6 = Backward animation
		7-31 = Spare

For a complete description of the group record, see [Group Record](#).

D.3.3. Level of Detail Record

The level of detail record has been modified to include an additional attribute, Significant size. This new value helps an application to calculate switch ranges for the geometry more effectively for different display settings (field of view, screen size and resolution).

The following field was added to the end (at the specified offset) of the level of detail record.

Table 139. LOD Record changes for OpenFlight15.8 - New Field

Data Type	Offset	Length	Description
Double	72	8	Significant size

For a complete description of the level of detail record, see [Level of Detail Record](#).

D.3.4. External Reference Record

The Flags field of the external reference record has been modified to include a new bit, Light point palette override, which is used to specify that the light point appearance and animation palettes override those contained in the master file. The new bit is shown in **bold font**.

Table 140. External Reference Record changes for OpenFlight15.8 - Flags field

Data Type	Offset	Length	Description
Int	208	4	Flags (bits, from left to right)
			0 = Color palette override
			1 = Material palette override
			2 = Texture and texture mapping palette override

		3 = Line style palette override
		4 = Sound palette override
		5 = Light source palette override
		6 = Light point palette override
		7-31 = Spare

For a complete description of the external reference record, see [External Reference Record](#).

D.3.5. Indexed String Record

Switch nodes now allow individual masks to be named. These names are stored in a new ancillary record called the Indexed String record. While these new ancillary records are currently only applicable to Switch records, they are not limited to Switch records and may be useful in future versions of OpenFlight in other contexts.

The new Indexed String Record is an ancillary record that contains an integer index followed by a variable length character string. In this way, arbitrary strings can be associated to indices in a general way.

With respect to Switch mask names, the index specifies the mask number for which the string specifies a name. Mask numbers start at 0. Not all masks are required to have names.

Table 141. Indexed String Record - New record for OpenFlight 15.8

Data Type	Offset	Length	Description
Int	0	2	Indexed string Opcode 132
Unsigned Int	2	2	Length - length of the record
Unsigned Int	4	2	Index
Char	8	Length - 8	ASCII string; 0 terminates

For a completed description of the indexed string record, see “Indexed String Record” on page 53.

D.3.6. Face Record

The Flags field of the face record has been modified to include a new bit, Roofline, which is used to specify that a face is part of a building’s roof as viewed from above. The new specification of the Flags field is shown here. The new bit is shown in **bold font**.

Table 142. Face Record changes for OpenFlight15.8 - Flags field

Data Type	Offset	Length	Description
Int	44	4	Flags (bits from left to right)
			0 = Terrain
			1 = No color
			2 = No alternate color
			3 = Packed color
			4 = Terrain culture cutout (footprint)
			5 = Hidden, not drawn
			6 = Roofline
			7-31 = Spare

D.3.7. Mesh Record

Similar to the Face record described above, the Flags field of the mesh record has been modified to include a new bit, Roofline, which is used to specify that a mesh is part of a building's roof as viewed from above. The new specification of the Flags field is shown here. The new bit is shown in **bold font**.

Table 143. Mesh Record changes for OpenFlight15.8 - Flags field

Data Type	Offset	Length	Description
Int	44	4	Flags (bits from left to right)
			0 = Terrain
			1 = No color
			2 = No alternate color
			3 = Packed color
			4 = Terrain culture cutout (footprint)
			5 = Hidden, not drawn
			6 = Roofline
			7-31 = Spare

D.3.8. Local Vertex Pool Record

The Local Vertex Pool record has been modified to include an alpha color component for those vertices in the pool that have color. The alpha color component is represented as a 1 byte integer value whose range is 0 (fully transparent) to 255 (fully opaque).

The definition of the Attribute mask field has been changed as shown here.

NOTE

The physical layout of this field has not changed, only its definition. The bits for which new definitions apply are shown in **bold font**:

Table 144. Local Vertex Pool Record changes for OpenFlight15.8 - Attribute mask field

Data Type	Offset	Length	Description
Unsigned Int	8	4	Attribute mask - Bit mask indicating what kind of vertex information is specified for each vertex in the local vertex pool. Bits are ordered from left to right as follows:
			Bit # Description
			0 Has Position - if set, data for each vertex will include x, y, and z coordinates (3 doubles)
			1 *Has Color Index - if set, data for each vertex will include a color value that specifies a color table index as well as an alpha value *
			2 Has RGBA Color - if set, data for each vertex will include a color value that is a packed RGBA color value
			Note: Bits 1and 2 are mutually exclusive - a vertex can have either color index or RGB color value or neither, but not both.
			3 Has Normal - if set, data for each vertex will include a normal (3 floats)

			4	Has Base UV - if set, data for each vertex will include uv texture coordinates for the base texture (2 floats)
			5	Has UV Layer 1 - if set, data for each vertex will include uv texture coordinates for layer 1 (2 floats)
			6	Has UV Layer 2 - if set, data for each vertex will include uv texture coordinates for layer 2 (2 floats)
			7	Has UV Layer 3 - if set, data for each vertex will include uv texture coordinates for layer 3 (2 floats)
			8	Has UV Layer 4 - if set, data for each vertex will include uv texture coordinates for layer 4 (2 floats)
			9	Has UV Layer 5 - if set, data for each vertex will include uv texture coordinates for layer 5 (2 floats)
			10	Has UV Layer 6 - if set, data for each vertex will include uv texture coordinates for layer 6 (2 floats)

			11	Has UV Layer 7 - if set, data for each vertex will include uv texture coordinates for layer 7 (2 floats)
			12-31	Spare

The color field of the vertex pool data (data for each vertex) has been modified to include an alpha color component as shown here. The affected field is shown in **bold font**.

Table 145. Local Vertex Pool Record changes for OpenFlight15.8 - Color field

Data Type	Offset	Description
Unsigned Int	4	<p>colorN - Color for vertex N - present if Attribute mask includes Has Color Index or Has RGBA Color.</p> <p>If Has Color Index, lower 3 bytes specify color table index, upper 1 byte is Alpha.</p> <p>If Has RGBA Color, 4 bytes specify (a, b, g, r) values.</p>

For a complete description of the local vertex pool record, see [Local Vertex Pool Record](#).

D.3.9. Vertex Palette Records

Vertex Palette Records have been modified to include an alpha color component. The alpha color component is represented as a 1 byte integer value whose range is 0 (fully transparent) to 255 (opaque).

Prior to OpenFlight version 15.8, vertex colors were represented in vertex palette records in one of two ways: Packed Color or Color Index. Depending on the value of the Packed color flag, either the Packed color (a, b, g, r) attribute or the Vertex color index attribute was valid, but not both. For example, if the Packed color flag was TRUE, then the Packed color attribute contained the RGB components of the vertex color and the Vertex color index attribute was not specified. Conversely, if the Packed color flag was FALSE, then the Vertex color index attribute contained the index (in the Color Palette) of the vertex color and the Packed color attribute was not specified. Furthermore, the A (alpha) component of the Packed color attribute was not valid and was ignored.

In OpenFlight version 15.8, the A (alpha) component of the Packed color attribute is valid and all vertex records include the Packed color (a, b, g, r) attribute, even those that also include the Vertex color index attribute. For those vertices that include the Vertex color index attribute, the RGB components of the Packed color attribute will match those of the color specified by the Vertex color index attribute if it was looked up in the color palette. This implies that an application concerned

only with the RGB components of a vertex color can simply reference the Packed color attribute and ignore the Vertex color index attribute in all cases.

All the updated vertex palette records are shown here. The Packed color is shown in **bold font** to emphasize that it is always specified (for both color index and packed color specifications).

Table 146. Vertex with Color Record changes for OpenFlight 15.8 - Packed color field

Data type	Offset	Length	Description
Int	0	2	Vertex with Color Opcode 68
Unsigned Int	2	2	Length - length of the record
Unsigned Int	4	2	Color name index
Int	6	2	Flags (bits, from left to right)
			0 = Start hard edge
			1 = Normal frozen
			2 = No color
			3 = Packed color
			4-15 = Spare
Double	8	8*3	Vertex coordinate (x, y, z)
Int	32	4	Packed color (a, b, g, r) - always specified when the vertex has color
Unsigned Int	36	4	Vertex color index - valid only if vertex has color and Packed color flag is not set

Table 147. Vertex with Color and Normal Record changes for OpenFlight 15.8 - Packed color field*

Data type	Offset	Length	Description
Int	0	2	Vertex with Color and Normal Opcode 69
Unsigned Int	2	2	Length - length of the record
Unsigned Int	4	2	Color name index
Int	6	2	Flags (bits, from left to right)
			0 = Start hard edge

			1 = Normal frozen
			2 = No color
			3 = Packed color
			4-15 = Spare
Double	8	8*3	Vertex coordinate (x, y, z)
Float	32	4*3	Vertex normal (i, j, k)
Int	44	4	Packed color (a, b, g, r) - always specified when the vertex has color
Unsigned Int	48	4	Vertex color index - valid only if vertex has color and Packed color flag is not set
Int	52	4	Reserved

Table 148. Vertex with Color and UV Record changes for OpenFlight 15.8 - Packed color field

Data type	Offset	Length	Description
Int	0	2	Vertex with Color and UV Opcode 71
Unsigned Int	2	2	Length - length of the record
Unsigned Int	4	2	Color name index
Int	6	2	Flags (bits, from left to right) <ul style="list-style-type: none"> 0 = Start hard edge 1 = Normal frozen 2 = No color 3 = Packed color 4-15 = Spare
Double	8	8*3	Vertex coordinate (x, y, z)
Float	32	4*2	Texture coordinate (u, v)
Int	40	4	Packed color (a, b, g, r) - always specified when the vertex has color

Unsigned Int	44	4	Vertex color index - valid only if vertex has color and Packed color flag is not set
--------------	----	---	--

Table 149. Vertex with Color, Normal and UV Record changes for OpenFlight 15.8 - Packed color field

Data type	Offset	Length	Description
Int	0	2	Vertex with Color, Normal and UV Opcode 70
Unsigned Int	2	2	Length - length of the record
Unsigned Int	4	2	Color name index
Int	6	2	Flags (bits, from left to right)
			0 = Start hard edge
			1 = Normal frozen
			2 = No color
			3 = Packed color
			4-15 = Spare
Double	8	8*3	Vertex coordinate (x, y, z)
Float	36	4*3	Vertex normal (i, j, k)
Float	44	4*2	Texture coordinate (u, v)
Int	52	4	Packed color (a, b, g, r) - always specified when the vertex has color
Unsigned Int	56	4	Vertex color index - valid only if vertex has color and Packed color flag is not set
Int	60	4	Reserved

For a complete description of the vertex palette records, see [Vertex Palette Records](#).

D.4. Light Points

The representation of Light Points in OpenFlight version 15.8 is significantly different from prior versions. Previously, all light point attributes were described completely within the primary record

of the light point node.

In OpenFlight version 15.8, light point attributes have been divided into two categories, appearance and behavioral. To accommodate this, two new palettes have been created, the Light Point Appearance Palette and the Light Point Animation palette. A new Indexed Light Point node record has been added that references entries from the Light Point Appearance and Animation palettes. In effect, this moves the light point attributes out of the node record itself into entries of the two palettes and makes it much easier to share common attributes between multiple light points.

Note that the Light Point Record (Opcode 111) used in previous versions of OpenFlight remains valid for applications that require the data in this other format. Creator and the OpenFlight API versions 2.6 support both light point formats.

Following is a description of the new records in OpenFlight version 15.8 used to describe Light Point Palettes and Indexed Light Points.

D.4.1. Light Point Appearance Palette Record

The light point appearance palette record defines the visual attributes of light points.

Table 150. Light Point Appearance Palette Record - New record for OpenFlight 15.8

Data Type	Offset	Length	Description
Int	0	2	Light Point Appearance Palette Opcode 128
Unsigned Int	2	2	Length - length of the record
Int	4	4	Reserved
Char	8	256	Appearance name; 0 terminates
Int	264	4	Appearance index
Int	268	2	Surface material code
Int	270	2	Feature ID
Unsigned Int	272	4	Back color for bidirectional points
Int	276	4	Display mode
			0 = RASTER
			1 = CALLIGRAPHIC
			2 = EITHER
Float	280	4	Intensity - scalar for front colors
Float	284	4	Back intensity - scalar for back color

Float	288	4	Minimum defocus - (0.0 - 1.0) for calligraphic points
Float	292	4	Maximum defocus - (0.0 - 1.0) for calligraphic points
Int	296	4	Fading mode
			0 = Enable perspective fading calculations
			1 = Disable calculations

Table 151. Light Point Appearance Palette Record - New record for OpenFlight 15.8 (Continued)

Data Type	Offset	Length	Description
Int	300	4	Fog Punch mode
			0 = Enable fog punch through calculations
			1 = Disable calculations
Int	304	4	Directional mode
			0 = Enable directional calculations
			1 = Disable calculations
Int	308	4	Range mode
			0 = Use depth (Z) buffer calculation
			1 = Use slant range calculation
Float	312	4	Min pixel size - minimum diameter of points in pixels
Float	316	4	Max pixel size - maximum diameter of points in pixels
Float	320	4	Actual size - actual diameter of points in database units
Float	324	4	Transparent falloff pixel size - diameter in pixels when points become transparent

Float	328	4	Transparent falloff exponent >= 0 - falloff multiplier exponent 1.0 - linear falloff
Float	332	4	Transparent falloff scalar > 0 - falloff multiplier scale factor
Float	336	4	Transparent falloff clamp - minimum permissible falloff multiplier result
Float	340	4	Fog scalar >= 0 - adjusts range of points for punch threw effect.
Float	344	4	Fog intensity
Float	348	4	Size difference threshold - point size transition hint to renderer
Int	352	4	Directionality 0 = OMNIDIRECTIONAL 1 = UNIDIRECTIONAL 2 = BIDIRECTIONAL
Float	356	4	Horizontal lobe angle - total angle in degrees
Float	360	4	Vertical lobe angle - total angle in degrees
Float	364	4	Lobe roll angle - rotation of lobe about local Y axis in degrees
Float	368	4	Directional falloff exponent >= 0 - falloff multiplier exponent 1.0 - linear falloff

Float	372	4	Directional ambient intensity - of points viewed off axis
Float	376	4	Significance - drop out priority for RASCAL lights (0.0 - 1.0)

Table 152. Light Point Appearance Palette Record - New record for OpenFlight 15.8 (Continued)

Data Type	Offset	Length	Description
Int	380	4	Flags (bits, from left to right)
			0 = reserved
			1 = No back color
			TRUE = don't use back color for bidirectional points
			FALSE = use back color for bidirectional points
			2 = reserved
			3 = Calligraphic proximity occulting (Debunching)
			4 = Reflective, non-emissive point
			5-7 = Randomize intensity
			0 = never
			1 = low
			2 = medium
			3 = high
			8 = Perspective mode
			9 = Flashing
			10 = Rotating
			11 = Rotate Counter Clockwise
			Direction of rotation about local Z axis
			12 = reserved
			13-14 = Quality

			0 = Low
			1 = Medium
			2 = High
			3 = Undefined
			15 = Visible during day
			16 = Visible during dusk
			17 = Visible during night
			18-31 = Spare
Float	384	4	Visibility range (> 0.0)
Float	388	4	Fade range ratio - percentage of total range at which light points start to fade (0.0 - 1.0)
Float	392	4	Fade in duration - time it takes (seconds) light point to fade in when turned on
Float	396	4	Fade out duration - time it takes (seconds) light point to fade out when turned off
Float	400	4	LOD range ratio - percentage of total range at which light points LODs are active (0.0 - 1.0)
Float	404	4	LOD scale - size of light point LOD polygon relative to light point diameter

D.4.2. Light Point Animation Record

The light point animation palette record defines the behavioral attributes of light points

Table 153. Light Point Animation Palette Record - New record for OpenFlight 15.8

Data Type	Offset	Length	Description
Int	0	2	Light Point Animation Opcode 129

Unsigned Int	2	2	Length - length of the record
Int	4	4	Reserved
char	8	256	Animation name; 0 terminates
Int	264	4	Animation index
Float	268	4	Animation period in seconds. Note: Rate = 1/Period
Float	272	4	Animation phase delay in seconds - from start of period
Float	276	4	Animation enabled period (time on) in seconds
Float	280	4	Axis of rotation for rotating animation, I
Float	284	4	Axis of rotation for rotating animation, J
Float	288	4	Axis of rotation for rotating animation, K
Int	292	4	Flags (bits, from left to right)
			0 = Flashing
			1 = Rotating
			2 = Rotate counter clockwise
			3-31 = Spare
Int	296	4	Animation type
			0 = Flashing sequence
			1 = Rotating
			2 = Strobe
			3 = Morse code
Int	300	4	Morse code timing
			0 = Standard timing
			1 = Farnsworth timing
Int	304	4	Word rate (for Farnsworth timing)

Int	308	4	Character rate (for Farnsworth timing)
char	312	1024	Morse code string
Int	1336	4	Number of sequences (for Flashing sequence)

The following fields are repeated for each sequence represented in the light point animation palette entry.

In the fields listed below, N ranges from 0 to Number of sequences - 1.

Unsigned Int	1340+(N*12)	4	Sequence StateN - state of sequence N 0 = On 1 = Off 2 = Color change
Float	1344+(N*12)	4	Sequence DurationN - duration of sequence N in seconds
Unsigned Int	1348+(N*12)	4	Sequence ColorN - color for sequence N. Defined if Sequence state is On or Color change

For a complete description of the light point animation palette record, see “Light Point Animation Palette Record” on page 85.

D.4.3. Indexed Light Point Record

The indexed light point record is one of the records that can represent a light point node.

The appearance index specifies an entry in the light point appearance palette that contains the visual attributes of the light point.

The animation index specifies an entry in the light point animation palette that contains the behavioral attributes of the light point.

The palette entries referenced by the indexed light point record describe the visual state of the light point’s child vertices. Only vertex nodes may be children of light point nodes.

Table 154. Indexed Light Point Record - New record for OpenFlight 15.8

Data Type	Offset	Length	Description
Int	0	2	Indexed Light Point Record Opcode 130

Unsigned Int	2	2	Length - length of the record
Char	4	8	7 char ASCII ID; 0 terminates
Int	12	4	Appearance index
Int	16	4	Animation index
Int	20	4	Draw order (for calligraphic lights)
Int	24	4	Reserved

For a complete description of the light point records, see “Indexed Light Point Record” on page 34.

D.4.4. Light Point System Record

The light point system record enables you to collect a set of light points and enable/disable or brighten/dim them as a group.

Table 155. Light Point System Record - New record for OpenFlight 15.8

Data Type	Offset	Length	Description
Int	0	2	Light Point System Record Opcode 130
Unsigned Int	2	2	Length - length of the record
Char	4	8	7 char ASCII ID; 0 terminates
Float	12	4	Intensity
Int	16	4	Animation state
			0 = On
			1 = Off
			2 = Random
Int	20	4	Flags (bits, from left to right)
			0 = Enabled
			1-31 = Spare

For a complete description of the light point system record, see [Light Point System Record](#).

D.4.5. Texture Mapping Palette Record

Parameters for 3 Point Put Texture Mapping (Type 1)

The UV display type field, previously labeled Reserved in prior versions of this document, has been re-labeled in the specification for OpenFlight 15.8. The physical layout of the record was not changed.

Table 156. Parameters for 3 Point Put Texture Mapping (Type 2) - changes for OpenFlight15.8- UV display type field (re-labeled)

Data Type	Offset	Length	Description
Int	388	4	UV display type
			1 = XY
			2 = UV

Parameters for 4 Point Put Texture Mapping (Type 2)

The following fields were added to the end (at the specified offsets) of the parameter subrecord.

Table 157. Parameters for 4 Point Put Texture Mapping (Type 2) - changes for OpenFlight15.8 - New Fields

Data Type	Offset	Length	Description
Float	576	4	U Repetition
Float	580	4	V Repetition

The UV display type field, previously labeled Reserved in prior versions of the document, has been re-labeled in the specification for Open Flight 15.8. The physical layout of the record was not changed.

Table 158. Parameters for 4 Point Put Texture Mapping (Type 2) - changes for OpenFlight15.8 - UV display type field (re-labeled)

Data Type	Offset	Length	Description
Int	436	4	UV display type
			1 = XY
			2 = UV

Annex E: Summary of Changes Version 16.0

E.1. E - Overview

This section describes the changes in the OpenFlight Scene Description between versions 15.8 and 16.0 as well as the errors contained in previous versions of this document that have been corrected in this version.

OpenFlight version 16.0 coincides with MultiGen Creator version 3.0 and the OpenFlight API version 3.0. The changes made for this version are:

[External Reference Record](#)

[Face Record](#)

[Mesh Record](#)

[Light Point Appearance Palette Record](#)

[Shader Palette Record](#)

[Texture Attribute File](#)

[Texture Mapping Palette Record](#)

E.2. Document Corrections

The errors corrected in this version of the document are described in the sections that follow.

E.2.1. Header Record

The value corresponding to User defined ellipsoid for the Earth ellipsoid model field has been corrected. It was previously listed as having a value of 5. The correct value is -1. The corrected value is shown in **bold font**.

Table 159. Header Record error corrected in OpenFlight 16.0 specification - Earth ellipsoid model field (corrected)

Data Type	Offset	Length	Description
Int	268	4	Earth ellipsoid model
			0 = WGS 1984
			1 = WGS 1972
			2 = Bessel
			3 = Clarke 1866
			4 = NAD 1927

			-1 = User defined ellipsoid
--	--	--	------------------------------------

E.2.2. Face Record

The possible values listed for the Draw type field have been corrected. The affected values are shown in **bold font**.

Table 160. Face Record error corrected in OpenFlight 16.0 specification - Draw type field (corrected)

Data Type	Offset	Length	Description
Int	18	1	Draw type
			0 = Draw solid with backface culling
			1 = Draw solid, no backface culling
			2 = Draw wireframe and close
			3 = Draw wireframe
			4 = Surround with wireframe in alternate color
			8 = Omnidirectional light
			9 = Unidirectional light
			10 = Bidirectional light

For a complete description of the face record, see “Face Record” on page 26.

E.2.3. Mesh Record

The Reserved field at offset 12, previously omitted in prior versions of this document, has been documented in the specification for OpenFlight 16.0. The offsets of fields following this field have been adjusted accordingly.

Table 161. Mesh Record error corrected in OpenFlight 16.0 specification - Reserved field (documented)

Data Type	Offset	Length	Description
Int	12	4	Reserved

The possible values listed for the Draw type field have been corrected. The affected values are shown in **bold font**.

Table 162. Mesh Record error corrected in OpenFlight 16.0 specification - Draw type field (corrected)

Data Type	Offset	Length	Description
-----------	--------	--------	-------------

Int	18	1	Draw type
			0 = Draw solid with backface culling
			1 = Draw solid, no backface culling
			2 = Draw wireframe and close
			3 = Draw wireframe
			4 = Surround with wireframe in alternate color
			8 = Omnidirectional light
			9 = Unidirectional light
			10 = Bidirectional light

For a complete description of the mesh record, see [Mesh Record](#).

E.2.4. Switch Record

The order of the fields were corrected. The affected fields are shown here.

Table 163. Switch Record error corrected in OpenFlight 16.0 specification - field order (corrected)

Data Type	Offset	Length	Description
Int	20	4	Number of masks
Int	24	4	Number of words per mask - the number of 32 bit words required for each mask, calculated as follows: (number of children / 32) + X where X equals: 0 if (number of children modulo 32) is zero 1 if (number of children modulo 32) is nonzero

For a complete description of the switch record, see [Switch Record](#).

E.2.5. Texture Mapping Palette Record

The parameters for warped mapping in the texture mapping palette record were corrected. The 128 byte 4x4 Trackplane to XY plane transformation matrix was erroneously listed where an 8 byte reserved field was located. The entire record is shown here. The corrected field and offsets are shown in **bold font**:

Table 164. Parameters for Warped Mapping error corrected in OpenFlight 16.0 specification - Reserved field (corrected)

Data Type	Offset	Length	Description
Int	X+0	4	Active geometry point
			0 = First warp FROM point
			1 = Second warp FROM point
			2 = Third warp FROM point
			3 = Fourth warp FROM point
			4 = Fifth warp FROM point
			5 = Sixth warp FROM point
			6 = Seventh warp FROM point
			7 = Eighth warp FROM point
			8 = First warp TO point
			9 = Second warp TO point
			10 = Third warp TO point
			11 = Fourth warp TO point
			12 = Fifth warp TO point
			13 = Sixth warp TO point
			14 = Seventh warp TO point

			15 = Eighth warp TO point
Int	X+4	4	Warp tool state
			0 = Start state - no points entered
			1 = One FROM point entered
			2 = Two FROM point entered
			3 = Three FROM point entered
			4 = Four FROM point entered
			5 = Five FROM point entered
			6 = Six FROM point entered
			7 = Seven FROM point entered
			8 = All FROM point entered
Int	X+8	8	Reserved
Double	X+16	8*8*2	FROM points transformed to XY plane by above matrix. 8 FROM points are ordered 1, 2, ... 8. Each point is (x, y)
Double	X+144	8*8*2	TO points transformed to XY plane by above matrix. 8 TO points are ordered 1, 2, ... 8. Each point is (x, y)

For a complete description of the texture mapping palette record, see [Texture Mapping Palette Record](#).

E.2.6. Indexed String Record

The length of the Index field has been corrected. It was previously listed as 2 bytes. The correct length is 4 bytes. The corrected field and length are shown in **bold font**:

Table 165. Indexed String Record

Data Type	Offset	Length	Description
Int	0	2	Indexed string Opcode 132
Unsigned Int	2	2	Length - length of the record
Unsigned Int	4	4	Index
Char	8	Length - 8	ASCII string; 0 terminates

E.2.7. Bounding Convex Hull Record

The description of this previously undocumented record has been added to the specification. For a complete description of this record, see [Bounding Convex Hull Record](#).

E.2.8. Bounding Histogram Record

The description of this previously undocumented record has been added to the specification. For a complete description of this record, see [Bounding Histogram Record](#).

E.3. Format Changes

E.3.1. External Reference Record

The Flags field of the external reference record has been modified to include a new bit, Shader palette override, which is used to specify that the shader palette override those contained in the master file. The new bit is shown in **bold font**.

Table 166. External Reference Record changes for OpenFlight 15.8 - Flags field

Data Type	Offset	Length	Description
Int	208	4	Flags (bits, from left to right)
			0 = Color palette override
			1 = Material palette override
			2 = Texture and texture mapping palette override
			3 = Line style palette override
			4 = Sound palette override

			5 = Light source palette override
			6 = Light point palette override
			7 = Shader palette override
			8-31 = Spare

For a complete description of the external reference record, see [External Reference Record](#).

E.3.2. Face Record

The face record has been modified to include a new attribute, Shader index, which is used to specify the shader (if any) that is applied to the face.

Table 167. Face Record changes for OpenFlight 16.0 - Flags field

Data Type	Offset	Length	Description
Int	78	2	Shader index, -1 if none

For a complete description of the face record, see [Face Record](#).

E.3.3. Mesh Record

Similar to the Face record described above, the mesh record has been modified to include a new attribute, Shader index, which is used to specify the shader (if any) that is applied to the mesh.

Table 168. Mesh Record changes for OpenFlight 16.0 - Flags field

Data Type	Offset	Length	Description
Int	78	2	Shader index, -1 if none

For a complete description of the mesh record, see [Mesh Record](#).

E.3.4. Light Point Appearance Palette Record

The light point appearance palette record has been modified to include a new attribute, Texture pattern index, which is used to specify the texture (if any) that is applied to the light point appearance.

Table 169. Light Point Appearance Record changes for OpenFlight 16.0 - Texture pattern index field

Data Type	Offset	Length	Description
Int	408	2	Texture pattern index, -1 if none
Int	410	2	Reserved

For a complete description of the light point appearance palette record, see [Light Point Appearance](#)

E.3.5. Shader Palette Record

The shader palette contains descriptions of shaders used while drawing geometry. It is composed of an arbitrary number of shader palette records. The shader palette records must follow the header record and precede the first push.

Table 170. Shader Palette Record - New record for OpenFlight 16.0

Data Type	Offset	Length	Description
Int	0	2	Shader Opcode 133
Unsigned Int	2	2	Length - length of the record
Int	4	4	Shader index
Int	8	4	Shader type
			0 = Cg
			1 = CgFX
			2 = OpenGL Shading Language
char	12	1024	Shader name; 0 terminates
char	1036	1024	Vertex program file name; 0 terminates (Cg Shader type specific)
char	2060	1024	Fragment program file name; 0 terminates (Cg Shader type specific)
Int	3084	4	Vertex program profile (Cg Shader type specific)
Int	3088	4	Fragment program profile (Cg Shader type specific)
char	3092	256	Vertex program entry point (Cg Shader type specific)
Char	3348	256	Fragment program entry point (Cg Shader type specific)

E.4. Texture Attribute File

The Wrap method fields (Wrap method u,v, Wrap method u and Wrap method v) have been changed to include a new possible value, Mirrored repeat as shown here. This new value is shown in **bold font**:

Table 171. Texture Attribute File Format changes for OpenFlight 16.0 Wrap method fields

Data Type	Offset	Length	Description
Int	36	4	Wrap method u,v - only used when either Wrap method u or Wrap method v is set to None
			0 = Repeat
			1 = Clamp
			4 = Mirrored Repeat
Int	40	4	Wrap method u
			0 = Repeat
			1 = Clamp
			3 = None - use Wrap method u,v
			4 = Mirrored Repeat

Table 172. Texture Attribute File Format changes for OpenFlight 16.0 Wrap method fields

Data Type	Offset	Length	Description
Int	44	4	Wrap method v
			0 = Repeat
			1 = Clamp
			3 = None - use Wrap method u,v
			4 = Mirrored Repeat

The Environment type field has been changed to include a new possible value, Add as shown here. This new value is shown in **bold font**.

Table 173. Texture Attribute File Format changes for OpenFlight 16.0Environment type field

Data Type	Offset	Length	Description
Int	60	4	Environment type
			0 = Modulate
			1 = Blend
			2 = Decal

			3 = Replace
			4 = Add

E.4.1. Texture Mapping Palette Record

Parameters for 3 Point Put Texture Mapping (Type 1)

The following fields were added to the end (at the specified offsets) of the parameter subrecord.

Table 174. Parameters for 3 Point Put Texture Mapping (Type 1) - changes for OpenFlight 16.0 - New Fields

Data Type	Offset	Length	Description
Float	392	4	U Repetition
Float	396	4	V Repetition

E.5. Index

NOTE Index removed during conversion to Asciidoc.

Annex F: Revision History

Date	Release	Editor	Primary clauses modified	Description
2016-02-12	1.0	C. Reed	Many	Many edits, added requirements, requirements classes, and Annex A
2016-02-26	1.0	C. Reed	Annex A	Finish ATS
2017-12-28	1.1	C. Reed	Various	Minor edits. Change URI to be 1.1 consistent.
2019-12-16	1.2	C. Reed	Various	Changes for version 1.2