Open Geospatial Consortium

Submission Date: 2015-03-02

Approval Date: 2015-05-23

Publication Date: 2015-07-22

External identifier of this OGC® document: http://www.opengis.net/doc/BP/cdb-vol2/1.0

Internal reference number of this OGC® document: 15-004

Version: 1.0.0

Category: OGC® Best Practice

Editor: David Graham

OGC Common DataBase
Volume 2 Appendices

Warning

This document defines an OGC Best Practices on a particular technology or approach. 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.

The Common DataBase (CDB) Specification provides the means for a single, versionable, simulation-rich, synthetic representation of the earth. A database that conforms to this Specification is referred to as a Common DataBase or CDB. A CDB provides for a synthetic environment repository that is plug-and-play interoperable between database authoring workstations. Moreover, a CDB can be used as a common on-line (or runtime) repository from which various simulator client-devices can simultaneously retrieve and modify, in real-time, relevant information to perform their respective runtime simulation tasks; in this case, a CDB is plug-and-play interoperable between CDB-compliant simulators. A CDB can be readily used by existing simulation client-devices (legacy Image Generators, Radar simulator, Computer Generated Forces, etc.) through a data publishing process that is performed on-demand in real-time.

The application of CDB to future simulator architectures will significantly reduce runtime-source level and algorithmic correlation errors, while reducing development, update and configuration management timelines. With the addition of the HLA/FOM and DIS protocols, the application of the CDB Specification provides a Common Environment to which inter-connected simulators share a common view of the simulated environment.

The CDB Specification is an open format Specification for the storage, access and modification of a synthetic environment database. The Specification defines the data representation, organization and storage structure of a worldwide synthetic representation of the earth as well as the conventions necessary to support all of the subsystems of a full-mission simulator. The Specification makes use of several commercial and simulation data formats endorsed by leaders of the database tools industry.

The CDB synthetic environment is a representation of the natural environment including external features such as man-made structures and systems. It encompasses the terrain relief, terrain imagery, three-dimensional (3D) models of natural and man-made cultural features, 3D models of dynamic vehicles, the ocean surface, and the ocean bottom, including features (both natural and man-made) on the ocean floor. In addition, the synthetic environment includes the specific attributes of the synthetic environment data as well as their relationships.

A CDB contains datasets organized in layers, tiles and levels-of-detail; together, these datasets represent the features of a synthetic environment for the purposes of distributed simulation applications. The organization of the synthetic environmental data in a CDB is specifically tailored for real-time applications.

i. Keywords

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

ogcdoc, OGC document, CDB, Common DataBase, simulation

ii. Preface

The industry-maintained Common CDB has been discussed and demonstrated at OGC Technical Committee meetings beginning in September, 2013.

At the suggestion of several attendees at the first CDB ad-hoc meeting in September, 2014, the current version of the existing CDB specification has been slightly reformatted for publication as an OGC Best Practice.

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.

iii. Submitting organizations

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

CAE Inc.

iv. Submitters

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

Name	Affiliation
David Graham	CAE Inc.
Mike Lokuta	CAE USA, Inc

1. Scope

The full CDB specification, in its current, industry-maintained format and version, addresses the interoperability challenge of full plug-and-play interoperability and re-use of synthetic environment databases used for high fidelity simulation and mission rehearsal.

The first CDB specification was developed under a competitive contract awarded to CAE to meet requirements of the United States Special Operations Command. The CDB Specification was required to be open and non-proprietary as part of the original requirements. The revision history of the industry-maintained specification is contained in the following document sections.

The CDB specification was been widely implemented by multiple, independent industry contractors for end-user simulation and mission rehearsal customers in many different countries over a period of ten years.

For ease of editing and review, the specification has been separated into two Volumes. Volume 1 contains the main body of the specification, and Volume 2 contains the appendices. Nevertheless, the documents remain large and verbose, as the current, industry maintained specification has functioned as a data model, an encoding specification, and an engineering tutorial on how to implement this new and different simulation synthetic environment paradigm.

The remainder of this document comprises Volume 2.

2. Common DataBase (CDB) Version 3.2 Specification Volume 2

Table of Contents

List of Figures

Appendix A

A. Guidelines, Clarifications, Rationales, Primers, Information

A.1 Guideline: Creating a 3D Model for a Powerline Pylon

The goal of this guideline is to model a typical high voltage electrical pylon resembling the one in this figure. This guideline is based on version 3.1 of the Specification, but is applicable to version 3.0 as well.

Figure A-1: Typical Electrical Pylon

A.1.1 Pylon Model Orientation

The front (and back) of a powerline pylon is aligned with the general direction of the attached wires as illustrated below.

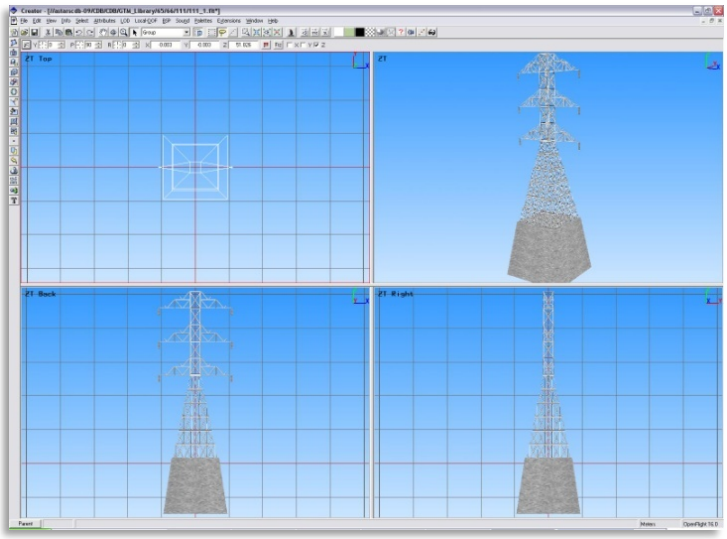


Figure A-2: Pylon Orientation

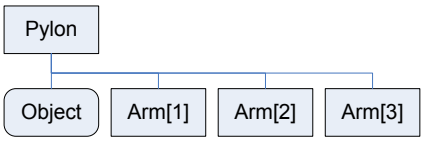
The above snapshot is similar to the one found in Figure 6–10 of the Specification.

A.1.2 OpenFlight Graph

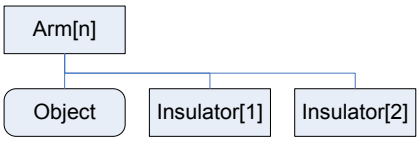
The graph of the above pylon exposes the 3 cross-arms, each with 2 insulators where wires are attached. Here are the names of the components that are used to model this power pylon:

- Pylon (global zone)
- Arm (horizontal cross-arm at the top of the structure)
- Insulator (ceramic insulator string attached at the end of each arm)

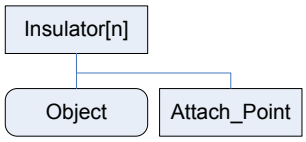
These 3 names are used to create CDB Zones as explained in section 6.5 of the Specification. Here is the first level of the resulting graph.



The rounded rectangle named Object is an OpenFlight object node containing the geometry of the concrete base and lattice steel structure of the pylon, but excluding the geometry of the cross-arms. Arm[1] is the lowest cross-arm; Arm[2] is the middle one; Arm[3] is the top one. Each arm is then made of a steel structure and 2 insulators.



Again, Object represents the steel structure of the cross-arm without the insulators. When looking at one of the cross-arm of the power pylon from the back, Insulator[1] is to the left while Insulator[2] is to the right. Finally, each insulator has an attach point to indicate where to connect an eventual wire.



The node Object contains the geometry of the insulator.

As explained in section 6.8 of the Specification, the resulting list of paths is as follow:

- \Pylon
- \Pylon\Arm[1]
- \Pylon\Arm[1]\Insulator[1]
- \Pylon\Arm[1]\Insulator[1]\Attach_Point
- \Pylon\Arm[1]\Insulator[2]
- \Pylon\Arm[1]\Insulator[2]\Attach_Point
- \Pylon\Arm[2]
- \Pylon\Arm[2]\Insulator[1]
- \Pylon\Arm[2]\Insulator[1]\Attach_Point
- \Pylon\Arm[2]\Insulator[2]
- \Pylon\Arm[2]\Insulator[2]\Attach_Point
- \Pylon\Arm[3]
- \Pylon\Arm[3]\Insulator[1]
- \Pylon\Arm[3]\Insulator[1]\Attach_Point
- \Pylon\Arm[3]\Insulator[2]
- \Pylon\Arm[3]\Insulator[2]\Attach_Point

Note the presence of a total of 6 attach points (1 attach point per insulator × 2 insulators per cross-arm × 3 cross-arms per pylon = 6 attach points per pylon). Even though all attach points have the same name, there is a unique path to reach each one. For this reason, there is no ambiguity identifying and locating each point.

A.1.3 Attach Point Orientation

When creating the attach point of the insulator, pay attention to its orientation. Since the cable attaches underneath the insulator, the Z-axis of the local coordinate system (LCS) must be pointing down. To achieve a proper positioning of the attach point, the modeler usually inserts two transformations in the node, one translation and one rotation. The translation positions the point underneath the insulator while the rotation changes the orientation of the Z-axis. Make sure to leave the Y-axis in the direction of the wire as in the figure below.

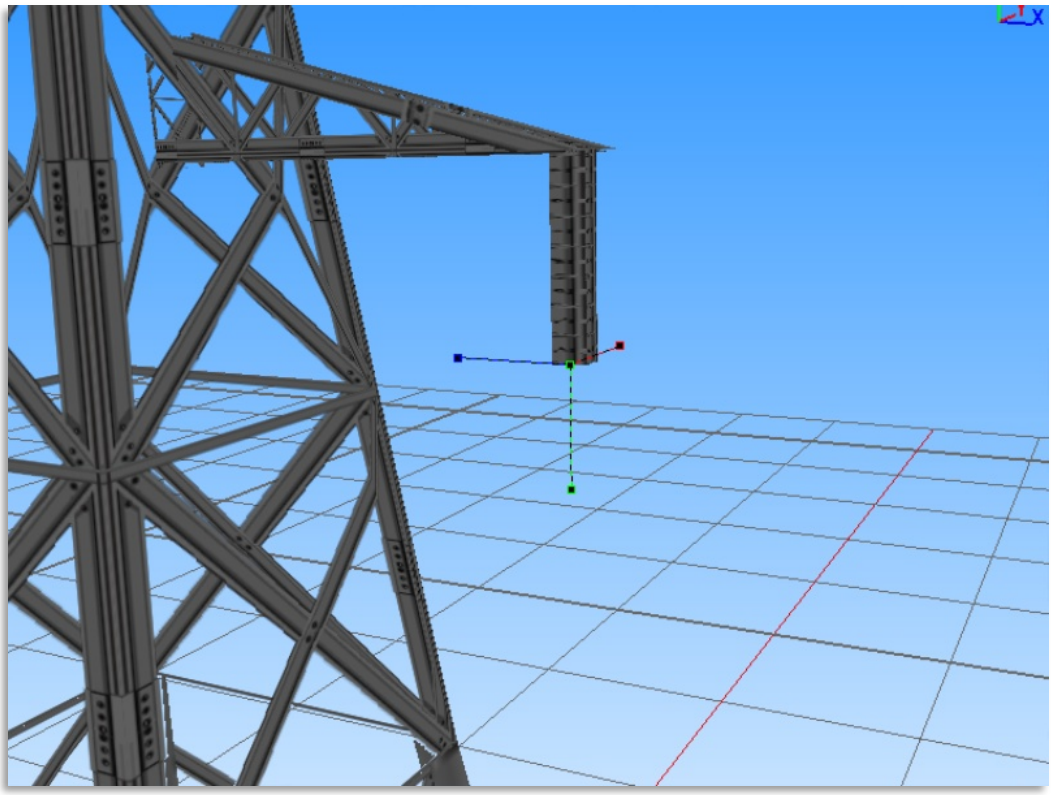


Figure A-3: Attach Point Orientation

In this figure, the position and orientation of the attach point is identified by the blue-red-green axis system beneath the insulator. The Y-axis is in red and points in the same direction as the model's Y-axis, which is toward the front of the model. The Z-axis is in green and points down indicating that wires attach under the insulator.

A.2 Guideline: Generating Wires between Pylons of a Powerline

This guideline is intended for both modelers and developers responsible for the creation of:

- CDB content such as 3D models representing pylons
- Tools used to generate the Powerline Network datasets
- Client-devices that use the Powerline Network datasets to generate pylons and wires along the transmission line.

The article is based on version 3.0 of the CDB Specification.

A.2.1 Powerline Network Attributes

The table below is the collection of class and instance-level attributes from tables 5-46 and 5-47 of the CDB Specification.

Table A-1: Powerline Attributes

Required Attributes	Optional Attributes
CMIX	AHGT
CNAM	AO1
DIR	BBH
EJID	BBL
FACC	BBW
FSC	BSR
JID	HGT
LENL	MODL
RTAI	MODT
SJID	SCALn
WGP	

The occurrence of some of the optional attributes depends on the occurrence of other optional attributes. In particular, when MODL is present, other attributes become required while others remain optional. The table below provides the relation between MODL and other attributes.

Table A-2: MODL-related Attributes

Required	Optional
BSR	AO1
HGT	BBH, BBL, BBW
MODT	SCALn

As a result of the above tables, a CDB-compliant Powerline Network dataset requires 11 mandatory attributes (listed in the first column of Table A-1). Optionally, when a 3D model representing a pylon is provided, 4 additional attributes are required (MODL obviously, plus BSR, HGT, and MODT) and 5 others remain optional (AO1, BBH, BBL, BBW, and SCALn).

A.2.2 Generation of HGT

The HGT attribute represents a special case because table 5-47 suggests that the attribute is optional while, in fact, it should always be present. If you carefully read its description in paragraph 5.3.1.2.3.17, you realize that HGT is required in both the lineal and figure point features of the Powerline Network.

In the lineal features, HGT represents the average height above ground of the powerline when no MODL is specified, as suggested by the discussion about HGT in section 5.3.1.17 of the Specification. In the figure point features, HGT represents the height above ground of the pylon, whether or not a MODL is provided. In either file, when MODL is supplied, HGT represents the height of the 3D model above the ground.

You should read guideline A.3 for a complete discussion about HGT

A.2.3 Pylon Orientation

If the orientation of the pylon is specified by AO1, then use the value as-is. If the orientation is not specified, the client device must compute its value using the orientation of the segments of the lineal that are adjacent to the pylon. In the case of the first and last segments, the orientation of the segment is also the orientation of the pylon. For the other segments, the orientation of the pylon is the average of the orientation of the two adjacent segments.

A.2.4 Number of Wires

When no MODL is provided at all – meaning no MODL for the lineal and none for the figure points – and because there is no attribute specifying the number of wires along the transmission line, the client device must assume a generic powerline with two wires separated by a width of WGP meters connecting generic posts (simple pylons) of HGT meters high.

When a common MODL is specified for the whole lineal and no figure points are provided, it is possible to determine the number of wires by counting the number of attach points in the 3D model. Refer to guideline A.1.2 for details on how to detect attach points.

If specific MODLs are defined through figure points, the number of attach points in each 3D model of the collection of all MODLs referenced by the powerline network must be identical. For instance, if the lineal refers to a generic pylon supporting 4 wires, then all specific pylons referenced as figure points must also support 4 wires. Furthermore, the general configuration of all pylons must be identical. If the general pylon supports 6 wires configured as a matrix of 2 wires horizontally by 3 wires vertically, then all specific pylons must also share the same configuration.

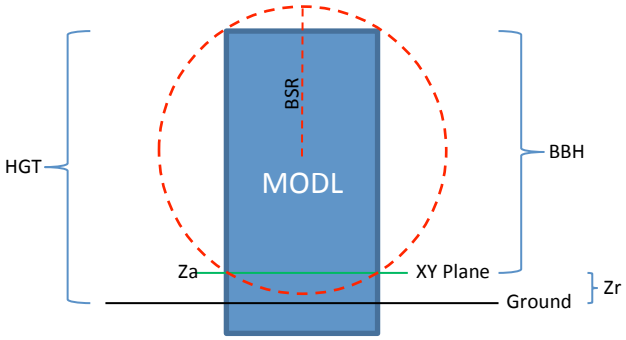
A.2.5 How to Connect Wires to Attach Points

If the client device has a single generic pylon along the line, then there is no problem connecting wires and attach points. That is when multiple pylons are used along the same line that problems arise. The client has to match attach points from one type of pylon to attach points on another pylon that may be of a different type. In CDB Spec 3.0, the algorithm to determine how to connect pylons of different types is left to the client device. The future version of CDB Specification will provide a robust and deterministic approach on how to connect the wires.

A.3 Guideline: How to Interpret the AHGT, HGT, BSR, BBH, and Z Attributes

The goal of this guideline is to promote a correct use of five CDB attributes: AHGT, HGT, BSR, BBH, and Z. The article is aimed to both developers and users of content creation tools as well as developers of client applications. The guideline is based on version 3.1 of the Specification, but remains applicable to version 3.0 as well.

A picture being worth a thousand words, the following diagram should help understand the relations between the AHGT, HGT, BSR, BBH, and Z attributes.



Here is a reminder of what these attributes are. The complete definitions can be found in Section 5.3.1.3, CDB Attributes.

- AHGT (Absolute Height) is a flag to interpret correctly the value of the Z coordinate of a feature. When false, the value of Z is relative to the ground (Zr); when true, Z is the absolute altitude (Za).
- AHGT is not related with HGT even though their names are similar.
- HGT (Height Above Surface Level) is the distance from the top of the model to the ground.
- BBH (Bounding Box Height) is the distance from the top of the model to its XY plane.
- BSR (Bounding Sphere Radius) encompasses the portion of the model that is above its XY plane.
- Z is the altitude of a feature, either absolute or relative to the ground.

In the diagram above, a model (MODL) is positioned above the ground. This is indicated by the fact that the model's XY plane does not lie directly on the ground. The distance above the

ground is represented by Zr. The diagram clearly shows the relation between HGT, BBH, and Zr.

$$HGT = BBH + Zr$$

When the value of Zr is not readily available from the instance of the feature itself (because AHGT is true), it can be computed using the ground height (Gh).

$$Zr = Za - Gh$$

The BBH attribute is optional and defaults to twice the value of BSR, which is mandatory for a MODL model.

$$default\ BBH = 2 \times BSR$$

$$default\ BBH \geq real\ BBH$$

A.3.1 Typical Use-case

Typically, a model is positioned relative to the ground without any offset. As a result, AHGT is false, and Zr is set to zero. Hence…

$$HGT = BBH$$

A.3.2 Light Points

In the case of airport and environmental light points, no model of a light fixture is provided (the MODL attribute is not allowed). Hence…

$$BSR = 0 \rightarrow BBH = 0$$

Since, in version 3.1 of the CDB Specification, the light point datasets do not allow the HGT attribute, the client application may have to compute its value using the equation given previously…

$$HGT = BBH + Zr$$

where BBH is null.

$$HGT = Zr$$

And if the light point is positioned at an absolute height (AHGT is true), then…

$$HGT = Za - Gh$$

A.3.3 Recommendation

Refrain from using AHGT. There are several advantages to leave this flag to false. First, it facilitates the creation of CDB datasets that are independent of each others. When the Z coordinate (altitude) of a feature is relative to the ground, the elevation dataset can be updated without the need to recompute and update all features that have an absolute altitude.

Second, when a feature has an absolute altitude, it is possible that it will end up being *displayed* below the ground by a given client. How is this possible? Isn't it an error in the database itself? No, this is not an error. It is perfectly possible to create content that is valid and – still – produce an incorrect result at the client level. Consider a feature that is positioned with an absolute height in a valley between two mountains of a high resolution terrain profile. At coarse LOD of terrain elevation, the valley and the mountains may (and will) be flattened producing a terrain skin that may no longer pass underneath the feature. Now imagine a client that uses that coarse LOD of elevation to create a terrain skin and then draw the feature at its absolute altitude, which happen to be underneath the terrain skin. The feature will not be visible or will be partially occluded by the terrain.

These reasons explain why the use of the AHGT flag should be avoided whenever possible.

A.3.4 When should AHGT be used?

Limit the use of AHGT to data whose source is inherently absolute. Such source data include geodetic marks or survey marks that provide a known position in terms of latitude, longitude, and altitude. Good examples of such markers are boundary markers between countries.

A.4 Guideline: How to Model a Wind Turbine

This text proposes a way to create a 3D model representing an articulated wind turbine. The articulations of interest are the yaw control to orient the turbine in the direction of the wind, the roll control to allow rotation of the rotor, and, optionally, the pitch control to change the orientation of the blades, if needed.

Beside  is a typical Horizontal Axis Wind Turbine. The components of interest are the following:

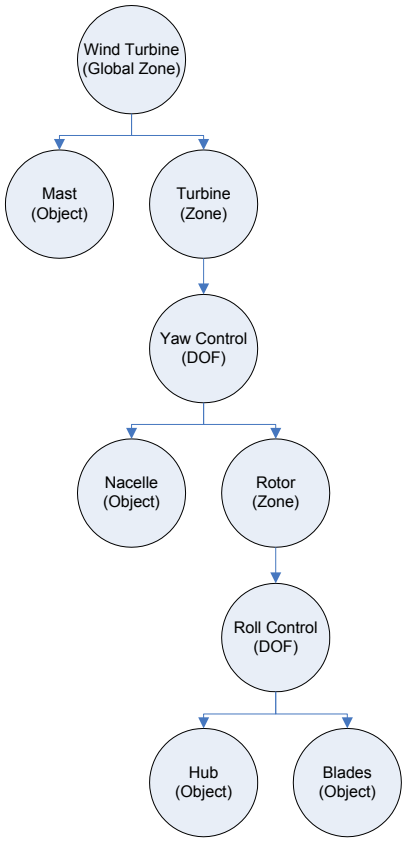
- Turbine
- Rotor
- Blade

Looking at appendix F – CDB Model Components – we note that Turbine is not listed and, consequently, will be proposed for addition to future version of the CDB Specification.

Appendix N provides the proper FACC-FSC code for a Wind Turbine, AD010-005. The code indicates the presence of a man-made point feature.

A = Culture
D = Power Generator
010 = Power Plant
005 = Wind

The hierarchy graph of the OpenFlight model could look like the one on the right. If individual control of the pitch of each blade is required, the Blades object (the lower right node) could be replaced with three (3) sub-trees each containing a Blade zone, a DOF node, and an object node.



With the proposed layout, a client device will detect the presence of a wind turbine through its FACC, and recognize and control two articulations, the Turbine Yaw angle, and Rotor Roll angle.

A last note: to comply with the prescribed orientation of the CDB coordinate system as defined in section 6.3, the rotor must represent the front of the wind turbine (and not its right side).

Reference: http://en.wikipedia.org/wiki/Wind_turbine

A.5 Guideline: Handling of Model Interiors

CDB 3.1 introduces the concept of the interior of a 3D model. The concept is developed in section 6.18, Model Interior, of volume 1 of CDB Specification 3.1. The following text serves as a complement to the specification to understand how the concept has been developed and how model interior is intended to be used.

A.5.1 Relationship between Model Shell and Model Interior

The ModelInteriorGeometry dataset is a subordinate dataset of the '*regular*' ModelGeometry dataset. It depends directly on it. This is best illustrated by an example.

LOD	ModelGeometry (Shell)	ModelInteriorGeometry (Interior)
…	-	-
0	-	-
1	-	-
2	Coarsest Shell	-
3	-	-
4	-	-
5	-	-
6	Medium Shell	Medium Interior
7	-	-
8	Fine Shell	Fine Interior
9	-	-
10	Finest Shell	Finest Interior
11	-	-
12	-	-
13	-	-
14	-	-
15	-	-
…	-	-

In the above table, the **Shell** column represents what is called the '*regular*' ModelGeometry dataset. In this example, the model appears at LOD 2, a better version exists at LOD 6, an even better at LOD 8, and finally, the most detailed shell is at LOD 10. The **Interior** column shows 3 different LODs of interiors. There cannot be more Interior LODs than Shell LODs. Also, once an interior is provided (here at LOD 6), it must be provided for all subsequent (finer) LODs of the shell (LOD 8 and 10). Which means… interior at LOD 8 and 10 must exist.

A.5.2 Detecting Presence of a Model Interior

It is expected that a client will first request the shell of the model, then discover that the model has an interior because of the presence of a CDB Zone whose name is Interior (see 6.18.2,

Pseudo-Interior), and then decide if the pseudo interior is sufficient for the application or if the real interior is necessary.

A.5.3 Access of a Model Interior

Client applications that are interested in 3D models will typically perform the following sequence of actions:

1. Load the GS Features of a tile
2. Load the GS and GT Models referenced by the GS Features
3. For each model, traverse its graph and detect the presence of an optional Interior (Zone name = Interior)
4. Decide to load the corresponding Interior (or not)

Interior datasets exists for both geospecific and geotypical models; hence, all features can be represented by a 3D model and all 3D models can have a separately modeled interior. Note the symmetry between the file names of shell and interior datasets. For geospecific models, the names of geometry files are…

- `GeoCell_D300_S001_T001_Lxx_Ux_Rx_FACC_FSC_MODL.flt`
- `GeoCell_D305_S001_T001_Lxx_Ux_Rx_FACC_FSC_MODL.flt`

For geotypical models, the file names become…

- `D510_S001_T001_Lxx_FACC_FSC_MODL.flt`
- `D515_S001_T001_Lxx_FACC_FSC_MODL.flt`

Note that in both cases, the only difference between the name of the shell and the name of the corresponding interior is the dataset code; and in both cases, a value of 5 is added to the '*regular*' ModelGeometry dataset code.

A.5.4 UHRB vs CDB Object Models

To help understand how CDB Model Interior maps to UHRB concepts, three (3) diagrams are provided below. The first two diagrams illustrate the UHRB Object Model[1] while the third diagram presents the corresponding CDB Object Model.

The first diagram is the UHRB Class Diagram presented in Figure A-4 below. The class diagram presents twelve classes of which eight are concrete classes that can be used to represent tangible objects. The UHRB_EDM_COMPLEX_FEATURE class implements an extension mechanism that is not required in the context of the CDB Specification. The remaining seven UHRB classes will be mapped to CDB zones.

[1] The two UHRB diagrams presented here come from the document entitled UHRB_2_Object_Model.pdf available on the OneSAF web site: www.onesaf.net.

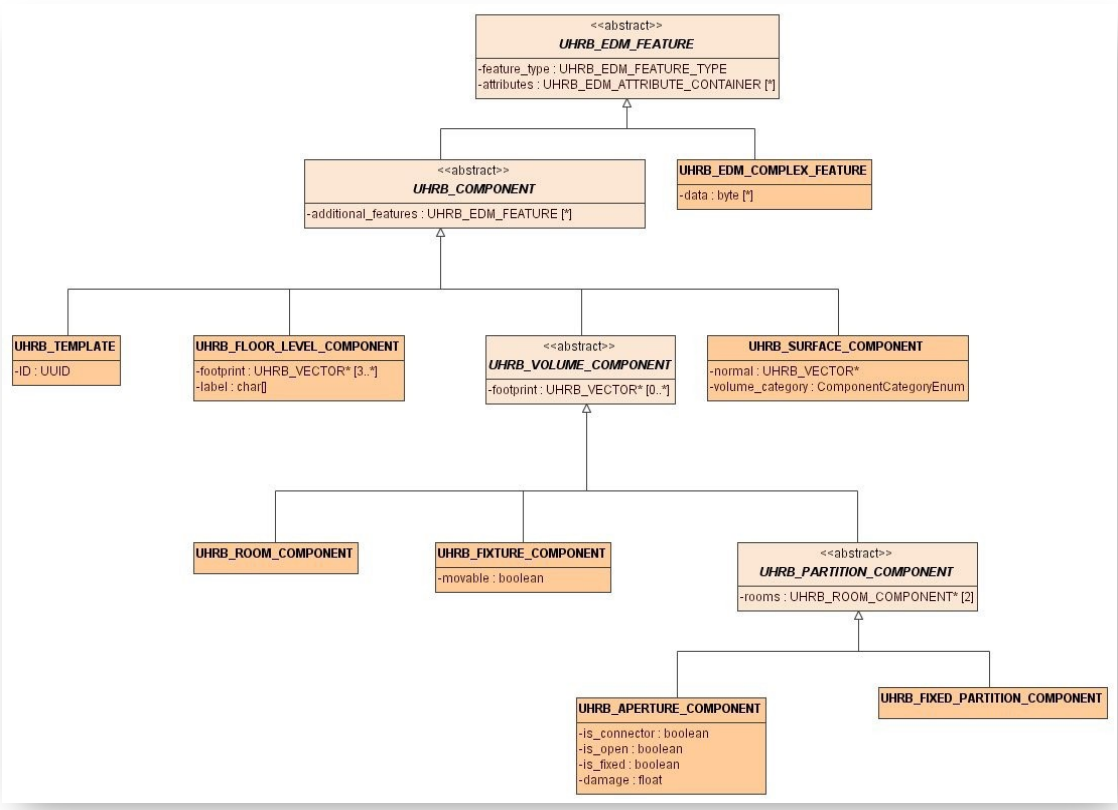


Figure A-4: UHRB Class Diagram

The second diagram is the UHRB Association Diagram of Figure A-5; it shows all permissible associations between the UHRB classes.

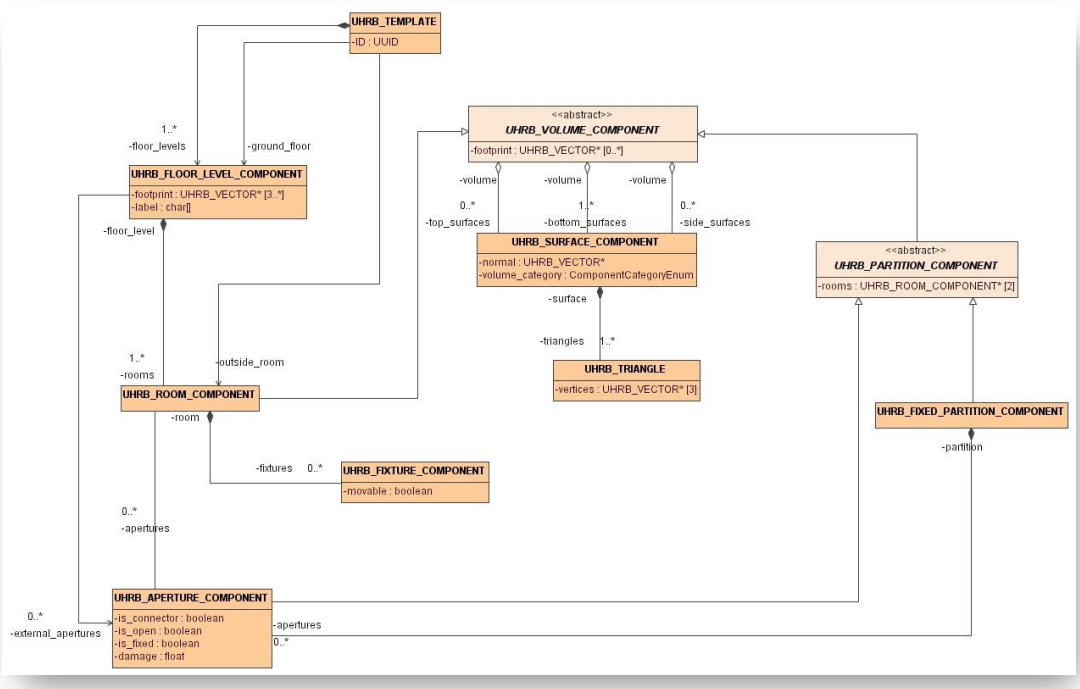


Figure A-5: UHRB Association Diagram

The third diagram, in Figure A-6 below, presents the Object Model proposed by CDB Model Interior objects. The UML diagram is both the class and association diagram of CDB zones listed in table 6-27 of section 6.18.5 of CDB 3.1.

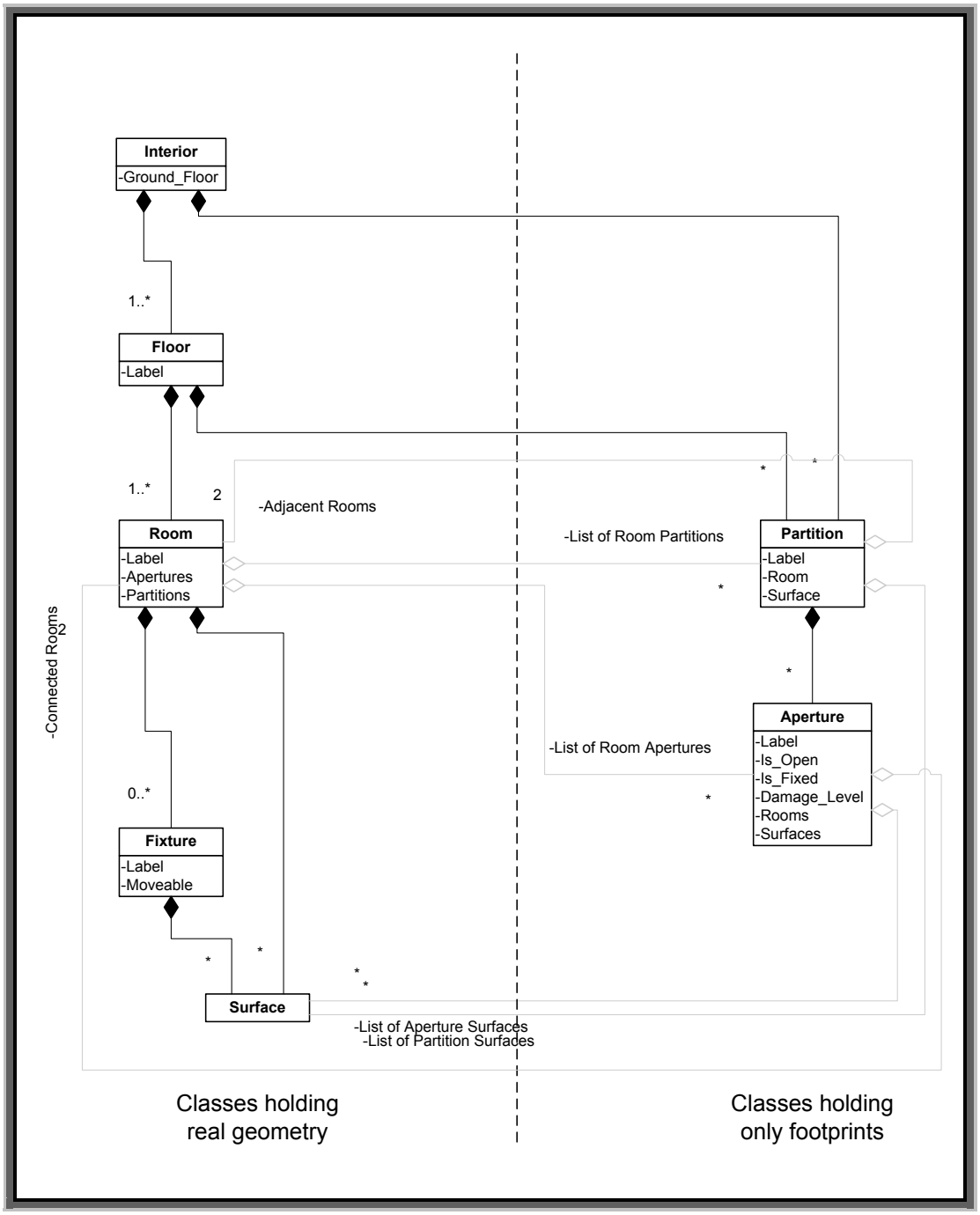


Figure A-6: CDB Model Interior Object Model

A.6 Guideline: Applying Constraints to Uniformly Gridded Terrain

The following sub-sections describe the handling of point, lineals and areal constraint features into a Uniformly Gridded Terrain Elevation dataset (e.g. terrain x,y offset datasets are not available)

Note that the rendering outcome into the Elevation dataset may vary depending on the rendering order of overlapping points, lines or areals. In order to achieve deterministic outcome by all types of client-devices, client-devices are required to sort features by their layer priority number LPN before using them to constrain the terrain elevation dataset.

The rendering of a point, a lineal or areal features into the Uniformly Sampled Terrain Elevation dataset is performed into the same LOD as the LOD in which the vector feature appeared.

A.6.1 Constraint Points

This section describes the required client-device behavior for PointZ and MultiPointZ features used as terrain elevation constraint points (AHGT is true) into a uniformly sampled terrain elevation dataset.

The application of a constraint point P is very much like drawing an anti-aliased rectangle centered on P into the uniform terrain elevation grid. The rectangle shape is defined by feature attributes BBL, BBH and AO1. Consider a terrain grid element A in the immediate vaccinity of a constraint point P. After applying the constraint P to terrain grid element A, the new elevation E_A is:

$$E_A = E_P * Ain_{PA} + E_A * Aout_{PA}$$

where…

E_A is elevation of grid element A

E_P is elevation of constraint point P

Ain_{PA} is the percentage overlap of constraint point P onto grid element A

$Aout_{PA} = (1 - Ain_{PA})$

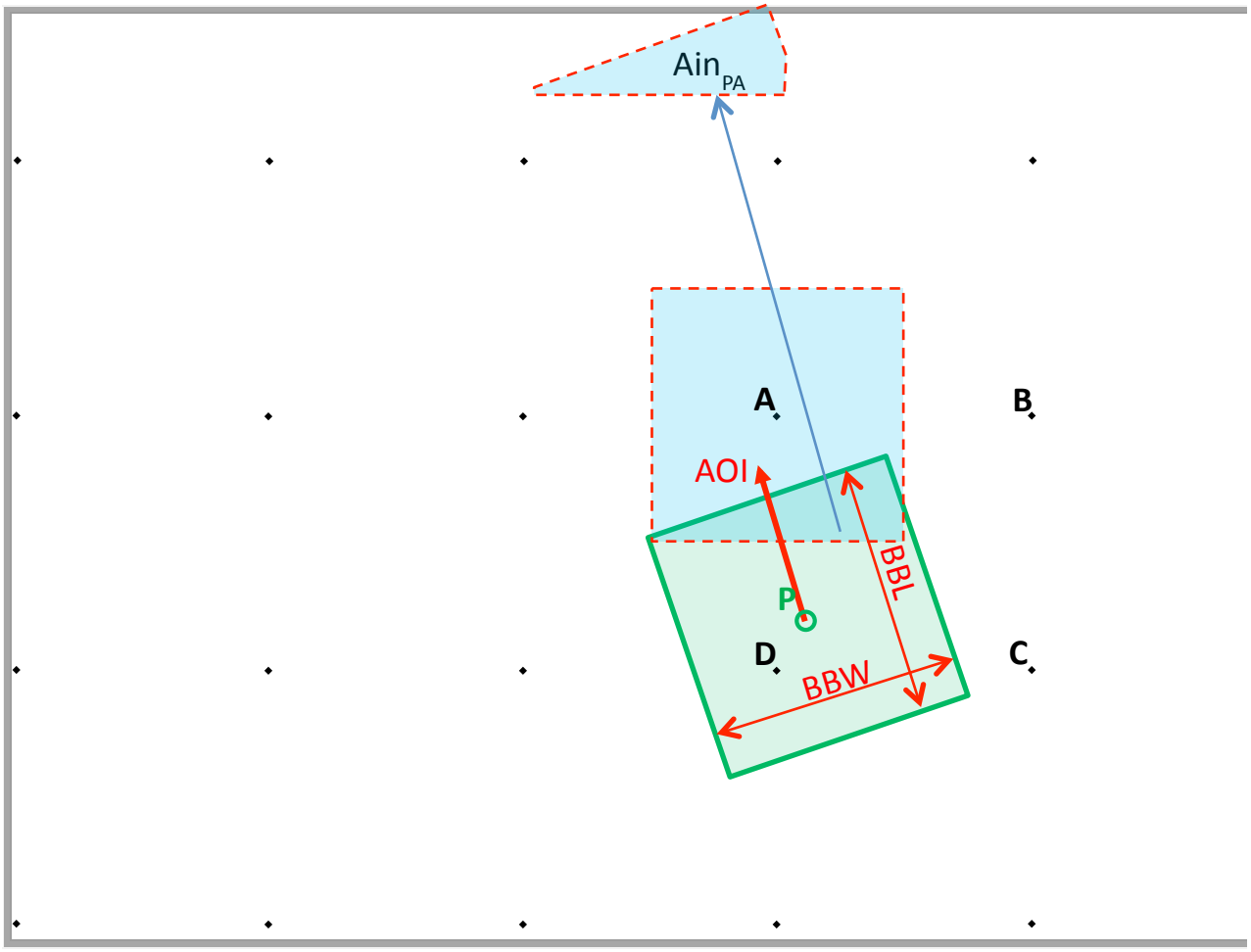


Figure A-7: Application of Constraint Point - Uniformly-Sampled Terrain

A.6.2 Constraint Lineals

This section describes the required client-device behavior for PolyLineZ features used as terrain elevation constraint lineal (AHGT is true) into a uniformly sampled terrain elevation dataset.

First, the PolyLineZ feature is broken into a series of constraint lines. The application of each constraint lineal L is very much like drawing an anti-aliased line centered on L into the uniform terrain elevation grid. The width of the line is defined by feature attribute WGP. Consider a terrain grid element A in the immediate vaccinity of a constraint lineal L, defined by vertices V1 and V2. After applying the constraint lineal L to terrain grid element A, the new elevation E_A is:

$$E_A = E_{LA} * Ain_{LA} + E_A * Aout_{LA}$$

where…

E_A is elevation of grid element A

E_{LA} is interpolated elevation of constraint lineal L at grid element A

Ain_{LA} is the percentage overlap of constraint lineal L onto grid element A

$Aout_{LA} = (1 - Ain_{LA})$

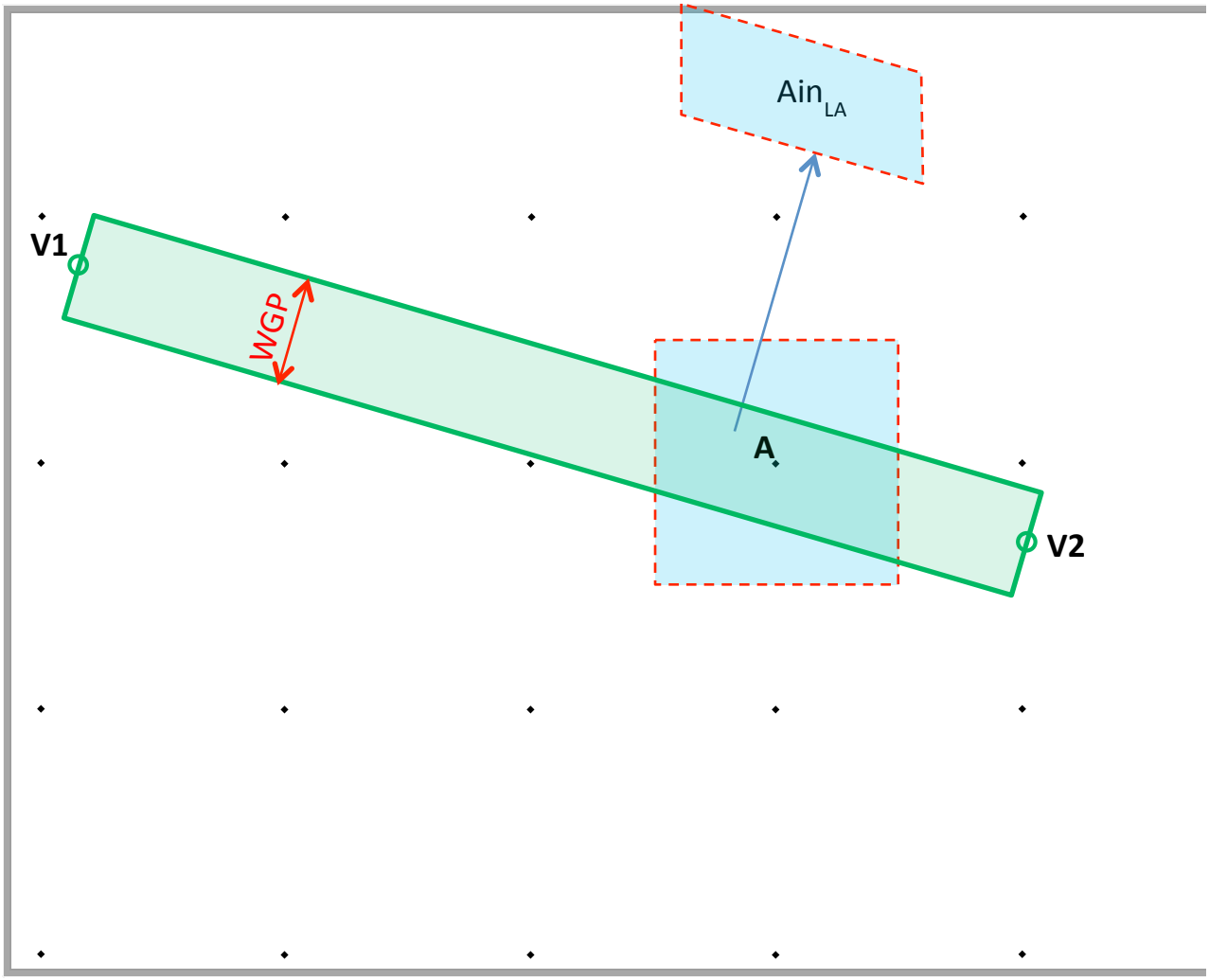
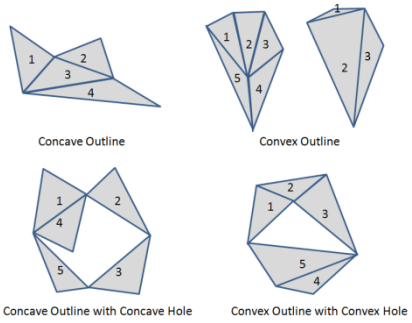


Figure A-8: Application of Constraint Line - Uniformly-Sampled Terrain

A.6.3 Constraint Areals

This section describes the required client-device behavior of PolygonZ and MultiPatch features used as terrain elevation constraint points (AHGT is true) into a uniformly sampled terrain elevation dataset.

Each ShapeFile PolygonZ feature consists of a number of rings (or parts). Each ring is a closed (the first vertex is same as the last vertex), non-self-intersecting loop. A PolygonZ feature may contain multiple outer rings. A sequence of rings can describe a convex or concave feature outline. In the CDB specification, rings can only be made up of triangles.



Each ShapeFile MultiPatch feature consists of a number of rings (or parts). Each ring is a closed (the first vertex is same as the last vertex), non-self-intersecting loop. A sequence of rings can describe a convex or concave feature outline. While the ShapeFile MultiPatch feature permits multiple inner rings (aka parts), this capability is dis-allowed in CDB. Furthermore, rings can only be made up of triangles.

The rendering of the ShapeFile feature is handled as a series of constraint triangles applied in the order in which they appear within the ShapeFile PolygonZ record. The application of each constraint triangle T is very much like drawing an anti-aliased triangle into the uniform terrain elevation grid. Consider a terrain grid element A in the immediate vaccinity of a constraint triangle T, defined by vertices V1, V2 and V3. After applying the constraint triangle T to terrain grid element A, the new elevation E_A is:

$$E_A = E_{TA} * Ain_{TA} + E_A * Aout_{TA}$$

where…

E_A is elevation of grid element A

E_{TA} is interpolated elevation of constraint triangle T at grid element A

Ain_{TA} is the percentage overlap of constraint lineal T onto grid element A

$Aout_{PA} = (1 - Ain_{TA})$

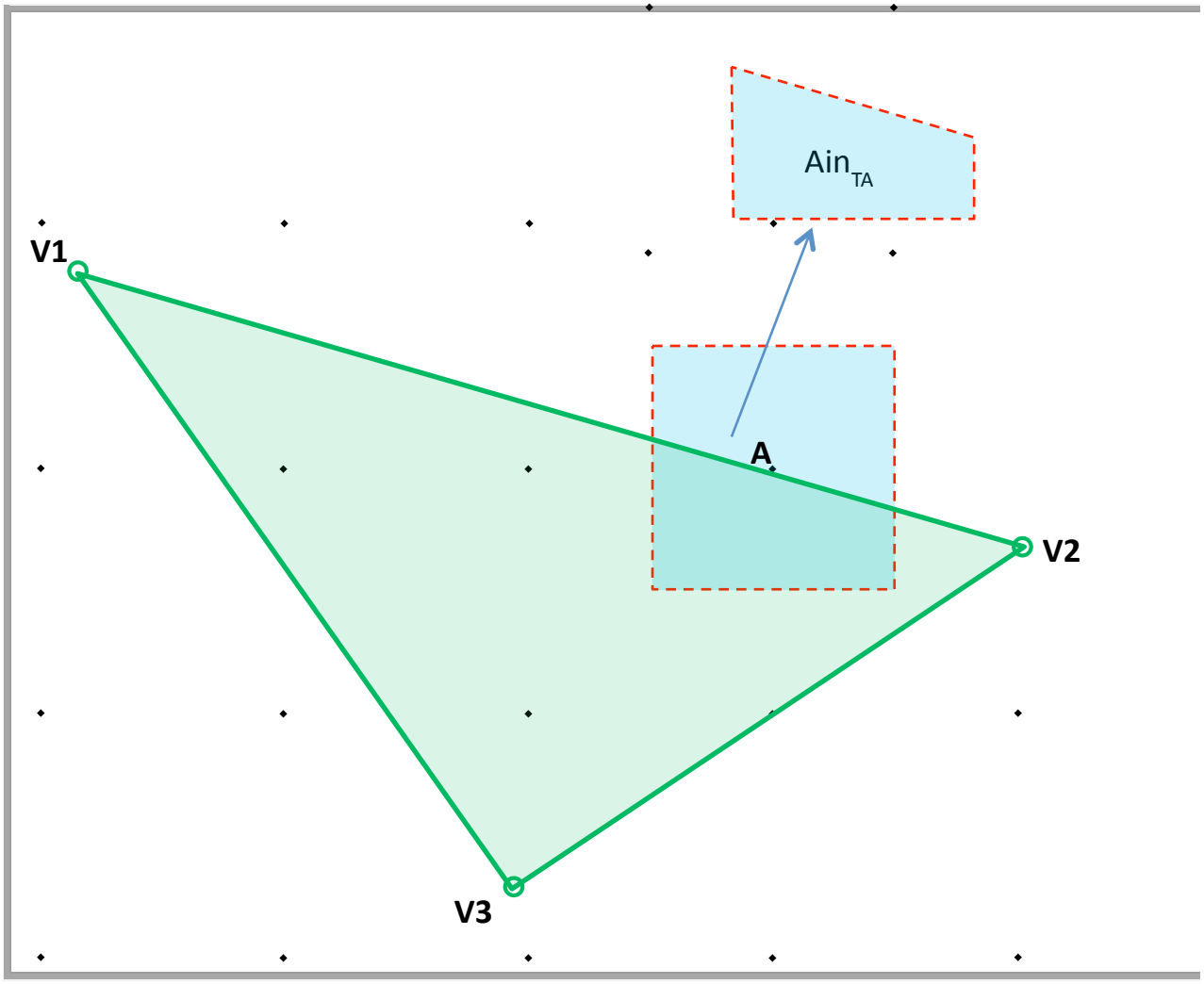


Figure A-9: Application of Constraint Triangle - Uniformly-Sampled Terrain

A.7 Guideline: Applying Constraints to Non-Uniform Gridded Terrain

The following sub-sections describe the rendering of point , lineals and areals into a Non-Uniformly Gridded Terrain Elevation dataset described in addendum "CDB Specification Addendum – Non-Uniform Sampled Terrain Elevation"

Note that the rendering outcome into the Elevation dataset may vary depending on the rendering order of overlapping points, lines or areals. The **L**ayer **P**riority **N**umber (LPN) attribute is used to achieve deterministic outcome by all types of client-devices. When ECP is supplied, client-devices are required to sort overlapping constraint points, lineals and areals in low-to-high order and then render them in that order. Value of LPN can range from 0-32767.

The rendering of a point, a lineal or areal features into the Non-uniformly Sampled Terrain Elevation dataset is performed into the same LOD as the LOD in which the vector feature appeared.

A.7.1 Constraint Points

This section describes the required client-device behavior for PointZ and MultiPointZ features used as terrain elevation constraint points (AHGT is true) into a non-uniformly sampled terrain elevation dataset.

The application of a constraint point P is applied as follows:

1. The x,y address of the affected terrain grid element is computed by truncating the lat-long coordinates of point P; note that the truncation operation varies in accordance to LOD of the terrain; however, it always yields grid element addresses in the range of 0-1023.
2. The x,y offset of the affected terrain grid element is computed by performing a MOD of the lat-long coordinates of point P in accordance to its LOD.

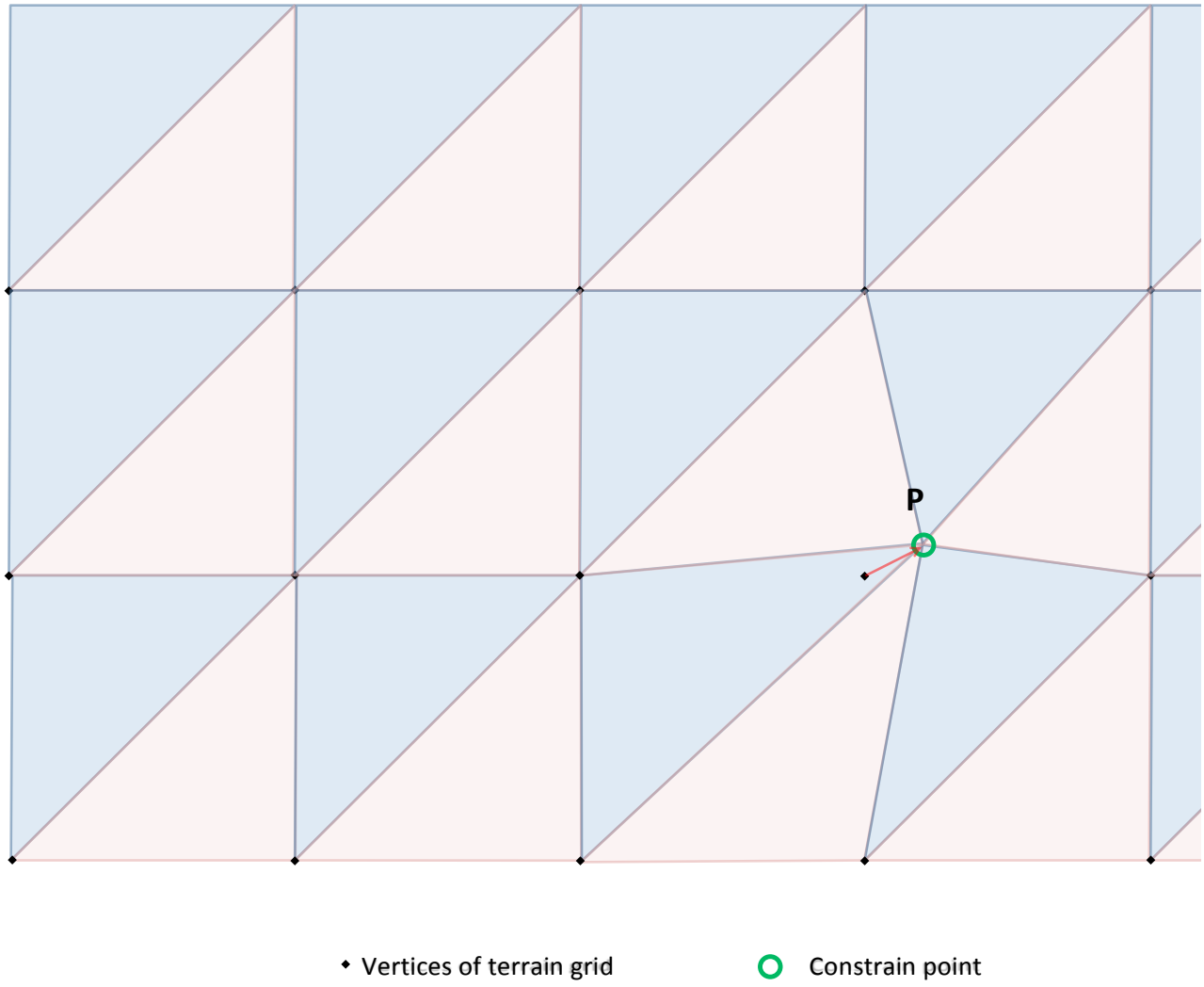
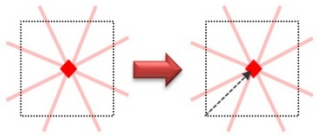


Figure A-10: Application of Constraint Point – Non-uniform Grid

A.7.2 Constraint Lineals

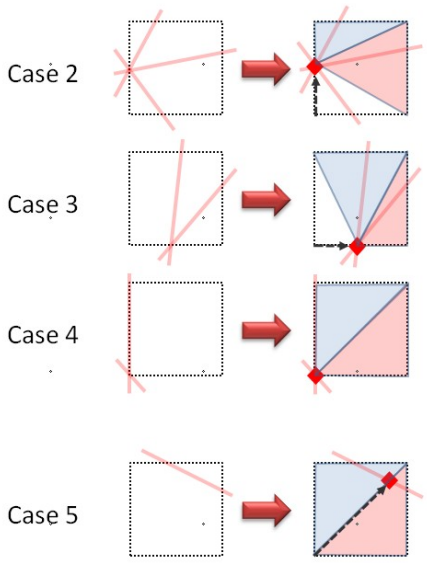
This section describes the required client-device behavior for PolyLineZ features used as terrain elevation constraint lineal (AHGT is true) into a non-uniformly sampled terrain elevation dataset.

First, the PolyLineZ feature consisting of n vertices is broken-down into (n-1) line segments defined by successive pairs of vertices.

Case 1 

The application of a constraint line segment L is applied as follows:

1. The x,y offsets of the grid elements of each vertex are computed. (see application of constraint points into non-uniformly sampled terrain (case 1).
2. The offsets of all of the other grid elements that are intersected by the line segment are handled in accordance to the illustration shown here. (case 2 to Case 5



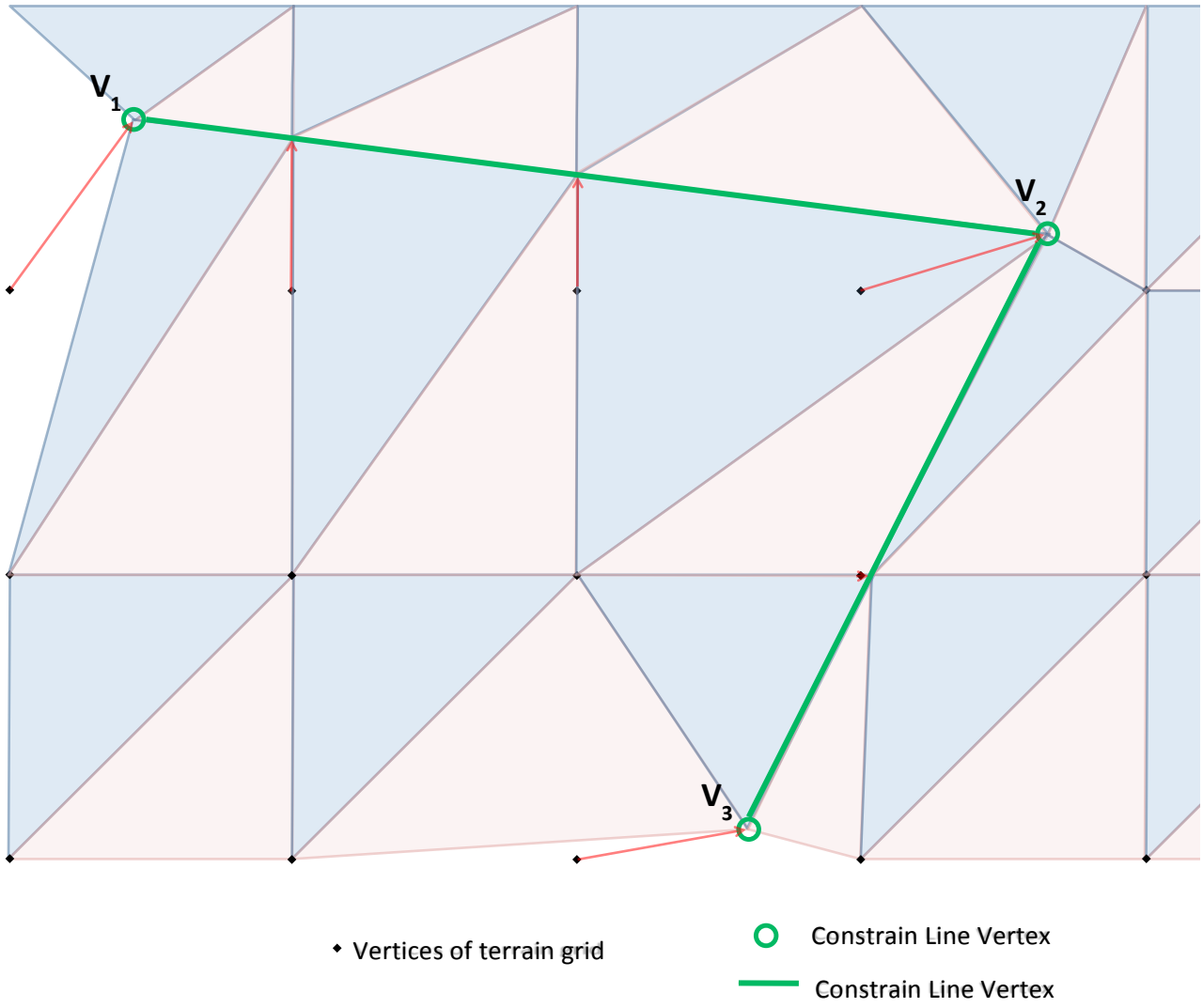


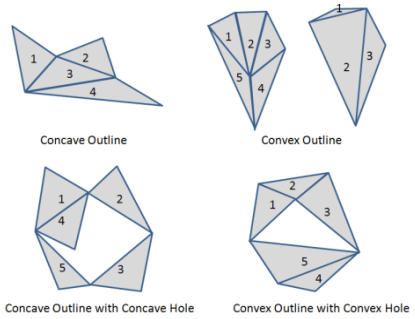
Figure A-11: Application of Constraint Lineal – Non-uniform Grid

A.7.3 Constraint areals

This section describes the required client-device behavior of PolygonZ and MultiPatch features used as terrain elevation constraint points (AHGT is true) into a non-uniformly sampled terrain elevation dataset.

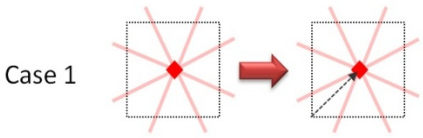
Each ShapeFile PolygonZ feature consists of a number of rings (or parts). Each ring is a closed (the first vertex is same as the last vertex), non-self-intersecting loop. A PolygonZ feature may contain multiple outer rings. A sequence of rings can describe a convex or concave feature outline. In the CDB specification, rings can only be made up of triangles.

Each ShapeFile MultiPatch feature consists of a number of rings (or parts). Each ring is a closed (the first vertex is same as the last vertex), non-self-intersecting loop. A sequence of rings can describe a convex or concave feature outline. While the ShapeFile MultiPatch feature permits multiple inner rings (aka parts), this capability is dis-allowed in CDB. Furthermore, rings can only be made up of triangles.

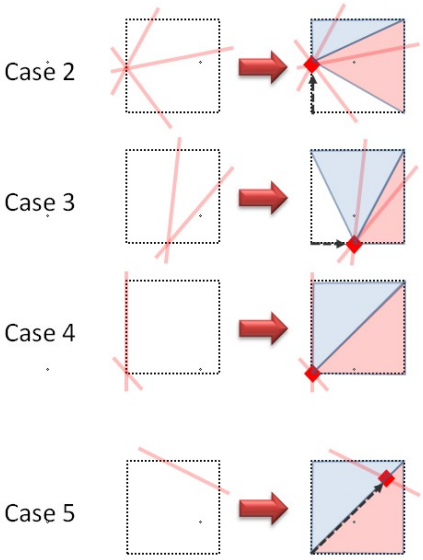


The application of a constraint triangle T is applied as follows:

1. The x,y offsets of the grid elements of each vertex are computed. (see application of constraint points into non-uniformly sampled terrain (case 1).



2. The x,y offsets of all the other grid elements that are intersected by the line segments are handled in accordance to the illustration shown here. (case 2 to Case 5)
3. The x,y offsets of all the other grid elements elevation are set to 0 and the elevation at that lat-long is interpolated using the elevation at the triangle's vertices.



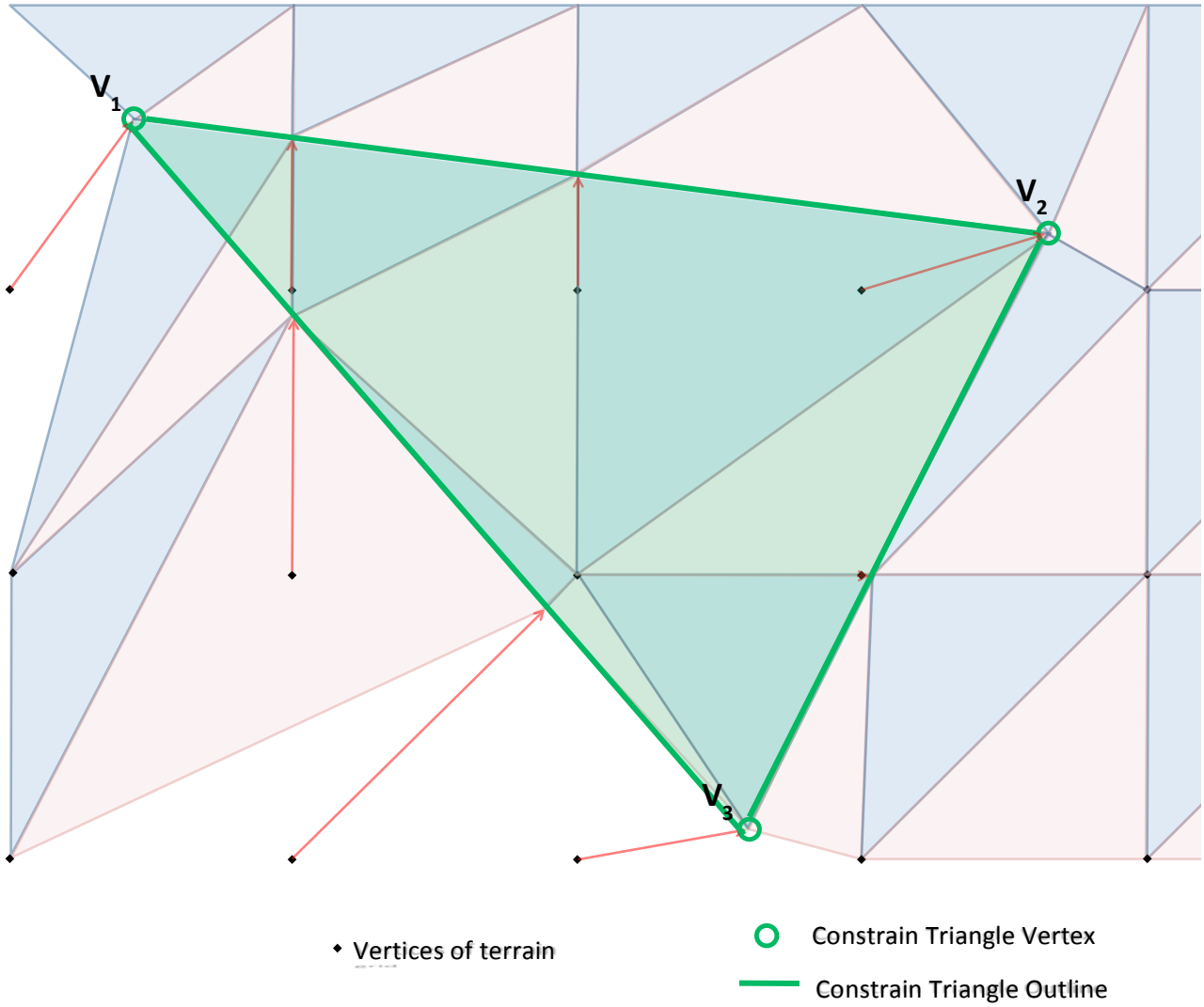


Figure A-12: Application of Constraint Areal – Non-uniform Grid

A.8 Guideline: LOD Read Behavior of Subordinate Datasets

In version 3.1 of the CDB Specification, LOD read behavior of subordinated datasets was mentioned only briefly in…

- Section 5.2.1.2.3 Subordinate Terrain Elevation Components: which stated "The CDB Specification does not permit the use of subordinate Terrain Elevation component when the primary Terrain Elevation component is not generated."
- Section 5.2.1.3.4 Default Read Value: which stated "Simulator client-devices should assume … if the data values are not available (files associated with the Subordinate Terrain Elevation component for the area covered by a tile, at a given LOD or coarser, are either missing or cannot be accessed).

- Section 5.2.1.6 Subordinate Bathymetry Component: which stated "Furthermore, since the Bathymetry values are relative to Terrain Elevation component, each value in the Bathymetry component must be matched to the finest available LOD elevation values of the Terrain Elevation component".
- Section 5.2.1.7.3 Default Read Value: which stated "Simulator client-devices should assume … if the data values are not available (files associated with the Subordinate Terrain Elevation component for the area covered by a tile, at a given LOD or coarser, are either missing or cannot be accessed).
- Section 5.2.2.3.2 Default Read Value: which stated "Simulator client-devices should assume … if the data values are not available (files associated with the Subordinate Terrain Elevation component for the area covered by a tile, at a given LOD or coarser, are either missing or cannot be accessed).

This guideline provides clarification on the client-device LOD read behavior of subordinated datasets; it describes the mandated behavior of a simulator client-device when reading a LOD of a Primary Elevation Component and combining it with another LOD of a Subordinate Terrain Elevation Component

Consider the case where a simulator client-device is attempting to read CDB data for a given region of the CDB at LOD = *p*. The CDB region has a Primary Elevation Component populated with data ranging from LOD = -10 to LOD = *m,* and a Subordinate Elevation Component populated with data ranging from LOD = -10 to LOD = *n.*

The required client-device read behavior is illustrated in Figure A-13 below, and can be summarized as follows:

- For $-10 \leq p \leq m$, the client-device accesses the primary elevation data at LOD = *p.*
- For $p > m \geq -10$, the client-device accesses the primary elevation data at LOD = *m*.
- For $-10 \leq p \leq n$, the client-device accesses the subordinate elevation data at LOD = *p.*
- For $p > n \geq -10$, the client-device accesses the primary elevation data at LOD = *n*.
- For $p > m$ and $p < n$ and $m < n$, the client-device interpolates the primary elevation data from LOD = *m* to LOD = *p* before combining it with the subordinate elevation data of LOD = *p.*
- For $p > m$ and $p > n$ and $m < n$, the client-device interpolates the primary elevation data from LOD = *m* to LOD = *n* before combining it with the subordinate elevation data of LOD = *n.*

- For $p < m$ and $p > n$ and $m > n$, the client-device interpolates the subordinate elevation data from LOD = n to LOD = p before combining it with the primary elevation data of LOD = p.
- For $p > m$ and $p > n$ and $m > n$, the client-device interpolates the subordinate elevation data from LOD = n to LOD = m before combining it with the primary elevation data of LOD = m.
- For $n = \varphi$ *(unavailable)* and $p > m \geq -10$, the client-device accesses the default value in Defaults.xml for the subordinate elevation data.
- The combination of ($m = \varphi$ *(unavailable)* and $n \geq -10$), is not permitted, i.e., the generation of Subordinate Terrain Elevation LODs is not permitted if the Primary Terrain Elevation component have not been generated.
- If the default value for the Primary Elevation dataset is unavailable in Defaults.xml, or if Defaults.xml file is missing, then the client-device must revert to the client-device's internal default value for this dataset.
- If the default value for the Subordinate Elevation dataset is unavailable in Defaults.xml, or if Defaults.xml file is missing, then the client-device must revert to the client-device's internal default value for this dataset.

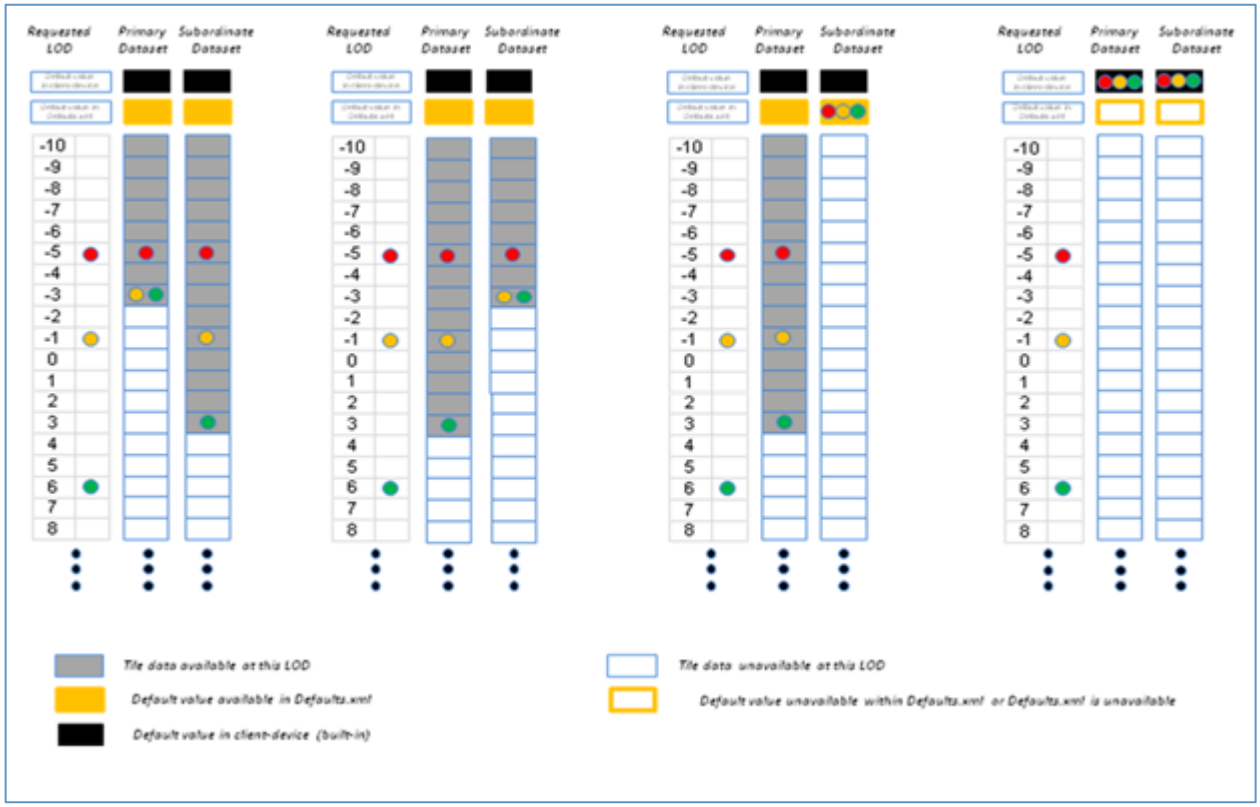


Figure A-13: Client-device Read Behavior

The default value for the Primary Terrain Elevation component is the constant Primary_Elevation, which can be found in \CDB\Metadata\Defaults.xml. The CDB Specification recommends that the value for Primary_Elevation = 0. In the case where the default value cannot be found within the Defaults.xml file, or that the Defaults.xml file cannot be found, the CDB Specification recommends that client-devices internally generate a default value of Primary_Elevation = 0.

The default values for the Subordinate Terrain Elevation layer "*n*" (where "*n*" is the subordinate elevation layer number, e.g., a value from 2 to 99) is the constant Subordinate_Elevation-*n*, which can be found in \CDB\Metadata\Defaults.xml. The CDB Specification recommends that the value for Subordinate_Elevation-*n* = 0. In the case where the default value cannot be found within the Defaults.xml file, or that the Defaults.xml file cannot be found, the CDB Specification recommends that client-devices internally generate a default value of Subordinate_Elevation-*n* = 0.

The CDB Specification does not permit the use of Subordinate Terrain Elevation components when the Primary Terrain Elevation component is not generated.

A.9 Clarification: Publisher Considerations

The CDB specification does not provide guidelines regarding its implementation within vendor SE toolsets and vendor simulation architectures. This clearly falls outside of the CDB specification mandate. Like other standards such as SIF and SEDRIS, the CDB Specification is solely a data schema for Synthetic Environmental information (i.e., it merely describes data) for use in simulation.

However, unlike SIF and SEDRIS, the CDB data schema lends itself to a real-time implementation within simulation architectures. As a result of this greater capability, many vendors have opted to achieve real-time capability with the CDB (CDB used as a runtime repository). This capability requires that the vendor's client-device be adapted with a Run-Time publishing (RTP) software function which transforms the CDB data into the client-device's internal legacy/proprietary format. This is a new concept for the simulation industry and consequently there is considerable confusion regarding the implementation of Off-line and Run-time Publishers (RTPs). While much of the attention has focused on RTPs, a similar set of considerations apply to the implementation of an off-line CDB capability (CDB is used as a Refined Source Data Repository). In this latter case, the capability requires that the vendor develop an off-line CDB import function which ingests the CDB into his Synthetic Environment Creation toolset; once imported, the toolset produces (as always) the vendor's proprietary data format through an off-line compilation function.

By definition, the function of an RTP is to bridge the "gap" (or adapt) between CDB data schema and the client-device's internal (proprietary) data schema. Since this gap is unknown, it is impossible in this addendum to provide hard-and-fast rules and detailed estimates for the implementation of an RTP (or a CDB import function).

Note that there are many alternatives open to a vendor when considering the level of compliancy he wishes to achieve. The level-of-effort is essentially a function of the level-of-compliancy the vendor wishes to achieve, and the size of the intrinsic "gap" between the CDB data schema and his device's internal schema.

Nonetheless, this section highlights aspects of the CDB that are particularly significant when considering such implementations. These aspects dominate the level-of-effort required to achieve ideal CDB compliancy.

The CDB Specification limits itself to a description of a data schema for Synthetic Environmental information (i.e. it merely describes data) for use in simulation. The CDB specification provides a rigorous definition of the semantic meaning for each dataset, each attribute and establishes the structure/organization of that data as a schema comprised of a folder hierarchy and files with internal (industry-standard) formats. This ensures that the all CDB data is understood, interpreted and efficiently accessed in the same way by each client-device. The CDB specification does not include detailed guidelines regarding off-line database compliers or run-time publisher implementations, since this would be tantamount to dictating internal vendor formats which are by their very nature proprietary.

> The CDB Specification is solely a data schema for Synthetic Environmental information (i.e. it merely describes data) for use in simulation. Given this mandate, it is entirely platform independent, i.e. it does NOT provide the implementation details of specific off-line database compilers or runtime publishers attached to specific client-devices.

As a result the CDB specification DOES NOT provide:

- The implementation details of an off-line CDB import function that can then be used to compile the imported Synthetic Environmental data into one or more types of proprietary runtime databases (only the client-device vendor has this knowledge and control)
- The implementation details or algorithms of runtime publishers attached to specific client-device (only the client-device vendor has this knowledge and control)
- The implementation details or algorithms of client-devices that use CDB data (only the client-device vendor has this knowledge and control)

While the CDB specification does not govern the actual implementation of client-devices, it is expected that the CDB specification will have a "unifying" effect on the implementation of each vendor's client-device by virtue of the fact that they will share the exact same Synthetic Environmental data. It is expected that side-by-side comparisons will be easier to undertake due to the fact that devices will run off the exact same runtime data. Prior to the advent of the CDB specification, side-by-side comparisons were considerably more difficult to undertake due to the fact the entire SE creation chain starting from raw source was implicated in such evaluations.

If we set aside legacy considerations, the simplest approach to adopting the CDB would require that client-devices ingest the CDB natively, i.e., client-devices would handle all of the CDB data schema/semantics without any off-line or run-time intermediary.

In practice however, most vendors have extensive legacy SE assets and cannot afford to obsolesce these. As a result, most client-devices must continue to support their own proprietary legacy runtime databases. Given these considerations, two solutions are possible:

a) No change to the Client-device: In this approach, vendors have chosen to achieve an off-line CDB capability (CDB is used as a Refined Source Data Repository). This capability requires that the vendor develops an off-line CDB import function which ingests the CDB into his Synthetic Environment Creation toolset; once imported, the toolset produces (as always) the vendor's proprietary data format through an off-line compilation function.

b) Insertion of a Runtime Publisher: In this approach, vendors have opted to achieve real-time capability with the CDB (CDB is used as a runtime repository). This capability requires that the vendor's client-device be adapted with a Run-Time publishing (RTP) software function which transforms the CDB data into the client-device's internal legacy/proprietary format.

A.9.1 Level-of-Effort of a Publisher Implementation

The following discussion attempts to qualify the level-of-effort to achieve CDB compliancy. The discussion applies equally to both paradigms, i.e., the CDB Runtime Publishing paradigm and the CDB import-then-compile paradigm.

In the case where a client-device already supports most of data schema and semantics concepts of the CDB, then the RTP (or import-then-compile) software is proportionally less complex. For instance, if an IG already supports the concepts of tiles, of levels-of-detail, of layers and understands the concepts of datasets such as terrain texture, gridded terrain elevation, gridded terrain materials, etc. then there is a modest amount of work to be performed by an RTP.

> The level-of-effort in adopting the CDB data schema is proportion to the difference between the CDB data schema and client-device's internal proprietary data schema.

Clearly, the algorithmic complexity of an RTP and the computational load imposed on the RTP is directly proportional to the above-mentioned "gap". The larger the "gap", the more expensive a RTP is to develop and the more computational resources need to be allocated to implement it. Conversely, with a smaller "gap", the RTP development is straightforward and relatively few computational resources need to be allocated to this function.

In order to assess the level-of-effort to adopt the CDB schema, the vendor must first evaluate the similarity of data schemas between the CDB and his client-device, in particular, he must assess whether he espouses the following fundamental CDB concepts:

- Independent Tiles (used for paging of all SE data)
- Independent Levels-of-Detail (for all SE data)
- Independent Layers (Dataset Layering)
- Following dataset concepts and semantics:
  - Semantic understanding of all CDB layers
  - Geo-gridded data layers consisting of terrain altimetry, terrain texture, terrain materials/mixtures
  - Geo-defined vector features (points, lineals, areals)
    - With/without modeled 3D representations
    - Feature attribution
  - 3D Modeled representation of features (using a data schema similar to or equivalent to OpenFlight)
    - Instanced geotypical models
    - Instanced model geometry
    - Instanced model texture
    - Non-instanced geospecific models
  - Conforming of features to terrain skin (e.g. height conforming)
  - Topological networks
  - JPEG-2K compression
  - Generation of device-specific NVG/FLIR rendering parameters for light-points and materials

In the case where a client-device does not intrinsically support one or more of the above-mentioned CDB concepts, the RTP must perform SE conversions that will likely fall beyond those of mere format/structure manipulations. Such conversions may affect the precision of the data, its semantic meaning, etc, and thus can compromise certain aspects of runtime correlation.

The CDB data schema favors modern up-to-date implementations of client-devices. In effect, the level-of-effort to develop an RTP for an obsolete legacy device is likely to be greater than for a modern device. This is because early approaches in digital computer based flight simulation were more severely constrained by severe hardware, software and data source limitations. Consequently, simulation engineers made important compromises between a subsystem's targeted fidelity and its level of generality, scalability, abstraction, and correlation with other simulator client-devices. In many cases, engineers reverted to complex support data structures (generated off-line) in order to reduce the computational load at runtime.

A classic example of this was the use of Binary Separation Planes (BSPs) data structures[2] which were required prior to the widespread adoption of Z-buffers by the IG vendors. The CDB specification does not make provisions for this and as such, the RTP for legacy BSP-based IG devices would be burdened with the rather difficult task to generate BSPs in real-time.

Given their tremendous benefit, the concepts of database paging (e.g. tiles) and levels-of-details have steadily been adopted by simulation vendors over the past 15-20 years and have been applied to most datasets, notably terrain and imagery datasets. (See Appendix "A" of the CDB Specification for a rationale for Tiles and Levels-of-detail). As a result, it is not expected that the CDB tiles and LOD concepts will be a problem for most vendors. Note however that CDB applies these two concepts to ALL dataset layers including vector features and 3D models.

A.9.2 Typical Functions Performed by a Publisher Implementation

The following discussion provides a typical list of software functions that must be developed in order to achieve CDB compliancy. The discussion applies equally to both paradigms, i.e. the CDB Runtime Publishing paradigm and the CDB import-then-compile paradigm.

Virtually all simulation client-devices in existence today natively ingest their own proprietary native runtime formats. In order to ingest CDB directly, vendors must adapt the device's software to natively ingest the CDB format (e.g. TIFF, Shape, OpenFlight, etc.) or alternately, he can insert a runtime publisher function that transforms the CDB data formats into legacy client device's native runtime format. The runtime publishing process is performed when the CDB is paged-in from the CDB storage device.

[2] Such BSP data structures where required by most IG vendors prior to ~1995 due to the fact that the IGs did not have sub-pixel level Z-buffer capability.

The runtime publishers are nothing more than well-optimized offline database publishers capable of responding to the on-demand compilation of datasets as they are being paged-in by the respective client devices. The function of a runtime publisher is no different than that of a conventional offline database publisher, i.e., it…

a) transforms the assembled database so that it satisfies the client-device's internal data structure and format

b) transforms the assembled database so that it satisfies the client-device's internal naming conventions

c) transforms the assembled database so that it satisfies the client-device's number precision and number representation

d) transforms the assembled database into parameters compatible with the client device's internal algorithms (typically light parameters, FLIR/NVG parameters, etc.

e) transforms the assembled database so that it satisfies the client-device's data fidelity requirements

f) transforms the assembled database so that it satisfies the client-device's performance and internal memory limitations

g) transforms the assembled database so that it satisfies the client-device's level of-detail representation requirements.

Ideally, the scope of an RTP should be purely limited to manipulations of data format and data structure and internal naming conventions (items 1-7 above). Under such circumstances, it is possible to achieve perfect runtime correlation between client-devices.

A.9.3 Publisher Implementation Recommendations

The use of the CDB data schema "as-is" by a client-device achieves all of the benefits stated in sections 1.4 and 1.5 of the CDB specification, namely:

a) Improved SE generation timeline and deployment

b) Interoperable simulation-ready SE

c) Improved client-device robustness/determinism

d) Increase SE longevity

e) Reduced SE storage infrastructure cost

f) Platform independence and scalability

g) SE scalability and adaptability

In the case where a client-device does not adhere to one or more of the above-mentioned "fundamental CDB concepts", fewer of the CDB benefits will be realizable.

For instance, a client-device incapable of dealing with levels-of-detail will not have the same level SE scalability (a benefit explained in section 1.4.7 of the CDB specification) as one that fully espouses that concept. While the latter may be acceptable, it is clearly a less-compliant and an inferior implementation of the CDB than the former.

Changes to the modeled representation of features are generally not advisable since it invariably affects the accuracy of the modeled representation. Most image generators in use today can ingest a (one-for-one correspondence) the CDB modeled polygonal representation of 3D features. However, in the case of terrain, there are two dominant approaches in industry, either a regular grid with LODs or alternately, the Terrain Irregular Network (TIN) mesh. The CDB specification has opted for the former given its greater scalability, determinism and compatibility with tiling schemes. Clearly, implementations where such conversions are not necessary are advantaged and provide more of the above-mentioned CDB benefits.

Furthermore, the CDB is designed to provide both the semantic (e.g. vector data/attribution) and the modeled representation of features. Since the CDB specification provides both, it is not advisable to ignore or replace the modeled representation (if provided) nor is it advisable to synthesize a non-CDB modeled representation if none was supplied within the CDB. While the CDB specification does not forbid vendors to interpret CDB feature data for the purpose of procedurally synthesizing more detailed feature data or synthesizing modeled data from the feature data, *this practice is not recommended as this would severely compromise correlation and inter-operability*. In the context of correlated synthetic environments, such approaches are viable if and only if all client-devices in a federation are equipped with the exact same procedural algorithms. Currently, this is not possible because there are no industry-standard, open-source procedural algorithms endorsed by all simulation vendors.

> In the case of the CDB Runtime Publishing paradigm and the CDB import-then-compile paradigm, it is not advisable to ignore or replace the modeled representation (if provided) nor is it advisable to synthesize a non-CDB modeled representation if none was supplied within the CDB.

A.10 Rationale: Sensor Simulation - Achieving Device-Independence

One of the primary objectives of this Specification is to provide and integrate all of the data required by all sensor devices, not just Image Generators producing the OTW scenes. The purpose of this integration, among other things, is to achieve and maintain a high level of correlation among the many client-devices (subsystems) within a simulator. Furthermore, this integration must be done independently of the client-device or the sensor type, with little or no duplication of data amongst clients. In addition to the OTW, many simulator client-devices are required to simulate the synthetic environment over different portions of the electromagnetic spectrum, infrared (e.g. FLIR, NVG), microwaves (e.g. radar), audio (e.g. sonar), etc. Up to now, the current state of the art approaches to the simulation of sensors has typically been quite

proprietary to the client-device implementation of the various vendors. There have been no universally accepted simulation models suitable for use in simulation.

Sensor simulation typically requires a simulation of the device itself, supplemented by a complete simulation of the synthetic environment over the portion of the electromagnetic spectrum that is relevant to this device. The former simulation is referred to as the **S**ensor **S**imulation **M**odel (SSM) while the latter is called the **S**ensor **E**nvironmental **M**odel (SEM). In the past, the SEM relied heavily on environmental database whose content was designed to match the functionality, fidelity, structure and format requirements of the SEM. The level of realism possible by the SEM depended **heavily** on the quality, quantity and completeness of the data available. The environmental database was highly device-specific and could not be readily ported to other platforms.

A SEM is usually based on mathematical model of the environment for the portion of the electromagnetic spectrum of interest. The SEM acts much as a black box that produces a response in accordance to input data. A significant portion of this data must come from the CDB; however, the key is to segregate all device-dependent data and all SEM-dependent data from the modeling data that represents the synthetic environment. In order to accommodate the most different kind of sensors possible, a common denominator must be chosen. In the CDB specification, this common denominator is called a material. This is the subject of this chapter.

One of the fundamental issues of sensor simulation involves the handling of material properties. As discussed earlier, the determination of which material properties should be supported heavily depends on:

a) the sensor types to be supported.

b) the vendors' sensor simulation implementations to be supported.

c) the level of fidelity, functionality and precision of the SEMs to be supported.

Clearly, the task of determining a definitive list of material properties that would accommodate all of the above requirements for the today's sensor types, vendor implementations and SEMs would be a significant challenge. Furthermore, once released, the materials properties would limit any SEM innovation by the industry. As a result, the CDB specification limits its jurisdiction over the material properties.

Instead, the CDB specification defines and publicly defines a list of materials that can be used in a CDB. It is the responsibility of each vendor to define the properties (that satisfies the sensor type) for these CDB materials. As a result, vendors are totally free to select material properties that satisfy the fidelity, functionality and precision requirements of the SEM for the sensor type of interest. Alternately, if the vendors have their own list of materials, they can create a mapping between CDB materials and their internally supported list of materials. This approach allows client-devices to retain their SEMs as well as their own sets of material properties.

The next section describes how materials are defined and modeled within a CDB.

Appendix L enumerates the base materials supported by this Specification.

A.11 Rationale: Partitioning the Earth into Tiles

This section provides rationale for partitioning the world into tiles.

The design of the CDB specification tile representation is centered on three primary considerations:

(1) A tile representation comprehensive enough to accommodate the entire earth.

(2) A tile representation that lends itself to real-time implementation by a CDB system and all of its attached simulator client-devices.

A numerically straightforward mapping (such as a simple scaling) to map lat-long coordinates into CDB coordinates and vice versa is highly desirable for real-time implementation considerations.

(3) A tile representation with a system of units that conforms as much as possible to geographic standards.
One of the underlying motivations driving the CDB tile representation is the need for a system that will remain as close to the raw source data as possible which currently is DTED and GeoTIFF; DTED uses a geographic coordinate system defined by latitudes and longitudes. The basic unit in DTED is a geo-cell, which always has a height and width of one degree. In order to maintain a density of data that does not increase inordinately when moving towards the poles, the grid post intervals (measured in degrees or arc-sec) along the longitudinal axis are increased at specific latitudes; for instance, at DTED level 2, the latitude interval is always one second of arc but the longitude interval is one second of arc at latitudes from 0 to 50 degrees, from latitudes 50 to 70 the interval is two arc seconds and so on as shown in Table A-3. INTERVALS FOR DTED LEVEL 2.

Table A-3. INTERVALS FOR DTED LEVEL 2

DTED Zone	Latitude Range (Degrees)	Latitude Interval (Arc seconds)	Longitude Interval (Arc seconds)
I	0 – 50 N-S	1	1
II	50 – 70 N-S	1	2
III	70 – 75 N-S	1	3
IV	75 – 80 N-S	1	4

V	80 – 90 N-S	1	6

Before going into the detailed design of the CDB tile representation, it is worth stating the guiding principles that constrain the approach used by the CDB tile representation:

(1) The earth model is divided (in latitude) into slices.

(2) The slice's x-axis is aligned to WGS-84 lines of latitude.

(3) The slice's y-axis is aligned to WGS-84 lines of longitude.

(4) The number of units along the slice's y-axis for a given level of detail is the same for all slices.

The earth surface geodetic dimension in arc-second of y-axis units within an earth slice and in all earth slices is exactly the same, regardless of latitude.

(5) The geodetic dimension of an x-axis unit in arc-second is constant within a zone, but is re-defined at pre-selected latitudes to achieve a greater level of spatial sampling uniformity in all tiles; this overcomes the narrowing effect of increased latitudes on longitudinal distances. The definition of zones in the CDB is the same as those in DTED (with the exception of the poles).

(6) The number of units along the slice's x-axis for a given level of detail is the same within each zone.

(7) The number of units along the slice's y-axis is constrained to a 2^n-multiple in all slices.

Many simulator client devices impose constraints related to the run-time use of binary pyramidal structures (such as mip-maps, quadtrees, etc.). A binary pyramidal structure is simply a collection of two-dimensional arrays; each array represents the same content but at successively finer levels of resolution.

(8) The number of units along the slice's x-axis will vary depending on which zone the latitude of the slice belongs. At this point we introduce the concept of a CDBGeocell, which differs slightly from a DTED Geocell. A DTED cell is always 1 × 1 degrees. In contrast, a CDBGeocell always has a height of 1 degree but has a varying width depending on its latitude. Table A-4. Size of CDB Geocell per zone shows the dimensions of a CDBGeocell per zones of latitude. For instance, in latitude zone 5, which goes from –50 to 50 degrees latitude, a CDBGeocell is 1 × 1 degree, in zone 4 and 6 which goes from latitude 50 to 70 degrees the cell size is 1 × 2 degrees. The main reason to introduce this concept is to maintain a reasonable eccentricity between the sides by trying to keep them as close to a square as possible. Two criteria are used to define the size of a CDB Geocell:

(a) A CDBGeocell must contain a whole number of DTED Geocells; in other words a CDBGeocell must start and end on a whole degree along the longitudinal axis. This is done so as to facilitate mapping from CDBGeocells to DTED Geocells.

(b) The length of the CDBGeocell must be a whole factor of 180, in other words length of 1, 2, 3, 4, 6 and 12 degrees are legal but lengths of 7 and 8 degrees would not be since these are not exact factors of 180.

Table A-4. Size of CDB Geocell per zone

CDB Zone	Latitude Range (Degrees)	CDBGeocell size (deg Lat × deg Lon))	Number of DTED Geocells
0	–90 □ lat < –89	1 X 12	12
1	–89 □ lat < –80	1 X 6	6
2	–80 □ lat < –75	1 X 4	4
3	–75 □ lat < –70	1 X 3	3
4	–70 □ lat < –50	1 X 2	2
5	–50 □ lat < +50	1 X 1	1
6	+50 □ lat < +70	1 x 2	2
7	+70 □ lat < +75	1 x 3	3
8	+75 □ lat < +80	1 x 4	4
9	+80 □ lat < +89	1 x 6	6
10	+89 □ lat < +90	1 x 12	12

The variable CDBGeocell size in the CDB specification has the following benefit:

(a) Reduces the simulator client processing overheads associated with the switching from one zone to another. (Due to the small number of zones across the earth.)

(b) Reduces the variation of longitudinal dimensions (in meters) to a maximum of 50%.

(c) Improves storage efficiency.

A.12 Rationale: Importance of Level-of-Details

The availability of LODs for most datasets is critical for real-time performance. Many simulator client-devices can readily take advantage of an LOD structure because many clients naturally require less detail with increasing distance away from the simulated ownship position. For example, the projection of screen pixels (i.e. pixels in an IG image plane) onto near-field terrain subtends much less area than the projection of screen pixel onto far-field terrain near the horizon; as a result, much less detail is required at far range. In addition, clients may need to revert to an alternate coarser representation if they cannot cope with the paging bandwidths, memory footprint or computational requirements of finer LODs. This provides a solid basis on which client-devices can build paging managers, load management and memory management algorithms.

The following example illustrates the important performance considerations and the inherent performance advantage that can be achieved with an LOD structure. Consider a simulator client-device, with a capability to display terrain imagery out to 128 km; the imagery is 1m at its finest available resolution and the simulated ownship is flying at 100 m/s. Under these conditions, and without the benefit of an LOD organization (as illustrated in Figure A-15: Paging of Terrain Imagery without an LOD Structure), the client-device would require access to the imagery at a rate of ~100 Mpixels/sec. Consider on the other hand the same operating conditions but with the client-device accessing LOD-organized imagery (as illustrated in Figure A-14: Paging of Terrain Imagery with an LOD Structure). Furthermore, assume that the client-device only requires 1m imagery for ranges less than 1/2 km, 2m for ranges less than 1km, 4m for ranges less than 2km, and so on. With the benefit of an LOD structure, the client-device would require access to the imagery at a much lower rate of ~1 Mpixels/sec, reducing access bandwidth by a factor of ~100x over the non-LOD approach. Clearly, such performance gains cannot be ignored for real-time applications such as flight simulators, especially when one realizes that access bandwidth increases as the square of the imagery resolution.

In addition to a reduction in access bandwidth, the LOD structure also benefits simulator client-devices that have a requirement to dynamically filter the data to control aliasing. In effect, part of the client-device filtering process is relegated to an off-line process.

The CDB specification does not enforce, nor does it specify the type of filter used to compute the data element values of raster-organized or list-organized datasets. Yet, it is clear that inadequate

off-line filter may affect the rendering quality of the affected client-devices. As a result, the CDB specification provides guidelines to govern the quality of the off-line LOD process; these guidelines are provided with each of the raster-organized dataset (or list-organized datasets in future releases of the CDB specification).

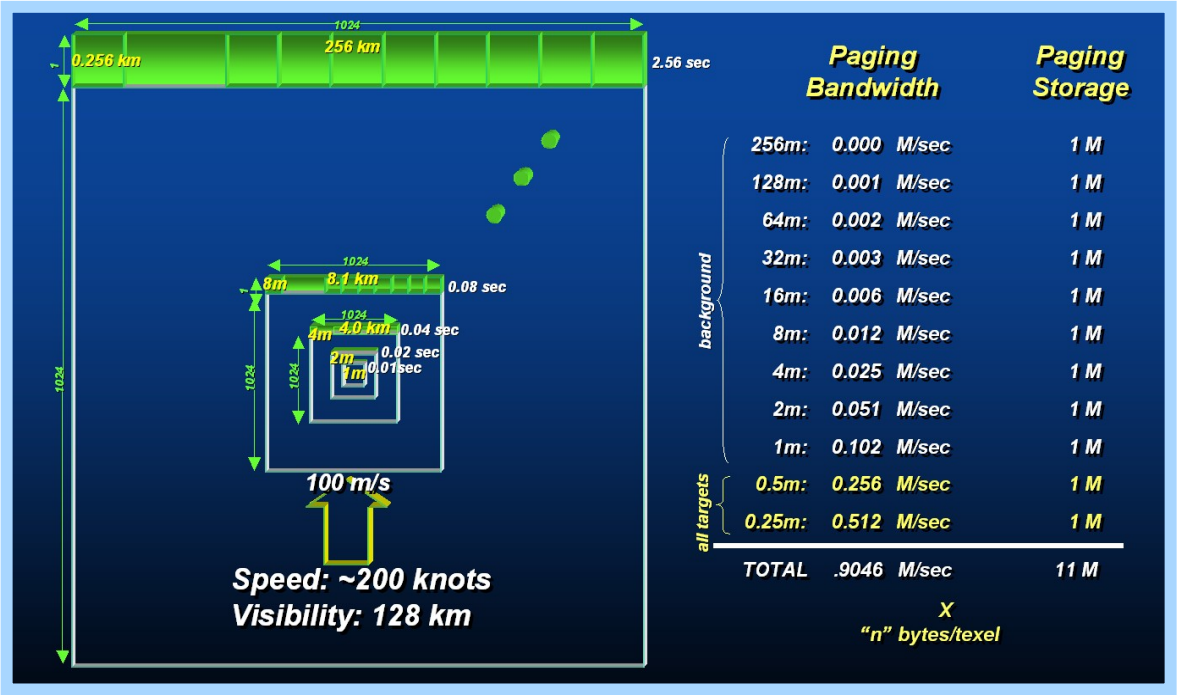


Figure A-14: Paging of Terrain Imagery with an LOD Structure

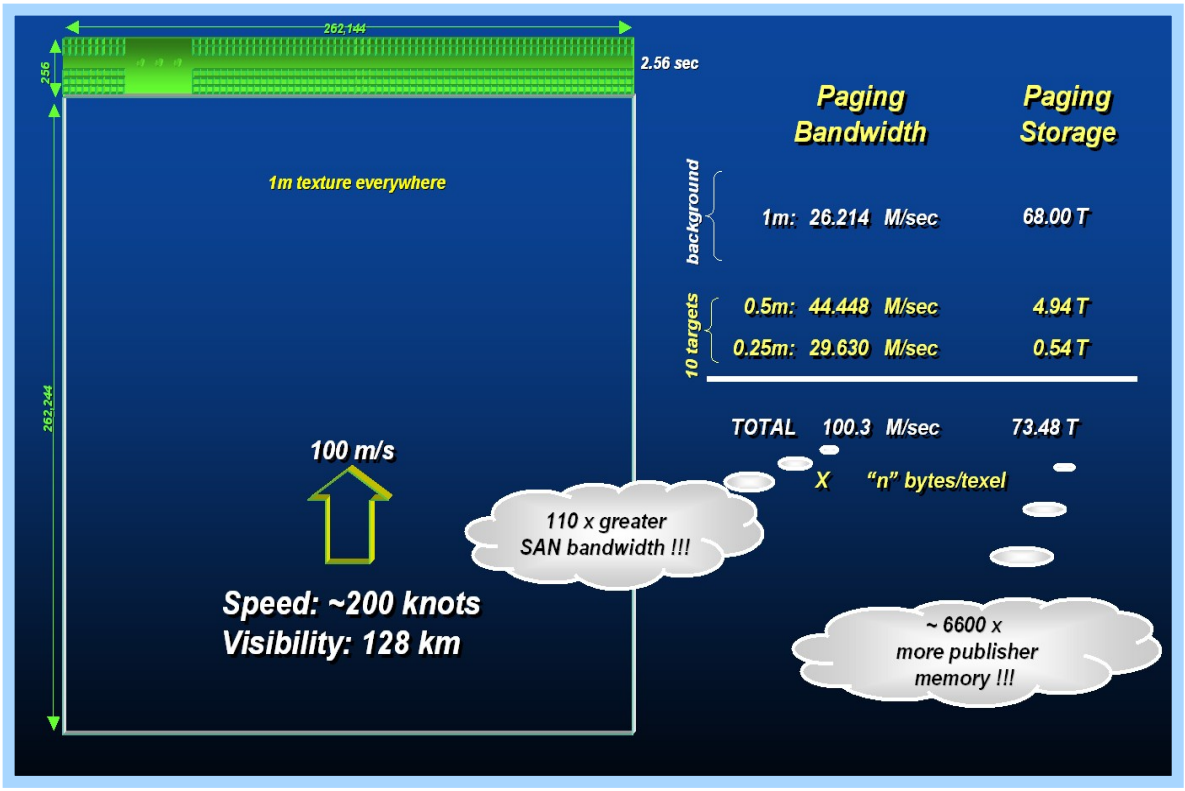


Figure A-15: Paging of Terrain Imagery without an LOD Structure

A.13 Primer: Line-of-Sight Algorithms Using MinElevation and MaxElevation Components

The purpose of the MinElevation and MaxElevation components is to provide the CDB with the necessary data and structure to achieve the required level of determinism in the computation line-of-sight calculations with the terrain. The values of each component are with respect to mean sea level. Since both the MinElevation and the MaxElevation values are provided by this Specification, any line-of-sight algorithm can rapidly assess an intersection status of the line-of-sight vector with the terrain.

There are three cases to consider:

1. **CASE 1 – No intersection:** If all of the LOS Bounding Boxes are above the MinMax Bounding Boxes, then there is no intersection between the line-of-sight vector and the terrain. No further testing is required. (Refer to Figure A-16: Case 1 – No Intersection.)

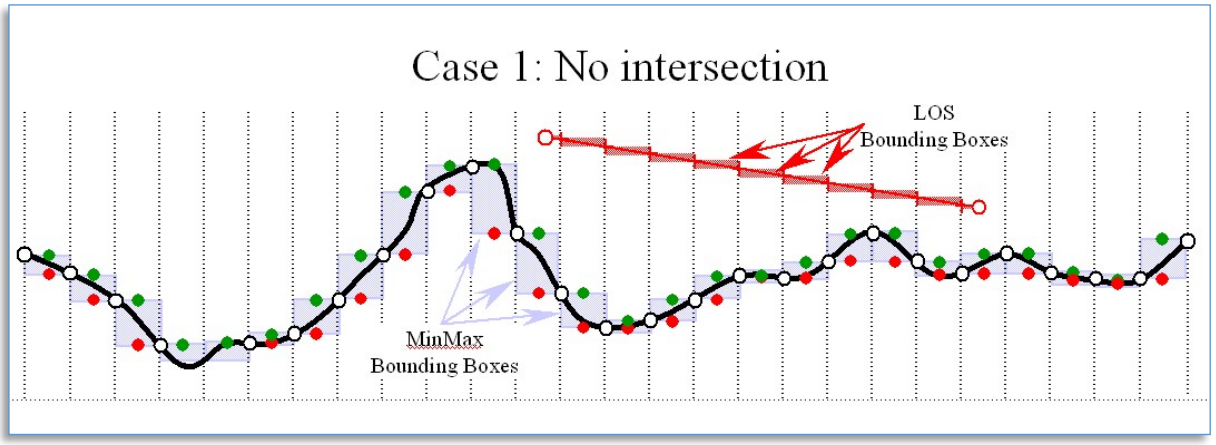


Figure A-16: Case 1 – No Intersection

2. **CASE 2 – Potential intersection:** If one or more of the LOS Bounding Boxes overlap with a MinMax Bounding Box, then there is a potential intersection between the line-of-sight vector and the terrain. This step must be repeated with progressively finer level-of-detail versions of the MinElevation and MaxElevation values until Case 1 or Case 3 is encountered. If the finest level-of-detail is reached and the LOS result still yields a potential intersection status (Case 2), then the LOS algorithm must perform a LOS intersection with the finest LOD of the Primary Terrain Elevation component using the prescribed CDB meshing convention. (Refer to Figure A-17: Case 2 – Potential Intersection.)

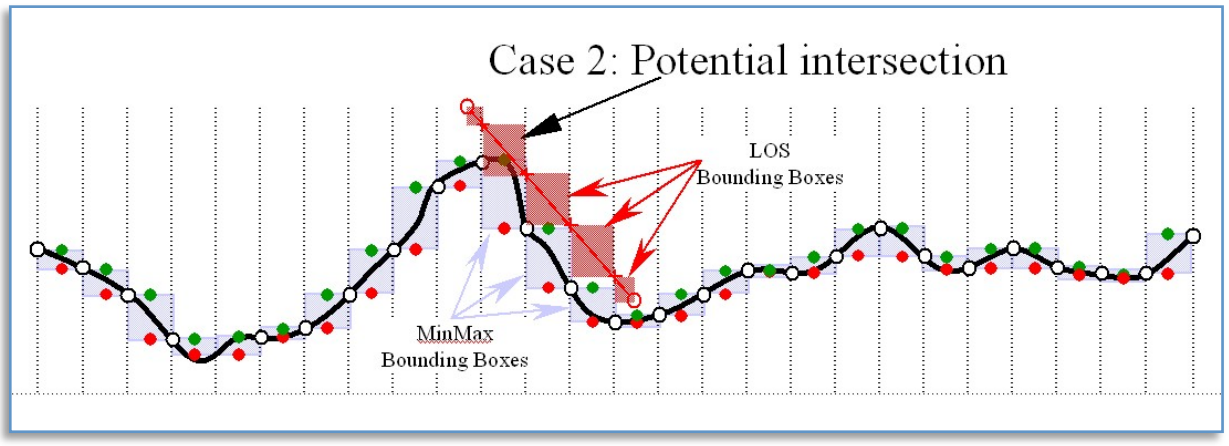


Figure A-17: Case 2 – Potential Intersection

3. **CASE 3 – Intersection:** If one or more of the LOS Bounding Boxes are below the MinMax Bounding Boxes, then there is an intersection between the line-of-sight vector and the terrain. No further testing is required to determine whether there is intersection or not. (Refer to Figure A-18: Case 3 – Guaranteed Intersection.) However, to determine the intersection point, the LOS algorithm must perform the following additional steps. If (starting with the LOS point-of-origin) one or more of the LOS Bounding Boxes overlap with a MinMax Bounding Boxes, then there is a potential intersection between the line-of-sight vector and the terrain for that MinMax Bounding Box. This step must be repeated with progressively finer level-of-detail versions of the MinElevation and MaxElevation values until Case 1 or Case 3 is encountered. If the finest level-of-detail is reached and the LOS result still yields a potential intersection status (Case 2), then the LOS algorithm must perform a LOS intersection with the finest LOD of the Primary Terrain Elevation component using the prescribed CDB meshing convention.

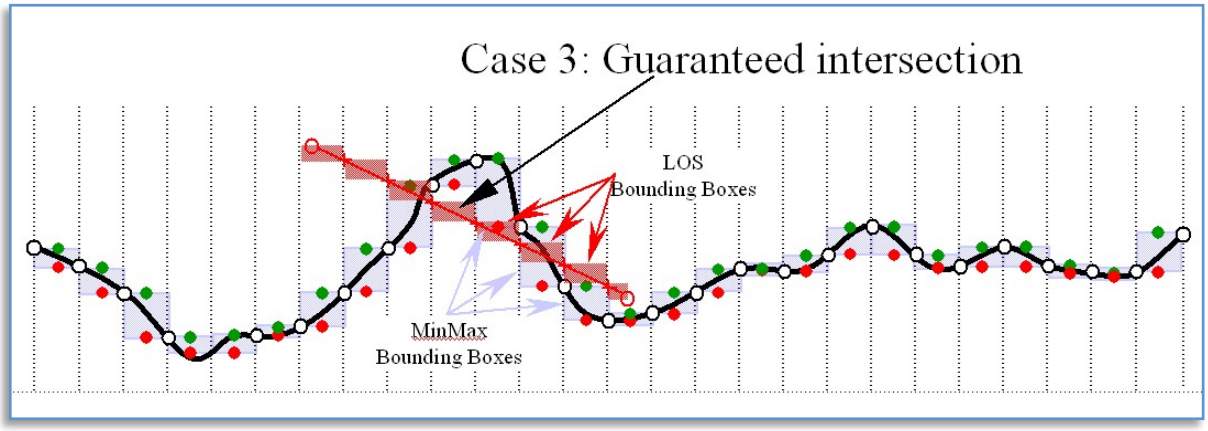


Figure A-18: Case 3 – Guaranteed Intersection

A.14 Primer: Radar Cross Sections (RCS)

This section provides technical description of the RCS data and its usage in real-time simulation.

The Radar Cross-Section (RCS) of a target is a measure of the radar reflection characteristics of a target (usually expressed in m^2, dBsm, or volts). It is equal to the power reflected back to the radar divided by power density of the wave striking the target. For most targets, the radar cross-section corresponds to the area of the cross section of the sphere that would reflect the same energy back to the radar, if a metal sphere were substituted. A sphere is sometimes used since the RCS of a sphere is independent of frequency if operating in the far field region of the radar. (Reference [24])

The RCS data unit of measure for the intensity are usually referenced as a normalized ratio in Decibels relative to a square meter (reference [25]), or otherwise known as dBsm. Another data measure that is linked to the intensity measure is also the 'phase shift' angle (in degrees) of the returned energy. It can provide some additional information about the reflective attributes of the elements reflecting back to the radar.

However, the RCS defines the echo at the radar for the model (target) in question, which varies considerably depending on the target's orientation, its relative distance and size with respect to the simulated radar's antenna. The viewing angles are shown in the diagram below.

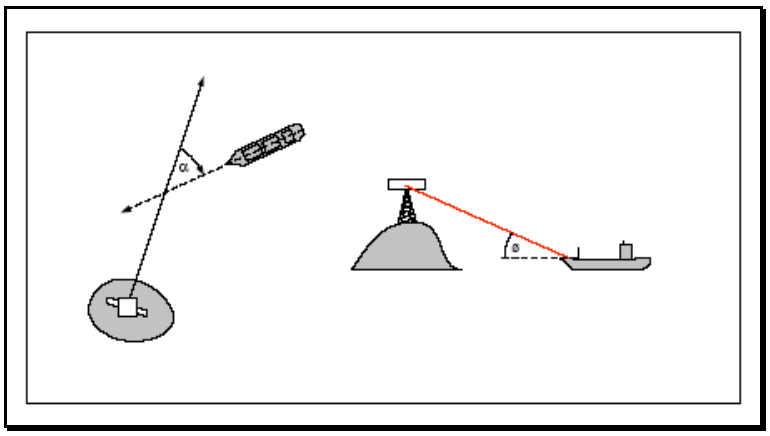


Figure A-19: Relative Azimuth (α) and Elevation (ϕ) Angles

RCS curves are normally produced using highly specialized off-line tools which input the model geometry and material attributes (typically an OpenFlight file) and applies physics-based processing like geometric ray-tracing, optical reflections/refractions, corner detection, material absorption and so on to the geometric data representation of the model. This processing is computationally expensive and is usually performed in non real-time. The end-result of this computation (usually 2D arrays of data points in elevation and azimuth) provides data that can be used more efficiently by simulation modeling such as radar at run-time. Those data curves are stored in a polar-type representation table, which provide specific reflectivity levels given a set of azimuth and elevation aspect angles.

A.14.1 Functional Description

To simulate a target for most modes of operation, the Radar software uses an RCS Polar Diagram as shown below:

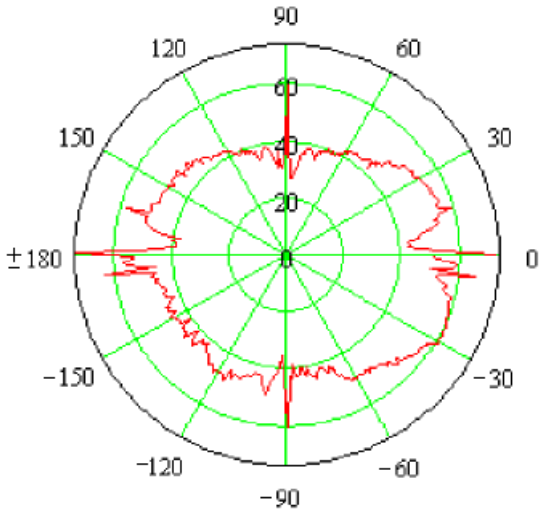


Figure A-20: Polar Diagram of RCS data in Decibels at a given elevation angle

The polar diagram allows the radar to use an RCS value array (indexed by azimuth/elevation angles) for getting an approximation of the overall RCS of distant targets. The approximated RCS data is a function of the model's materials, geometry, view angles, and multi-paths reflections generated within the model itself.

It can also be depicted more linearly as shown in the following diagram:

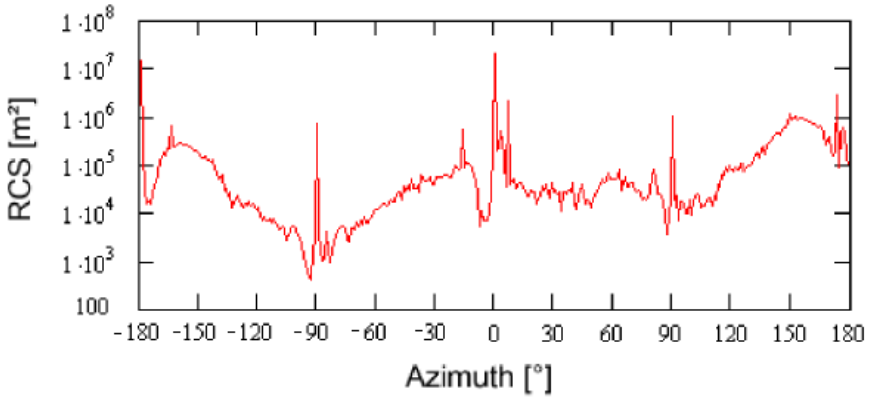


Figure A-21: Linear Diagram of RCS data in Decibels at a given elevation angle

As it can be seen in the example above, relative intensities are much greater when viewing the model directly in front (0° azimuth), from the back (±180° azimuth) and on the sides (-90° and +90° azimuth).

The RCS data is often characterized by its data resolution and physical modeling parameters. The data resolution determines the angular increments between successive RCS values, and modeling parameters specify the attributes of the physical parameters used to drive the RCS mathematical model computations (such as the Electro-Magnetic properties of the simulated electric field).

Both wavelength and polarization affect how a radar system "sees" the elements in the scene. Therefore, radar using different polarization and wavelength combinations may provide different and complementary information, which can be used to enhance the radar image in a specific way.

A.14.2 Wave Polarization

When computing an RCS model, it is important to consider microwave energy propagation and scattering, and also the polarization of the radiation, which is an important property. For a plane electromagnetic (EM) wave, polarization refers to the locus of the electric field vector in the plane perpendicular to the direction of propagation. The length of the vector represents the amplitude of the wave, and the rotation rate of the vector represents the frequency of the wave. Polarization refers to the orientation and shape of the pattern traced by the tip of the vector. (Reference [23])

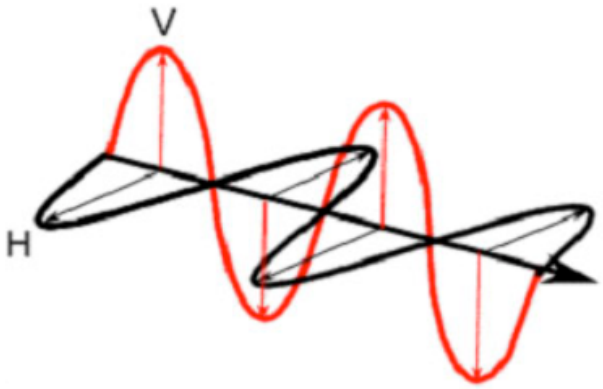


Figure A-22: Horizontal and Vertical Polarization of a plane of EM wave

The waveform of the electric field strength (voltage) of an EM wave can be predictable (the wave is polarized) or random (the wave is un-polarized), or a combination of both. In the latter case, the degree of polarization describes the ratio of polarized power to total power of the wave. An example of a fully polarized wave would be a monochromatic sine wave, with a single, constant frequency and stable amplitude.

Many types of radar antennae are designed to transmit and/or receive microwave radiation that is either horizontally (H) or vertically (V) polarized, or a combination of both. A transmitted wave of either polarization can generate a backscattered wave with a variety of polarizations, thus an equal amount of resulting RCS curves.

Polarization type on either transmission or reception mode can be synthesized by using H and V components, with a well-defined relationship between them. For this reason, systems that transmit and receive both of these linear polarizations are commonly used. With these radars, there can be four combinations of transmit and receive polarizations:

HH - for horizontal transmit and horizontal receive

VV - for vertical transmit and vertical receive

HV - for horizontal transmit and vertical receive, and

VH - for vertical transmit and horizontal receive.

The first two polarization combinations are referred to as “like-polarized” because transmit and receive polarization types are the same. The last two combinations are referred to as “cross-polarized” because transmit and receive polarizations are orthogonal to one another.

Radar systems can have one, two, or all four of these transmit/receive polarization combinations. Examples include the following types of radar systems:

Single polarized	HH or VV (or possibly HV or VH)
Dual polarized	HH and HV, VV and VH, or HH and VV
Alternating polarization	HH and HV, alternating with VV and VH
Polarimetric	HH, VV, HV, and VH

Both wavelength and polarization affect how a radar system “sees” the elements in the scene. Therefore, radar using different polarization and wavelength combinations may provide different and complementary information, which can be used to enhance the radar image in a specific way.

Therefore, polarization information is an important part of the CDB's RCS Data representation.

A.14.3 Wave Parameters

In addition to the wave polarization explained above, other physical parameters of the modeled electromagnetic wave are also a contributing factor to the RCS of a target when seen by Radar. Therefore those parameters are available in conjunction with the RCS data curves:

Those parameters are generally as follows:

- Radar Mode (Continuous wave or Pulsed)
- Radiating Frequency
- Antenna Main Lobe Gain
- Antenna Main Lobe Bandwidth
- Antenna Side Lobe 3dB point
- Radar Pulse width (if pulsed radar mode)
- Radar Pulse Repetition Frequency (if pulsed radar mode)
-

A.15 Information: Tide Simulation Modeling Alternatives

The availability of a Tide component permits realistic simulation of tides with a minimal computational overhead by the simulation application. Furthermore, the Tide component also permits simulation of tides whose amplitude varies differently with location. In order to determine the shoreline profile at a given location, the simulator client-devices must first determine the height of (say) the ocean in the immediate vicinity of that location. The sophistication of this calculation can vary greatly with simulation fidelity.

Figure A-23: Examples of Ocean Tide Simulation Fidelity in Simulator, illustrates examples of how tide simulation might be handled. At the low-end of the fidelity spectrum, the tide level (expressed as a value between –100% (average low tide) and 100% (average high tide) could be provided directly at the simulator's control console. In a high-end simulation, one could develop a simulation of the earth's oceans that takes into account Bathymetry profile of the oceans and the ephemeris model (particularly moon and sun) as a function of time and date. Regardless of simulation fidelity, the CDB internal representation facilitates the work of simulation client devices that are interested in obtaining the shoreline profile and ocean heights.

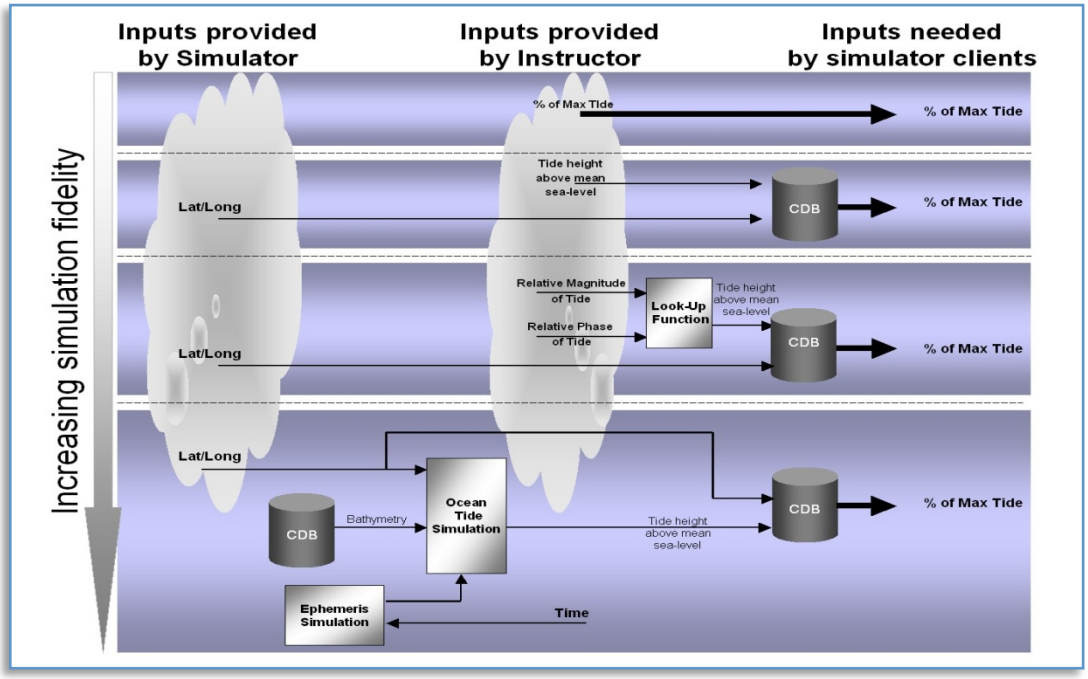


Figure A-23: Examples of Ocean Tide Simulation Fidelity in Simulator

A.16 Information: Comparison of CDB to FalconView Structure

While the CDB file naming convention and its directory structure are somewhat different from that used in FalconView, it is possible to find equivalent files between the two.

The FalconView directory structure contains some metadata describing its content and area coverage; it has a three level directory structure. The first level "rpf" is a raster product format: the second level being the dataset such as "gnc" (global navigation chart): and the third level relates to the zones; all files are under the third level. The file name is eight characters long followed by a three-character file extension, and the file name portion uses a radix 34 numbering notation that is based on the position of the frame in the zone as well as revision info and the producer ID key. The file extension is based on the dataset and the zone. Note that frames are equivalent to CDB tiles.

From information such as a given lat/lon, a given resolution such as one-meter pixel size and the dataset such as global navigation chart, it is possible to generate the corresponding FalconView file name and its path. Similarly, given a lat/lon, an LOD and a dataset it is possible to generate a CDB file name and its path. Though not identical in coverage and resolution these two files should be similar in content for the same dataset.

Note that when given a CDB file name, it is possible to extract the tile position in lat/lon, the dataset it belongs to, and the LOD, even its full path name, i.e. the file name is unique for the

entire CDB. This is not the case for FalconView. Since the resolution is not implicit in the name, the file itself must be read to extract this information; the dataset and zone info can be extracted from the file extension. Also note that directories in FalconView can potentially be very large since all files in a zone reside in the same directory; this is especially true for fine resolutions.

The FalconView directory structure follows the guidelines and conventions specified by MIL-STD-2411.

The algorithms used to find file name are given by examples within the MIL-C-89041 Controlled Image Base (CIB) document; in that document, zones are shown as overlapping. Note that this may not reflect the manner in which FalconView was implemented; nonetheless this does not affect the methodology provided in this section.

A.16.1 FalconView Directory structure

In FalconView, a top-level directory contains files that are metadata containing information about the various datasets and files in the directories.

The FalconView directory structure is as follows:

Falconviewmaps

+---covdata Coverage data

| cgnc.cov Global Navigation charts

| cjga.cov Joint Operation Graphics Air

| cjnc.cov Jet Navigation Chart

| conc.cov Operational Navigation Chart

| ctpc.cov Tactical Pilotage Chart

| mm100.cov 1:100,000 maps

| mm250.cov 1:250,000 maps

| sigfile.sig

| trs_8km.cov Township Range Section

|

+---rpf Raster Product Format

+---cgnc	Global Navigation Map
| | +---1	Zone
| | | 00023023.GN1	File Name

A.16.2 FalconView Zones definition

MIL-STD-2411 divides the world into 18 zones, nine in the northern hemisphere and nine in the southern hemisphere. The first eight zones in both hemispheres are divided into frames, which in turn are divided into sub-frames. Frames are made of pixels with 1536 x 1536 pixels in a frame; there are 36, 6x6 sub-frames per frame. Between each zone, there is an overlap of one frame; this implies that the size of zones will vary slightly depending on the resolution that is used. Table A-5 Zones Range No Overlap gives the approximate range of each zones; 1 – 9 in the north, A - J in the south. The two extreme zones, which cover the north and south poles, use a different scheme and are not discussed here.

Table A-5 Zones Range No Overlap

Zone	Zone Extent No overlap (deg)	Zone extent No overlap (deg)
1, A	0	32
2, B	32	48
3, C	48	56
4, D	56	64
5, E	64	68
6, F	68	72
7, G	72	76
8, H	76	80
9, J	80	90

A.16.3 FalconView Zone resolution

Along lines of constant longitude, the pixel constant used to determine the size of frames is a function of the resolution but is independent of the zone. Along lines of constant latitude the constant is a function of both resolution and zone and is based on the mid latitude of the zone.

Table A-6 Example Resolution east-west pixel constants that is extracted from MIL-C-89041 enumerates the factors for three resolutions.

Table A-6 Example Resolution east-west pixel constants

Zone	Pixel constant (10 meter product)	Pixel constant (5 meter product)	Pixel constant (1 meter product)
1,A	3,696,640	7,393,280	36,966,400
2,B	3,025,920	6,051,840	30,259,200
3,C	2,457,600	4,915,200	24,576,000
4,D	1,991,680	3,983,360	19,916,800
5,E	1,633,280	3,266,560	16,332,800
6,F	1,372,160	2,744,320	13,721,600
7,G	1,100,800	2,201,600	11,008,000
8,H	824,320	1,648,640	8,243,200
Lat	1,000,960	2,001,920	10,009,600

The north-south or latitudinal pixel constant is the number of pixels from the equator to the pole (90°). The east-west pixel constant is the number of pixels longitudinally from the 180° west longitude meridian going 360° in an easterly direction along the zone midpoint.

A.16.4 FalconView Zone extension based on resolution

To illustrate, we will use as an example a resolution of 10 meters. To calculate the exact latitudinal zone extent for a given resolution, first calculate the number of pixels in a degree of latitude for the resolution $N_\phi = \frac{1000960}{90} = 11121.7777$

The number of frames needed to reach the nominal zone boundary is the number of pixels per degree of latitude multiplied by the nominal zone boundary (in degrees), divided by 1536, the number of pixels rows in a frame, and rounded up to the nearest integer. For example in the first zone the number of frames is $Roundup\left(\frac{32N_\phi}{1536}\right) = 232$

The extent of the zone is then $\frac{1536 \times 232}{N_\phi} = 32.0409207$

In order to find the extent of the next zone we use the following method, which applies to all zones from 2 to 8 or B to H.

Since there is an overlap of one frame the start point of the zone 2 will be $\frac{1536 \times 231}{N_\phi} = 31.9028133$ the number of frames required to reach the next zone which nominally is at 48 is: $Roundup\left(\frac{(48 - 31.9028133)N_\phi}{1536}\right) = 117$ and the extent is $31.9028133 + \frac{1536 \times 117}{N_\phi} = 48.0613811$

The number of longitudinal frames and subframes is computed by determining the number of subframes to reach around the earth along a parallel at the zone midpoint. The east-west pixel constant is divided by 256 pixels to determine the number of subframes. The results are divided by 6 and rounded up to obtain the number of frame columns.

For example longitudinally in the first zone we get $Roundup\left(\frac{3696640}{256}\right) = 14440$ subframes and $Roundup\left(\frac{14400}{6}\right) = 2407$ frames. Table A-7 Frame/Subframe Sizes for Source Image GSD of 10 Meters, shows the complete set for a resolution of 10 meters.

Table A-7 Frame/Subframe Sizes for Source Image GSD of 10 Meters

Zone Number	Subframes in Zone (Rows) Latitudinal	Frame Rows in Zone Latitudinal	Equator-ward Zone Extent with Overlap	Pole-ward Zone Extent with Overlap
1,A	1,392	232	0°	32.0409207
2,B	702	117	31.9028133	48.0613811
3,C	354	59	47.9232737	56.0716113

4,D	348	58	55.9335038	64.0818414
5,E	180	30	63.9437340	68.0869565
6,F	180	30	67.9488491	72.0920716
7,G	180	30	71.9539642	76.0971867
8,H	180	30	75.9590793	80.1023018
9,J	— — —	— — —	varies	90°

Zone Number	Subframes (Columns) Longitudinal	Frames (Columns) Longitudinal	E-W Pixel Constant
1,A	14,440	2,407	3,696,640
2,B	11,820	1,970	3,025,920
3,C	9,600	1,600	2,457,600
4,D	7,780	1,297	1,991,680
5,E	6,380	1,064	1,633,280
6,F	5,360	894	1,372,160
7,G	4,300	717	1,100,800
8,H	3,220	537	824,320

A.16.5 FalconView Frame Position

MIL-C-89041 states that "the origin for counting nonpolar frame rows and columns is the southernmost latitude of the zone and 180° west longitude, with columns counted in an easterly direction from that origin, as opposed to frames and subframes where "the origin for the subframe and pixel numbering within frames and subframes shall be from the upper left corner".

For a given latitude ϕ and longitude λ the row and column for the frame where that geographic position is situated can be computed. The determination of the zone is derived from the latitude except at the border of zones where an overlap exists and the zone must be picked.

The row is given by $F_R = \text{int}\left(\frac{(\phi - \phi_{szr})N_{\phi r}}{1536}\right)$ where ϕ_{szr} is the bottom southern most latitude of the zone at resolution *r* and $N_{\phi r}$ is the number of pixels per degrees of latitude at resolution *r*.

Similarly the column corresponding to the longitude is given by $F_C = \text{int}\left(\frac{(180 + \lambda)N_{\lambda zr}}{1536}\right)$ where $N_{\lambda zr}$ is the number of pixel per longitudinal degrees in zone *z* at resolution *r*, λ ranges from –180 to 180.

As an example, for latitude of 36 degrees and longitude of –88 degrees we would get for a resolution of 10 meters

$$N_{\phi r} = \frac{1000960}{90} = 11121.778$$

$$N_{\lambda zr} = \frac{3025920}{360} = 8405.333 \text{ since we are in zone 2}$$

$$F_R = \text{int}\left(\frac{(36 - 31.9028133) \times 11121.778}{1536}\right) = 29$$

$$F_C = \text{int}\left(\frac{(180 + (-88)) \times 8405.333}{1536}\right) = 503$$

A.16.6 FalconView File Naming Convention

MIL-C-89041 for Controlled Image Base (CIB) states that:

"The naming convention for all resolutions of images registered in MIL-STD-2411-1, where it is intended for producers to provide contiguous [frame file] coverage, shall conform to MIL-STD-2411. In addition, the CIB [frame file] names are further restricted to conform to the form "ffffffvp.ccz." The "ffffff" portion of the name shall be a radix 34 value that encodes the unique cumulative frame number within a zone in base 34, with the right-most digit being the least significant position. The radix 34 value incorporates the numbers 0 through 9 and letters A through Z exclusive of the letters "I" and "O" as they are easily confused with the numbers "1" and "0". For example, the "ffffff" portion of the names would start with "000000," proceed through "000009," "00000Z," "000010," and so forth until "ZZZZZZ." This allows 1,544,804,416 unique [frame file] names; a contiguous grid of frame names down to a resolution of 0.2 meters (approximately 8 inches) can be defined. The "v" portion of the name shall be a radix 34 value that encodes the successive version number. The "p" portion of the name shall be a radix 34 value that designates the producer code ID, as defined in MIL-STD-2411-1. The "cc" and "z" portions of the name extension shall encode the data series code and the zone, respectively, as defined in MIL-STD-2411-1. The CIB producers are responsible to ensure that [frame files] for all image resolutions, zones, and revisions, have unique names."

In our case: $\text{ffffff} = F_C + F_R \times N_{crz}$

… where N_{czt} is the number of columns in zone z for a resolution r

In the example of a lat of 36 and lon –88 with a resolution of 10 meters we get:

ffffff = 503+29x1970=57633 or 001FV3$_{(34)}$

… where 1970 is number columns in zone 2 as given in Table A-7 Frame/Subframe Sizes for Source Image GSD of 10 Meters, and in RADIX 34 we get ffffff = 001FV3 ; for a global navigation chart dataset a version level 0, a manufacturer code of 3 and zone 2 the file name would be equal to "001FV303.GN2"

Note that nothing in the file name defines the resolution for the data; this information is part of the [coverage section] in the file itself (see section 3.12.3 in MIL-C-89041). Also note that the file name is unique only to the zone at a given resolution.

On the other hand a similar file for imagery (VSTI, Visible Spectrum Terrain Imagery) in the CDB convention for an LOD of 04 which has a resolution of approximately 8 meters; at position lat 36 and lon –88 we would get for the file name:

```
\CDB\Tiles\N36\W088\004_Imagery\L04\U0\
   N36W088_D004_S001_T001_L04_U0_R0.jp2
```

Note that the file name itself is unique worldwide and that from it we can derive the directory path to which it belongs.

A.17 Information: JPEG-2000

The CDB specification supports JPEG2000 for both VSTI and VSTLM component data.

As a result of the high rates of compression there are no real advantages to be gained in supporting a broad range of alternate color representations (such as single channel representations, indexed color representations, RGB-triplet color encoding such as 5-6-5, etc.). The underlying motivation behind all such schemes is driven by a desire to reduce storage and transmission bandwidths. JPEG-2000 achieves these goals and many others, refer to Table A-8 JPEG 2000 Features.

Table A-8 JPEG 2000 Features

High compression efficiency: **Compression better than 0.25 bits per pixels, 20% compression efficiency improvement over JPEG.**	High dynamic range: **Compress images with various dynamic ranges (e.g. 1-16 bit) for each color component.**

Lossless and lossy compression: Lossless compression ratios approx. 1.7:1.	**Seamless quality / resolution scalability:** Without having to download the entire file.
Progressive image reconstruction: Allows images to be reconstructed with increasing pixel accuracy and resolution.	**Large images sizes - up to (2^{32} - 1).**
Perceptual color space internal coding.	**Single decompression architecture.**
Region of interest coding: Permits certain ROI's in the image to be coded and transmitted with better quality and less distortion than the rest of the image.	**Error resilience during transfers.**

A.18 Information: Managing CDB Versions

The incremental versioning mechanism of the CDB provides a fast method of creating versions of the CDB changes since all the data changes are located under a single root directory. The creation and the managing of the (incremental) data files are however under the application control.

A CDB can simultaneously hold multiple incremental versions of the data. As a result, it is possible to select any of the versions without transferring or copying files. Consider the case where a database generation facility, a database quality assurance facility, a simulator mission planning facility, a mission rehearsal facility and a mission debrief are all operating concurrently on distinct versions of the CDB. This is illustrated in Figure 3 2: Concurrent Usage of Versions of the CDB. By the fourth day, there are four versions of the CDB, say the active default CDB (v1) and three incremental versions (v2, v3, v4). Any of these four versions can be instantly invoked (without copying or transferring files) by the simulator operator at the Mission Rehearsal facility, or by an instructor at the Mission Debrief facility.

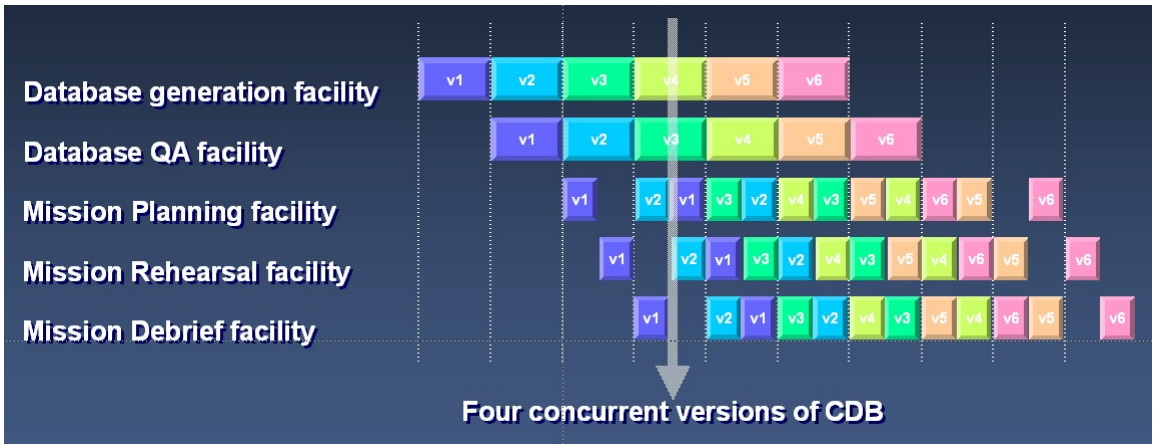


Figure A-24: Concurrent Usage of CDB Versions

The underlying CDB versioning mechanism is a fine-grained file-level mechanism, i.e., only the affected files of the geographic areas of the CDB need to be versioned, leaving the rest of the CDB intact. This approach is invaluable in mission rehearsal applications where the target areas of the CDB require frequent updates based on the latest intelligence data.

The approach can also be applied to the handling of classified secure data. In this case, a CDB version can be used to hold the portion of the CDB that contains the classified information. The incremental versioning mechanism would be used to segregate the classified portion of the CDB onto a separate storage medium. Since the classified portion of the CDB is embedded within the overall CDB structure, it is possible for the runtime publishers to instantly switch back and forth between the classified and non-classified versions of the database.

A.19 Guideline: Handling of GS and T2D Models

A.19.1 GSModels

A.19.1.1 GSModel Levels-of-detail

The insertion of a 3DModel-LOD into the LOD hierarchy of the GSModel Dataset is solely dependent on its Location, its Significant Size and on its Storage Size.

The location and Significant Size of a 3DModel-LOD determines where it is nominally inserted into the GSModel Dataset hierarchy. This approach ensures that the modeled content is organized in files that contain co-located objects of similar size. *This approach provides client-device with an optimal means of accessing and filtering modeled content (by location and by size).*

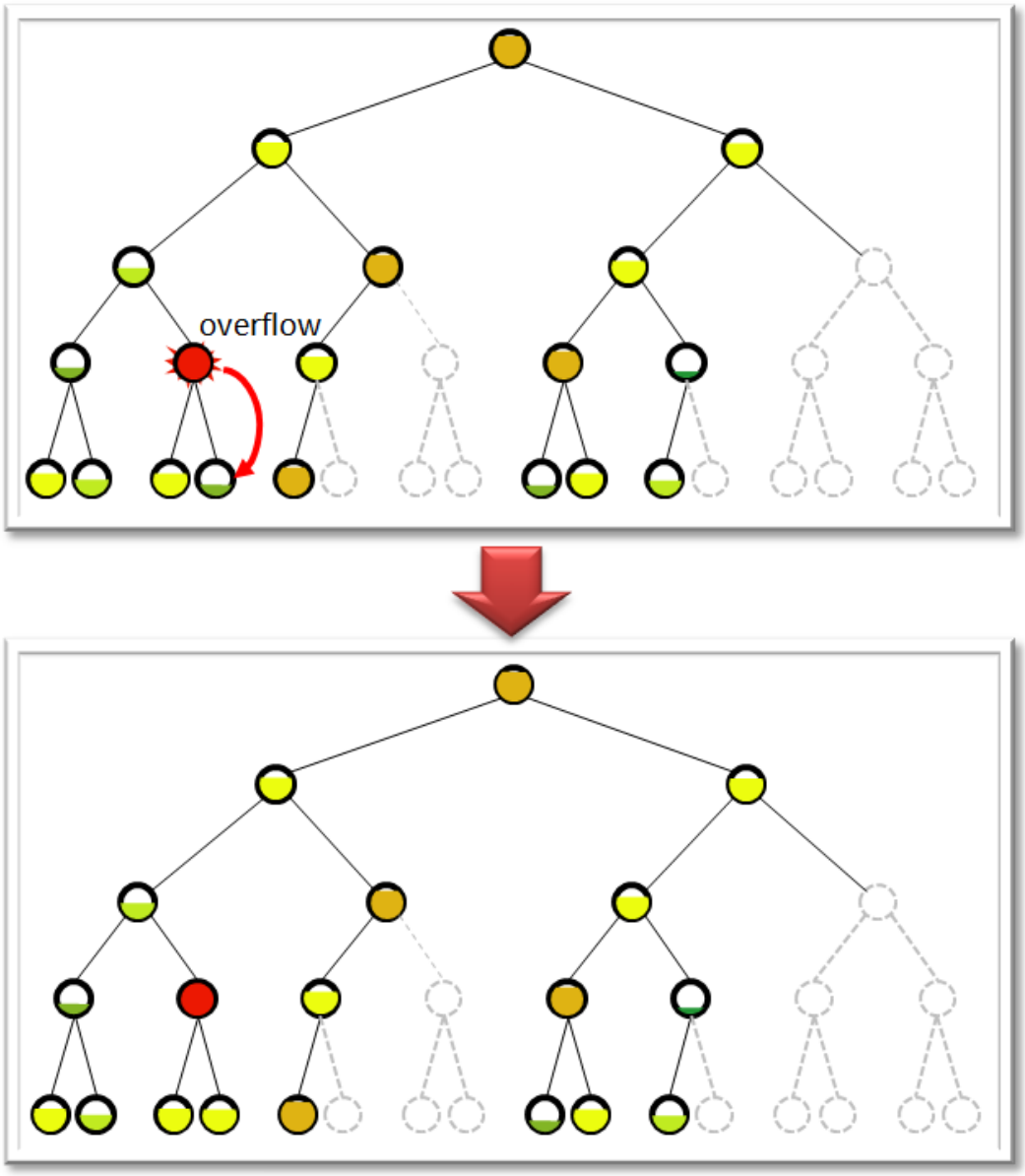


Figure A-25: Handling Tile-LOD Overflows in GSModel Dataset

3DModel-LODs are accumulated into the Tile-LODs of the GSModel hierarchy. The size of these Tile-LODs is capped to *GSModelFileSize*. In the event that a group of 3DModel-LODs nominally assigned to a Tile-LOD causes this limit to be exceeded, the 3DModel-LODs that are deemed to have the lowest contribution to the Tile-LOD are moved to finer (children) Tile-LODs until the Tile-LOD is once again within its size limit (illustrated in Figure A-25: Handling Tile-LOD Overflows in GSModel Dataset). In the event that a 3DModel-LOD is itself larger than *GSModelFileSize*, the 3DModel-LOD is moved to the 4 finer Tile-LODs of the GSModel

Dataset hierarchy. *This approach ensures that the modeled content is accessible in chunks that are bounded; this improves the allocation and management of memory allocation in the client-devices.*

NOTE: The Significant Size of a 3DModel-LOD determines where it is nominally inserted into the 3DModel LOD hierarchy. In this nominal case, each Tile-LOD of the 3DModel Dataset holds a group of 3DModels-LODs that have similar Significant Sizes. This enables the client-devices to determine the range at which the 3DModel-LOD can be optimally blended-in to the scene (so that the model falls within a specified angular error criterion).

The bounding criterion of 3DModel Tiles can lead to LOD migration, thus breaking the relationship between the Significant Size of a 3DModel-LOD and the nominal CDB LOD it belongs to. As a result, client-devices can no longer guarantee the range at which the 3DModel-LOD will be blended-in to the scene. In effect, each time the 3DModel-LOD is migrated by one LOD, the client-device will likely shorten the range at which it is blended into the scene by a factor of 2X, leading to potentially distracting artifacts. The severity of the artifacts is proportional to the amount of content that has migrated to finer LODs and to the number of LODs by which the content has moved.

While the CDB Specification allows the migration of 3DModel-LODs to finer LODs when Tile-LODs overflows are encountered, it is understood that this may lead to rendering artifacts that might be considered unsatisfactory. ***Consequently, it is strongly recommended that tools (that generate the CDB hierarchy) be designed to optionally disallow the migration of 3DModel-LODs to finer LODs upon overflows, and instead flag the overflow condition and then abort.*** Upon such cases, modelers can then re-assess which 3DModels should be discarded or remodeled in order to simultaneously satisfy the CDB bounding criteria and the application requirements.

Each of the 3DModel-LODs is nominally configured as exchange-LODs. The exchange-LOD mechanism assumes that client-devices gradually substitute a coarser 3DModel-LOD located in a coarser Tile-LOD with a finer 3DModel-LOD located in a finer Tile-LOD.

While this exchange-LOD mechanism is simple, it can lead to inefficiencies when extremely fine features cause the GSModel Dataset hierarchy to be extended by several LODs. Consider the case of a 1 meter road sign located next to a large building (30m wide x 30m long x 10 m high). As we will see in the following section, the road sign would nominally be inserted at LOD 9 of the GSModel Dataset hierarchy. Conversely, the large modeled building would nominally be

inserted at LOD 4. The road sign forces the GSModel Dataset hierarchy to be extended by 5 additional LODs.

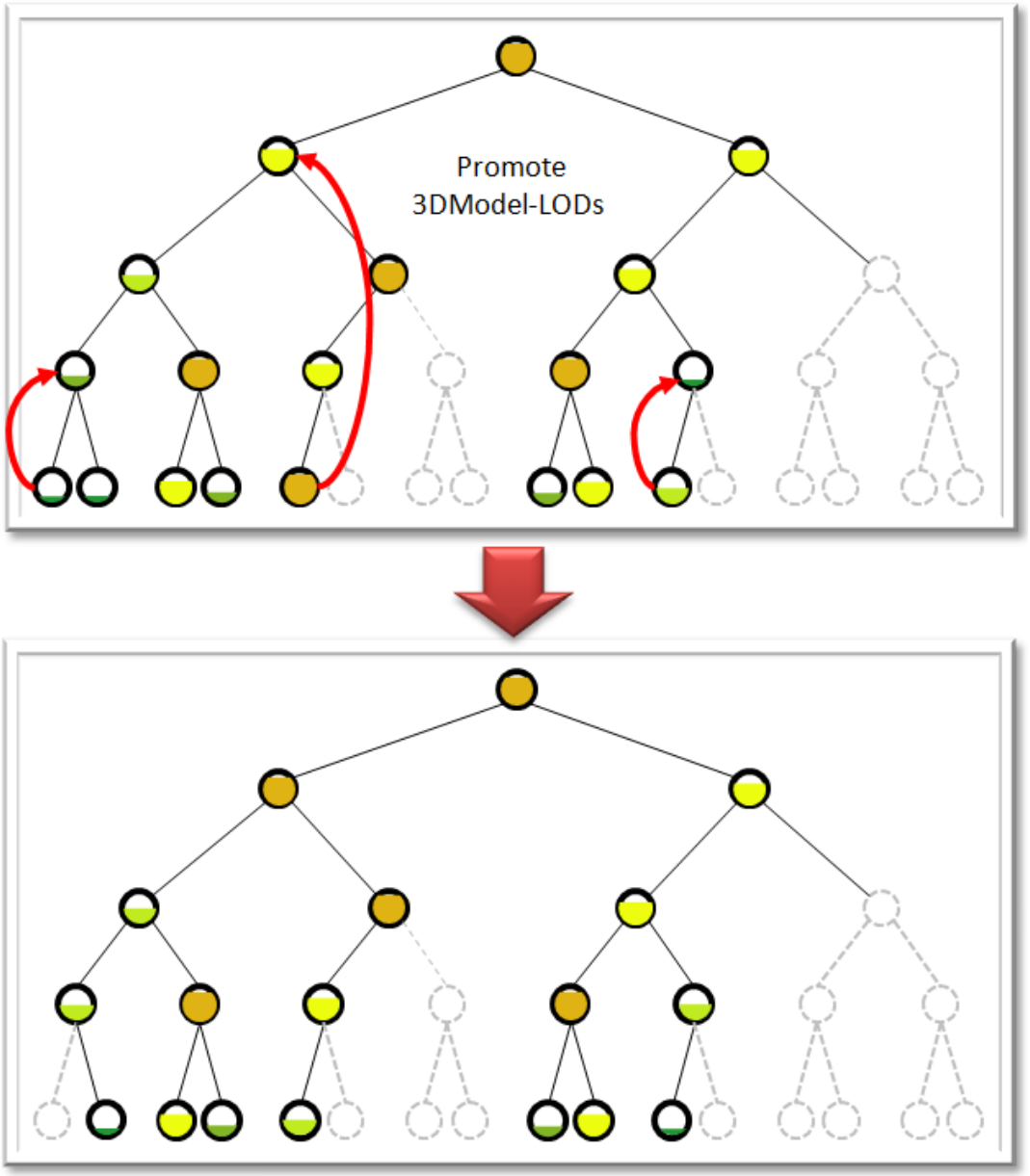


Figure A-26: Compacting the GSModel Dataset

In order to reduce the depth of the LOD hierarchy, the GSModel Dataset is post-processed and subjected to a "compaction" process, starting from the finest LOD (e.g. LOD_{max}) and progressing to the coarser levels. The compaction process takes finer 3DModel-LODs and appends them to

the corresponding 3DModels in coarser Tile-LODs of the GSModel Dataset. The appended (finer) 3DModel-LOD must have an explicit OpenFlight LOD node with the Significant Size of the 3DModel-LOD; this provides the necessary information for the client-device to control the range at which the 3DModel-LOD will be introduced into the rendered scene. The process is recursively applied to the coarser LODs until the parent LOD is packed to capacity. *This approach ensures that the modeled content is accessible in similarly-sized chunks of processing; this provides the means to improve internal parallelism and pipelining (i.e. improves client-device determinism)*

The access and selection of 3DModel-LODs is done through the GSFeature Dataset. Each of the Tile-LODs of the GSFeature Dataset contains a list of Features; each Feature in turn points to a 3DModel-LOD at the appropriate LOD. In effect, the appearance of a Feature (along with its modeled representation) and the evolution of its modeled representation are entirely controlled by the GSFeature Dataset. As a result, the 3DModel-LODs of a 3DModel need not be located in consecutive LODs of GSModel Dataset hierarchy.

A.19.1.2 CDB LOD versus GSModel Significant Size

Chapter 6 provides a set of guidelines to establish the values for Significant Size SS_c and SS_{LOD} for GSModels.

Table 3 1: CDB LOD vs. Model Resolution shows the nominal position of a GSModel within the LOD hierarchy of the GSModel Dataset. Note all of the GSModel-LODs of a GSModel normally fall within a range of 8 levels-of-detail (i.e. the smallest tile size the GSModel can sit on). However, it is possible to extend this range by breaking up a GSModel-LOD into several OpenFlight files.

Here is a summary of the rules required by the CDB Specification in order to ensure deterministic operation from client-devices:

1. Each feature may have multiple modeled representations at progressively coarser levels of detail. Each of the modeled representations is referred to as a GSModel-LOD. In absence of pre-modeled coarser LOD representations, the tools may automatically generate coarser modeled levels-of-detail.
2. A GSModel-LOD consists of a group of polygons that represent a feature at a specific level-of-detail; this group of polygons shares a unique Model Identifier derived from the Feature Attribute Code (FACC), a Feature Sub-Code (FSC), a Model Name (MODL or MMDC), the GSModel-LOD's Significant Size SS'_{LOD}.
3. Each GSModel has a distinct Significant Size *SS'* value based on its dimensions. In turn, each GSModel-LOD of a same GSModel has a distinct Significant Size value SS'_{LOD} based on its modeled accuracy.
4. Insertion of a GSModel-LOD into the GSModel Dataset hierarchy proceeds as follows. Starting with LOD_{max} (LOD_{max} is a variable set by the user that sets the maximum depth of the LOD hierarchy) and progressing to coarser LODs…
   a. For each Tile-LOD, create a Model_List that is constructed from the GSModel-LODs that straddle the Tile-LOD.

i. If the GSModel-LOD is not the coarsest LOD and its Significant Size is in accordance to Table 3 1: CDB LOD vs. Model Resolution, then add it to the Tile-LOD. Only the coarser GSModel-LODs of this GSModel are available for future insertion into the GSModel LOD hierarchy.
ii. If the GSModel-LOD is the coarsest LOD of the GSModel and its Significant Size is in accordance to Table 3 1: CDB LOD vs. Model Resolution, then insert it at this LOD of the hierarchy. If the GSModel-LOD matches the Tile-LOD, remove it from the list for the processing of the coarser Tile-LOD.

b. If the Model_List is less than *GSModelFileSize*, no further processing is required.

NOTE: The Storage Size of (statically-positioned) MModels is assumed to be zero.

c. The Model_List of each Tile-LOD is sorted in decreasing order of Diff, where Diff is the difference between the Significant Size SS of the Model and the Significant Size as specified in Table 2.
d. If the size of the Model_List is greater than *GSModelFileSize*, then (starting with the first entry in the sorted Model_List), Models are simplified one-by-one until the size of the Model_List is less than *GSModelFileSize*. When a simplification occurs, the Model_List is re-sorted using the Diff value.
e. If a) the Model_List is deemed non-reducible and b) the Model_List is still greater than *GSModelFileSize* …
   i. If LOD < LOD_{max}, then…
      (1) a Temp_Model_List is created and initialized with the contents of the Model_list. Starting from the end of the Model_List, Models are removed one-by-one from the Model_list (starting with the first Model in the Model_List) and are copied into the Temp_Model_List until the Model_List reaches *GSModelFileSize*.
      (2) The Temp_Model_List is merged to the children Tile-LODs and the children are re-processed using steps 4a to 4e. The process is iterative, i.e., the "overflow" is propagated into the finer LODs of the GSModel hierarchy.
   ii. Else…
      (1) Models are removed one-by-one, starting with the first Model in the Model_List, until the Model_List is less than *GSModelFileSize*. The corresponding GSModels are removed from the CDB and a warning is issued stating that content was removed

NOTE: It is strongly recommended that GSModels be modeled using several GSModel-LODs, spanning a wide range of fidelity. The availability of many LODs ensures suitability of the resulting CDB for real-time use with a minimum degradation in fidelity. Conversely, a low number of LODs can lead to unacceptably large steps in fidelity.

NOTE: It is strongly recommended that the coarsest modeled LOD of GSModels have no more than 128 vertices; this reduces the likelihood that the coarsest modeled LOD need be propagated to a finer LOD of the hierarchy.

NOTE: This algorithm preserves the highest available modeled content while ensuring that the runtime constraint file size limits are respected. While the CDB data model allows for infinitely-sized GSModel-LODs, a client-device may refuse to render the GSModel-LOD if it has insufficient memory to load all of the OpenFlight files that make-up the GSModel-LODs.

5. Each GSModel-LOD is subject to an OpenFlight file size limit of *GSModelFileSize*, i.e. several OpenFlight files, each within the *GSModelFileSize* limit, can be used to represent a very complex GSModel-LOD. Each of OpenFlight files that form the GSModel-LOD share the same GSModel-LOD Identifier (see rule 2) and GSModel-LOD origin. Client-devices must render the GSModel-LOD in its entirety, even if it is allocated to several OpenFlight files.

NOTE: While the CDB data model allows for infinitely-sized GSModel-LODs, a client-device may refuse to render the GSModel-LOD if it has insufficient memory to load all of the OpenFlight files that make-up the GSModel-LODs.

6. Each Tile-LOD is subject to a file size limit of *GSModelFileSize*.
7. All of the GSModel-LODs in a GSModel OpenFlight file are nominally exchange-LODs (see exception in next rule).
8. The depth of the GSModel LOD hierarchy should be reduced by folding-in the finer GSModel-LOD located in a finer Tile-LOD to the next coarser Tile-LOD of the hierarchy. Failure to perform this "compaction step" may result in significantly deeper GSModel LOD hierarchy when the finest GSModel-LODs consist of small features or small details on the same features (e.g., small posts next to a terminal building or fine window details on a large building).
9. The finer modeled representation of a GSModel (i.e. a GSModel-LOD with a smaller Significant Size) always appears in finer LODs of the GSModel Dataset LOD hierarchy than a coarser GSModel-LOD.
10. A Tile-LOD cannot contain more than one GSModel-LOD of the same GSModel.
11. Once inserted into the GSModel Dataset LOD hierarchy, there is no mandatory requirement to clip the contents of a GSModel Tile-LOD against its Tile-LOD boundaries. However, the contents of the GSModel Tile-LOD cannot protrude Tile-LODs by more than ½ the dimension of the Tile-LOD.

12. There is no mandatory requirement to have consecutive GSModel-LODs in consecutive LODs of Tile-LOD hierarchy; it is permissible to have gaps within the Tile-LOD hierarchy.
13. Gaps in the LOD file hierarchy of the GSFeature Dataset are not permitted. This may result in Tile-LODs that are empty (e.g. without any GSFeatures). The presence of an empty Tile-LOD file for the GSFeature Dataset indicates the availability of modeled content invoked by finer LODs of the GSFeature hierarchy.

A.19.1.3 Example – Insertion of a GSModel with 3 LODs into the CDB Hierarchy

Consider an industrial building 200m wide x 200m length x 10m high. The modeler has not supplied any values for its Significant Size, nor has he provided a value for RTAI. It is modeled in three distinct levels of detail as follows:

a) Coarsest level: 5 polygons
b) Mid level: 60 polygons
c) Finest level: 300 polygons

Based on this information, we can derive Significant Size values for each of the modeled representation as follows and determine where within the hierarchy each of the GSModel-LODs should be inserted:

a. Coarsest level-of-detail:

a. Compute the model's Significant Size …

$$SS = \sqrt{\frac{(10\times.96)\times200) + (200\times200\times.259)}{\pi}}$$

$$SS = 62.5m$$

b. Since the model is opaque and has no assigned value for RTAI, the final value for *SS'* is 62.5m.
c. Table 3 1: CDB LOD vs. Model Resolution, tells us that the (coarsest LOD) of the model should be nominally inserted at LOD 3 of the Tile-LOD (assuming its file size limit is not exceeded)

b. Mid level-of-detail:

a. Compute the ratio of vertices

$$R = {}^{V_{LOD}}/_{V_{coarsest}} = \frac{60}{5} = 12$$

b. Compute the Significant Size of the GSModel-LOD…

$$SS'_{LOD} = \frac{SS'_{coarsest}}{\sqrt{12}} = 18.04m$$

c. Since the model is opaque and has no assigned value for RTAI, the final value for *SS'* is 18.04m.

d. d. Table 3 1: CDB LOD vs. Model Resolution, tells us that the (mid- LOD) of the model should be nominally inserted at LOD = 5 of the Tile-LOD (assuming its file size limit is not exceeded)

c. Finest level-of-detail:

a. Compute the ratio of vertices

$$R = {}^{V_{LOD}}/_{V_{coarsest}} = \frac{300}{60} = 5$$

b. Compute the Significant Size of the GSModel-LOD…

$$SS'_{LOD} = \frac{SS'_c}{\sqrt{5}} = 8.07m$$

c. Since the model is opaque and has no assigned value for RTAI, the final value for SS'_{LOD} is 8.07.

d. d. Table 3 1: CDB LOD vs. Model Resolution, tells us that the (finest-LOD) of the model should be nominally inserted at LOD = 6 of the Tile-LOD (assuming its file size limit is not exceeded)

A.19.2 T2DModels

The T2DModels are stored in the OpenFlight format. The CDB conventions described herein are designed to facilitate the integration of such models onto the terrain tile, hence the name "**T**iled **2D Models**". Each 2DModel can have one or more modeled representation (called a 2DModel-LOD) that represents the feature to a certain level of fidelity. 2DModel-LODs are re-grouped into T2DModel Tile-LODs; this re-grouping approach is designed to reduce the overheads associated with the access of 2DModel-LODs. Furthermore, T2DModel-LODs can be accessed without a prior reference to a corresponding feature in the GSFeature dataset.

The integration of T2DModels to the underlying terrain skin is performed by the client-devices at runtime. Historically, this integration has always been performed by the tools and was "baked-in" into the SE terrain skin during the offline database generation process. Many client-specific considerations went into the mechanisms required to support this integration and as a result, the resulting synthetic environments were very client-specific and did not scale easily to higher resolutions.

In line with CDB principles, the T2DModel Dataset defers this integration and imposes it on the consumers (not the producers) of synthetic environments. As a result, client-devices can independently access, manage and control each dataset, i.e., the Primary Elevation, the VSTI Imagery, the T2DModel, etc. This layered approach to synthetic environment production and consumption provides a much greater level of abstraction between the SE data model and the data models internal to each client-device. It is understood, that the deferral of the integration process imposes added functionality and computational requirements on the part of the CDB client-devices.

While it would be possible, in theory, to use the T2DModel Dataset for the modeling of the terrain skin, this use-case is specifically forbidden because the T2DModel Dataset does not provide a guarantee of full tile coverage. As a result, the Primary Elevation Dataset is always required regardless of whether a corresponding Tile-LOD of the T2DModel is present or not.

Furthermore, since CDB forbids the duplication of information, the terrain skin cannot be duplicated by the T2DModel Dataset.

Client-devices must always access the Primary Elevation prior to any other raster datasets. Once a Tile-LOD of the Primary Elevation is loaded, a client-device can then access the T2DModel Dataset at an "appropriate" LOD[3]. Following this, the client-device must integrate the models found within the T2DModel Tile-LOD with the terrain found in the Primary Elevation dataset.

A.19.2.1 T2DModel Levels-of-detail

As with 3D features, 2D features can have modeled representations at varying levels of detail. Each of these modeled-representations is referred to as a 2DModel-LOD. A 2DModel-LOD consists of a group of polygons that represent a 2D feature at a specific level-of-detail.

Once a 2DModel-LOD is inserted into the T2DModel Dataset hierarchy, it is then referred to as a T2DModel-LOD. The insertion of a 2DModel-LOD into the LOD hierarchy of the T2DModel Dataset is solely dependent on its Location, its Significant Size and on its Storage Size. 2DModel-LODs are regrouped into files called T2DModel Tile-LODs. Note that when a 2DModel is clipped to the T2DModel's Tile-LOD boundaries, each of the clipped model fragments will appear in distinct OpenFlight files of the T2DModel Dataset. The T2DModel Tile-LODs are assembled into a hierarchy of Tile-LODs called the T2DModel Dataset.

The organization of the modeled content into files that contain co-located objects of similar size greatly improves runtime performance. The location and Significant Size of a 2DModel-LOD determines where it is nominally inserted into the T2DModel LOD hierarchy. This approach ensures that the modeled content is organized in files that contain co-located objects of similarly size. *This approach provides client-device with an optimal means of accessing and filtering modeled content (by location and by size).*

2DModel-LODs are accumulated into Tiles for each LOD of the T2DModel hierarchy. The size of these T2DModel Tiles is capped to *T2DModelFileSize*[4]. In the event that the insertion of a 2DModel-LOD causes this limit to be exceeded, the 2DModel-LODs that are deemed to have the lowest contribution to the Tile are moved to finer Tiles of the T2DModel hierarchy until the Tile is once again within its size limit. In the event that the 2DModel-LOD is larger than *T2DModelFileSize*, the 2DModel-LOD can be moved to the 4 finer Tiles of the T2DModel hierarchy and clipped against the Tile boundaries as illustrated in Figure A-27: Handling Tile-LOD Overflows within the T2DModel Dataset Hierarchy. *This approach ensures that the modeled content is accessible in chunks that are bounded; this is critical to the effective allocation and management of memory in the client-devices as well as improving client-device performance and determinism.*

[3] In this context, "appropriate" means a LOD that falls within the capabilities of the client-device.

[4] The *T2DModelFileSize* storage size limit for T2DModel Tile-LODs is critical in achieving runtime determinism.

NOTE: The Significant Size of a 2DModel-LOD determines where it is nominally inserted into the T2DModel LOD hierarchy. In this nominal case, each Tile-LOD of the T2DModel Dataset holds a group of 2DModel-LODs that have similar Significant Sizes. This enables the client-devices to determine the range at which the T2DModel-LOD can be optimally blended into the scene so that the model falls within a specified angular error criterion.

The bounding criterion of T2DModel Tiles can lead to LOD migration, thus breaking the relationship between the Significant Size of a 2DModel-LOD and the nominal CDB LOD it belongs to. As a result, client-devices can no longer guarantee the range at which the 2DModel-LOD will be blended into the scene. In effect, each time the 2DModel-LOD is migrated by one LOD, the client-device will likely shorten the range at which it is blended into the scene by a factor of 2, leading to potentially distracting artifacts. The severity of the artifacts is proportional to the amount of content that has migrated to finer LODs and to the number of LODs by which the content has moved.

While the CDB Specification allows the migration of 2DModel-LODs to finer LODs when Tile-LODs overflows are encountered, it is understood that this may lead to rendering artifacts that might be considered unsatisfactory. ***Consequently, it is strongly recommended that tools (that generate the CDB hierarchy) be designed to optionally disallow the migration of T2DModel-LODs to finer LODs upon overflows, and instead flag the overflow condition and then abort.*** Upon such cases, modelers can then re-assess which T2DModels should be discarded or remodeled in order to simultaneously satisfy the CDB bounding criteria and the application requirements.

Each of the Tile-LODs of the T2DModel Dataset is nominally configured as exchange-LODs (aka substitution-LODs) as defined in chapter 6.

The exchange-LOD mechanism assumes that client-devices gradually substitute a coarser Tile-LOD with a four finer Tile-LODs.

While this exchange-LOD mechanism is simple, it can lead to inefficiencies when extremely fine features cause the T2DModel Dataset hierarchy to be extended by several LODs. Consider the case of 13m road lineals overlaid with 6 cm stripe lineals. As we will see in the following section, insertion of the **Stripe** lineal would nominally occur at LOD=7 of the T2DModel hierarchy while the **Road** lineal would occur at LOD=-1. The Stripe lineals force the T2DModel Dataset hierarchy to be extended (and clipped) to 8 additional LODs. In effect, the Road lineals are repeated[5] in LODs 0 through 7 leading to important storage inefficiencies and greater computational burden by the client-devices.

[5] Since the nominal LOD mechanism is the exchange-LOD, and that gaps are not permitted in the LOD hierarchy

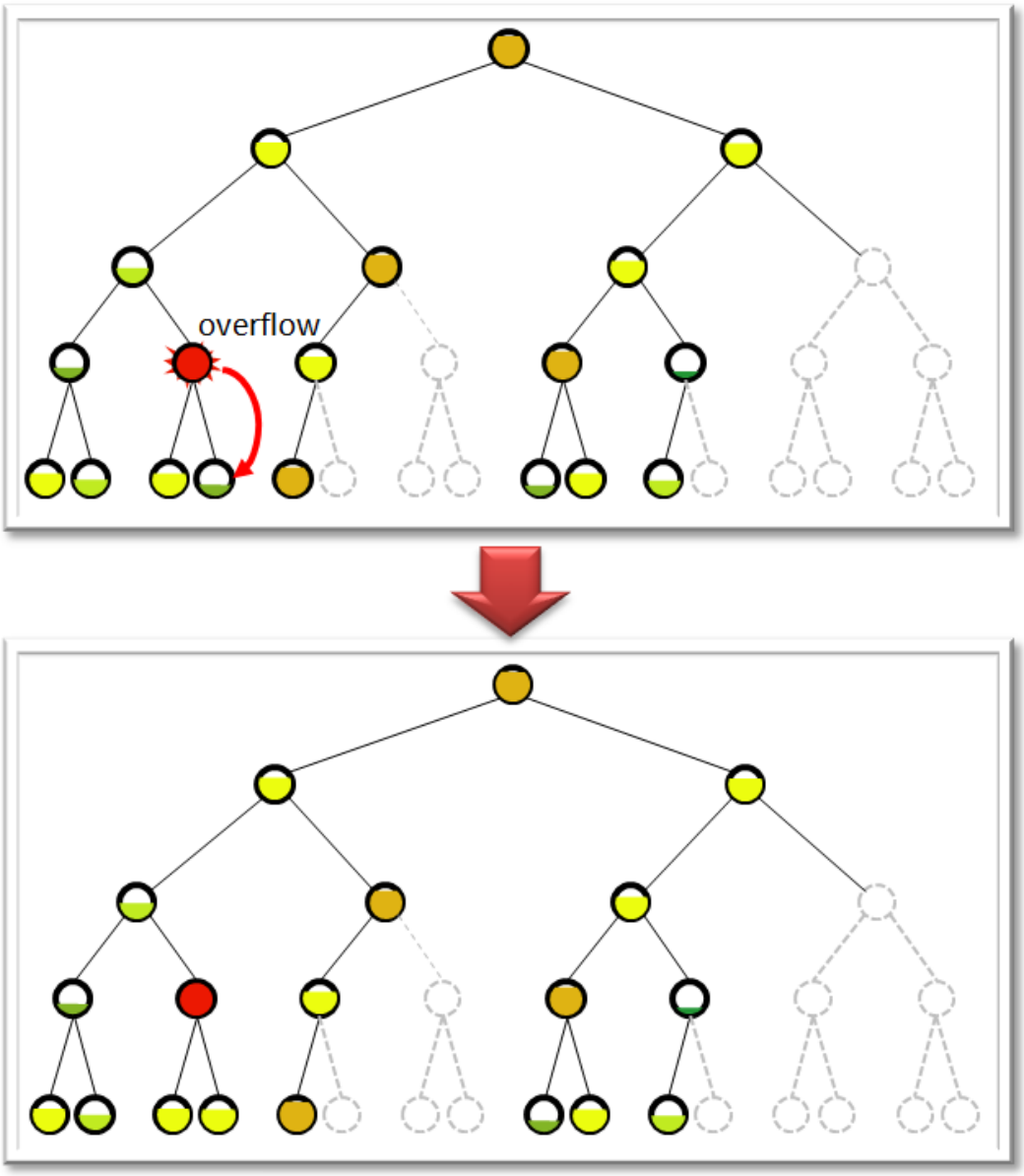


Figure A-27: Handling Tile-LOD Overflows within the T2DModel Dataset Hierarchy

In order to resolve this use-case, the T2DModel Dataset is post-processed and subjected to a "compaction" process, starting from the finest LOD (e.g. LOD_{max}) and progressing to the coarser levels. The compaction process takes the content of the Tile-LODs located at LOD_{max} and packs them as an additive LODs of the parent Tile-LOD at ($LOD_{max} - 1$) of the parent Tile-LOD. The process is recursively applied to the coarser LODs until the parent LOD is packed to capacity. *This approach ensures that the modeled content is accessible in similarly-sized chunks of processing; this provides the means to improve internal parallelism and pipelining (ie. improves client-device determinism).* The result is a LOD hierarchy which is less deep, and with content which is more uniformly distributed; both of these characteristics improve runtime performance and determinism.

The T2DModel LOD structure is continuous i.e. there is no gap in the LOD hierarchy. This means that once a 2DModel-LOD is inserted into a finer level of the T2DModel hierarchy, the same 2DModel-LOD is propagated to coarser LODs until a coarser 2DModel-LOD is available.

Note that some client-devices may be sensitive to the precision of clipped vertices; some client-devices may demand that the clipped vertices be shared at the tile boundary between two tiles of the same LODs. This can be done as follow:

- The X coordinate (longitude) of clipped vertices along the top or bottom edges of the tile can be used to uniquely identify the matching coordinate in an adjacent tile.
- The Y coordinate (latitude) of clipped vertices along the right or left edges of the tiles can be used to uniquely identify the matching coordinate in an adjacent tile.

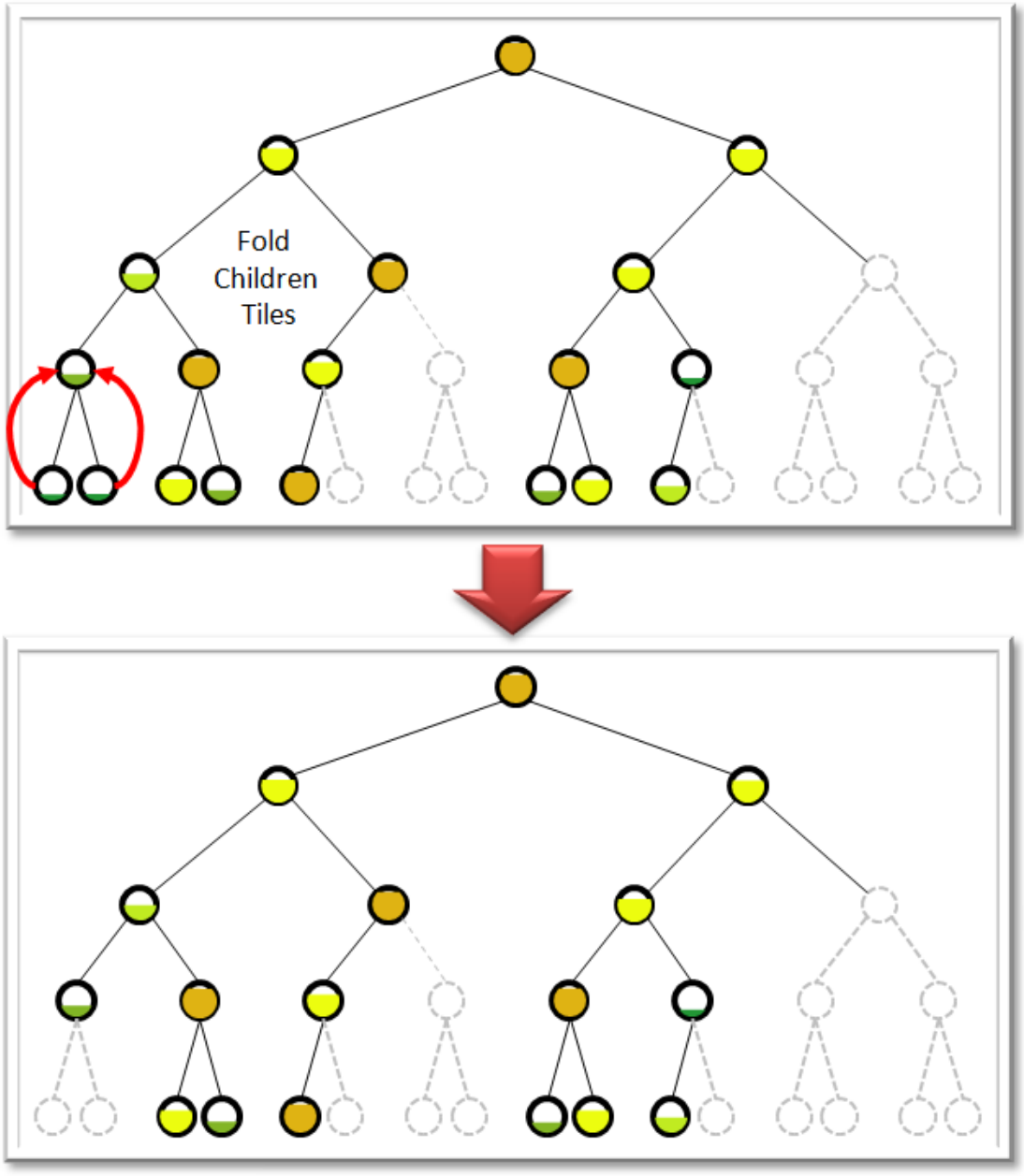


Figure A-28: Compacting the T2DModels Dataset Hierarchy

A.19.2.2 CDB LOD versus T2DModel Significant Size

Chapter 6 provides a set of guidelines to establish the values for Significant Size SS_c and SS_{LOD} for T2D Models (for both lineals and areals).

Table 3 32: T2DModel LOD versus Significant Size, shows us the relationship between SS_c and SS_{LOD}. They are offset by 3 LODs. The implication of this statement is in the case of a model with two LOD, the finer 2DModel-LOD must have sufficient detail to justify its existence.

> ***Note:*** Each of the 2DModel-LODs of a 2DModel must differ by at least one CDB LOD. Some 2DModel-LODs will be discarded if this relationship is not respected.

Consider for example a 12m lineal road feature with two modeled representations. The nominal CDB LOD for the coarsest 2DModel-LOD is LOD=3 in accordance to the table below. The Significant Size of the finer 2DModel-LOD is obtained by "walking" around its outline; we determine that the largest value of *d* for successive vertex triplets is 3m, hence SS_{LOD} = 3m. Table A-9: T2DModel LOD versus Significant Size, tells us that the 2DModel-LOD should also be nominally inserted at CDB LOD = 3. Since both 2DModel-LODs have the same nominal CDB LOD, only one of them is retained (preferably the more detailed of the two)

Table A-9: T2DModel LOD versus Significant Size

T2DModel CDB Level	Significant Size SS_c (Coarsest Model-LOD)	Significant Size SS_{LOD} (Other Model-LODs) OTHER Interp. Max Error with respect to finest	Tile-LOD Size
-10	56 km < SS < 110 km	SS < 14 km	110 km
-9	28 km < SS ≤ 56 km	SS < 6.9 km	110 km
-8	14 km < SS ≤ 28 km	SS < 3.5 km	110 km
-7	6.9 km < SS ≤ 14 km	SS < 1.7 km	110 km
-6	3.4 km < SS ≤ 6.9 km	SS < 870 m	110 km
-5	1.7 km < SS ≤ 3.4 km	SS < 430 m	110 km
-4	860 m < SS ≤ 1.7 km	SS < 220 m	110 km
-3	430 m < SS ≤ 860 m	SS < 110 m	110 km
-2	220 m < SS ≤ 430 m	SS < 54 m	56 km
-1	110 m < SS ≤ 220 m	SS < 27 m	28 km

0	54 m < SS ≤ 110 m	SS < 13 m	14 km
1	27 m < SS ≤ 54 m	SS < 6.8 m	6.9 km
2	13 m < SS ≤ 27 m	SS < 3.4 m	3.4 km
3	6.7 m < SS ≤ 13 m	SS < 1.7 m	1.7 km
4	3.4 m < SS ≤ 6.7 m	SS < 840 mm	860 m
5	1.7 m < SS ≤ 3.4 m	SS < 420 mm	430 m
6	840 mm < SS ≤ 1.7 m	SS < 210 mm	220 m
7	420 mm < SS ≤ 840 mm	SS < 110 mm	110 m
8	210 mm < SS ≤ 420 mm	SS < 52 mm	54 m
9	110 mm < SS ≤ 210 mm	SS < 26 mm	27 m
10	52 mm < SS ≤ 110 mm	SS < 13 mm	13 m
11	26 mm < SS ≤ 52 mm	SS < 6.6 mm	6.7 m
12	13 mm < SS ≤ 26 mm	SS < 3.3 mm	3.4 m
13	6.7 mm < SS ≤ 13 mm	SS < 1.6 mm	1.7 m
14	3.4 mm < SS ≤ 6.7 mm	SS < 820 um	840 mm
15	1.7 mm < SS ≤ 3.4 mm	SS < 410 um	420 mm
16	820 um < SS ≤ 1.7 mm	SS < 210 um	210 mm
17	410 um < SS ≤ 820 um	SS < 100 um	110 mm
18	210 um < SS ≤ 410 um	SS < 51 um	52 mm
19	110 um < SS ≤ 210 um	SS < 26 um	26 mm
20	52 um < SS ≤ 110 um	SS < 13 um	13 mm
21	26 um < SS ≤ 52 um	SS < 6.7 um	6.7 mm
22	13 um < SS ≤ 26 um	SS < 3.4 um	3.4 mm

23	SS ≤ 13 um	SS < 1.7 um	1.7 mm

A.19.2.3 Rules Governing T2DModel LOD Hierarchy

Here is a summary of the rules required by the specification in order to ensure deterministic operation from client-devices:

1. Each feature may have multiple modeled representations at progressively coarser levels of detail. Each of the modeled representations is referred to as a 2DModel-LOD. In absence of pre-modeled coarser LOD representations, the tools may automatically generate coarser modeled levels-of-detail.
2. A 2DModel-LOD consists of a group of polygons that represent a feature at a specific level-of-detail; this group of polygons shares a common Feature Attribute Code (FACC), a Feature Sub-Code (FSC), a Model Name (MODL) and 2DModel-LOD's Significant Size SS'_{LOD}.
3. Each 2DModel has a distinct Significant Size value *SS'* based on its dimensions. In turn, each of the 2DModel-LODs of a 2DModel has a distinct Significant Size value SS'_{LOD} based on its modeled accuracy.
4. Insertion of a 2DModel-LOD into the T2DModel Dataset hierarchy proceeds as follows. Starting with LOD_{max} (LOD_{max} is a variable set by the user that sets the maximum depth of the LOD hierarchy) and progressing to coarser LODs…
   a. For each Tile-LOD, create a Model_List that is constructed from the 2DModel-LODs that straddle the Tile-LOD.
      i. If the 2DModel-LOD is not the coarsest LOD and its Significant Size is in accordance to Table A-9: T2DModel LOD versus Significant Size, then iteratively simplify the 2DModel-LOD (iterate until its Significant Size is no longer in accordance to Table A-9: T2DModel LOD versus Significant Size and keep results of previous iteration) and add it to the Tile-LOD. Only the coarser 2DModel-LODs of this 2DModel are available for future insertion into the T2DModel hierarchy.
      ii. If the 2DModel-LOD is the coarsest LOD of the 2DModel and its Significant Size is in accordance to Table A-9: T2DModel LOD versus Significant Size, insert it at this LOD of the hierarchy. If the 2DModel-LOD matches the Tile-LOD, remove it from the list for the processing of the coarser Tile-LOD.
   b. If the Model_List is less than *T2DModelFileSize*, no further processing is required.
   c. The Model_List of each Tile-LOD is sorted in decreasing order of Diff, where Diff is the difference between the Significant Size *SS* of the Model and the Significant Size as specified in Table 3.
   d. If the Model_List is greater than *T2DModelFileSize*, then (starting with the first entry in the sorted Model_List), Models are simplified one-by-one until the size of the Model_List is less than *T2DModelFileSize*. When a simplification occurs, the Model_List is re-sorted using the Diff value.

e. If a) the Model_List is deemed non-reducible and b) the Model_List is still greater than *T2DModelFileSize* …
   i. If LOD < LODmax, then…
      (1) a Temp_Model_List is created and initialized with the contents of the Model_list. Starting from the end of the Model_List, Models are removed one-by-one from the Model_list (starting with the first Model in the Model_List) and are copied into the Temp_Model_List until the Model_List reaches *T2DModelFileSize*.
      (2) The Temp_Model_List is merged to the children Tile-LODs and the children are re-processed using steps 4a to 4e. The process is iterative, i.e., the "overflow" is propagated into the finer LODs of the T2DModel hierarchy.
   ii. Else…
      (1) Models are removed one-by-one, starting with the first Model in the Model_List, until the Model_List is less than *T2DModelFileSize*. The corresponding T2DModels are removed from the CDB and a warning is issued stating that content was removed.

NOTE: The algorithm preserves the highest available modeled content while ensuring that the runtime constraint file size limits are respected. While the CDB data model allows for infinitely-sized 2DModel-LODs, a client-device may refuse to render the 2DModel-LOD if it has insufficient memory to load all of the OpenFlight files that make-up the 2DModel-LOD.

5. Each T2DModel Tile-LOD is subject to an OpenFlight file size limit of *T2DModelFileSize*, i.e., several OpenFlight files, each within the *T2DModelFileSize* file size limit, can be used to represent a very complex T2DModel Tile-LOD. Each of T2DModel-LODs of an T2DModel Tile-LOD share the same T2DModel-LOD Identifier (see rule 2)
6. Each Tile-LOD is subject to a file size limit of *T2DModelFileSize*.
7. All of the 2DModel-LODs in a T2DModel Tile-LOD are nominally exchange-LODs (see exception in next rule).
8. The depth of the T2DModel LOD hierarchy should be reduced by folding-in the Tile-Models_List of finer Tile-LODs as an additive LOD to the Tile-Model_List of a coarser Tile-LOD. Failure to perform this "compaction step" may result in significantly deeper T2DModel LOD hierarchy when the finest 2DModel-LODs consist of small details (e.g., thin stripes and markings on roads), and reduce the paging performance of client-devices.
9. The finer modeled representation of a T2DModel (i.e., a 2DModel-LOD with a smaller Significant Size) always appears in finer LODs of the Tile-LOD hierarchy than a coarser 2DModel-LOD.
10. A Tile-LOD cannot contain more than one 2DModel-LOD of the same T2DModel.

11. All T2DModels are clipped against the Tile-LOD boundaries.
12. Gaps in the LOD file hierarchy of the T2DModel Dataset are not permitted. This may result in Tile-LODs that are empty (e.g., without any T2DModels). The presence of an empty Tile-LOD file indicates the availability of content in T2DModel files located in finer LODs of the T2DModel hierarchy.

A.20 Guideline: Examples of Vector Dataset Usages

A.20.1 Lineal Feature Radar Simulation Example

The following diagram represents a typical usage of a lineal model in the CDB for a typical radar client-device.

The radar application first extracts the lineal feature from the CDB and constructs an object. The constructed object contains the necessary information for the radar to compute the equivalent radar image using the radar cross-section (RCS) of the lineal features with material attributes and directivity, etc.

NOTE: With the introduction of version 3.2 of the CDB Specification, it is recommended that the terrain-conformal features be modeled using T2DModels and that radar client-devices use this modeled representation instead of the vector lineal and areal features.

Figure A-31: Example of Lineal Features, illustrates three lineal features stored in the tile in Shape format. The junction nodes of each lineal feature represents the start and end junctions of the lineal feature. In this example, there is only one chain per lineal feature.

[8] The SGI format is fully supported by the CDB Specification 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.

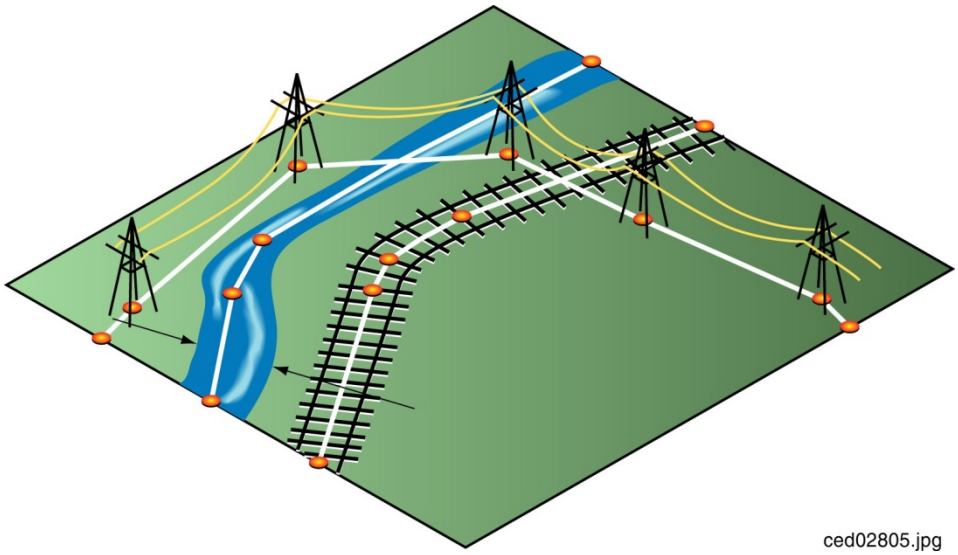


Figure A-31: Example of Lineal Features

The radar uses the position of the lineal coordinates to construct a line representation of the radar image. It extracts the lineal feature information from the chains to construct an internal local representation. The necessary information needed by radar is:

1. **Network Datasets:**
   The datasets along with the Feature Attribution Code (FACC) indicate if the feature is a road, a highway, or river for example. In the above illustration, we have a river, a powerline and a railway. The CDB Specification represents this in the *.dbf file of the Shapefile representation.
2. **Composite Material IndeX (CMIX):**
   The Composite Material IndeX attribute points to the Composite Material Table and provides the Radar the types of Base Materials that the feature is made of. This information is used, in addition to the geometry of the lineal feature or a generic RCS, to provide a radar signature of the target, which is proportional to the reflection value of the various materials. The intensity of the radar image represents the interaction of the simulated Radar Beam with the features in the synthetic environment. Each lineal contains a reference to a composite material which in turn is mapped to a reflectivity factor value in the radar simulation.
3. **Width (WGP):**
   The width of the lineal features is also taken into consideration. This information is part of the Shapefile data used to construct a 2D radar image of the terrain. The width information is encoded as an attribute of the lineal feature.
4. **Height (HGT):**
   The height of the lineal feature is used to indicate the height of each point/lineal with respect to the terrain height.

NOTE: The height value is a delta height above the terrain and is only provided for objects that require it such as the powerlines or the train tracks in this example.
The height property is especially valuable to radar client-devices because erect objects in the database produce significant returns and occultation areas in the displayed radar image. The height property can be assigned to the train tracks, long fences and the powerlines each with average altitudes.

5. **Position:**
This information is contained in the lineal *.shp files. The x and y coordinate of each point is extracted from those objects.

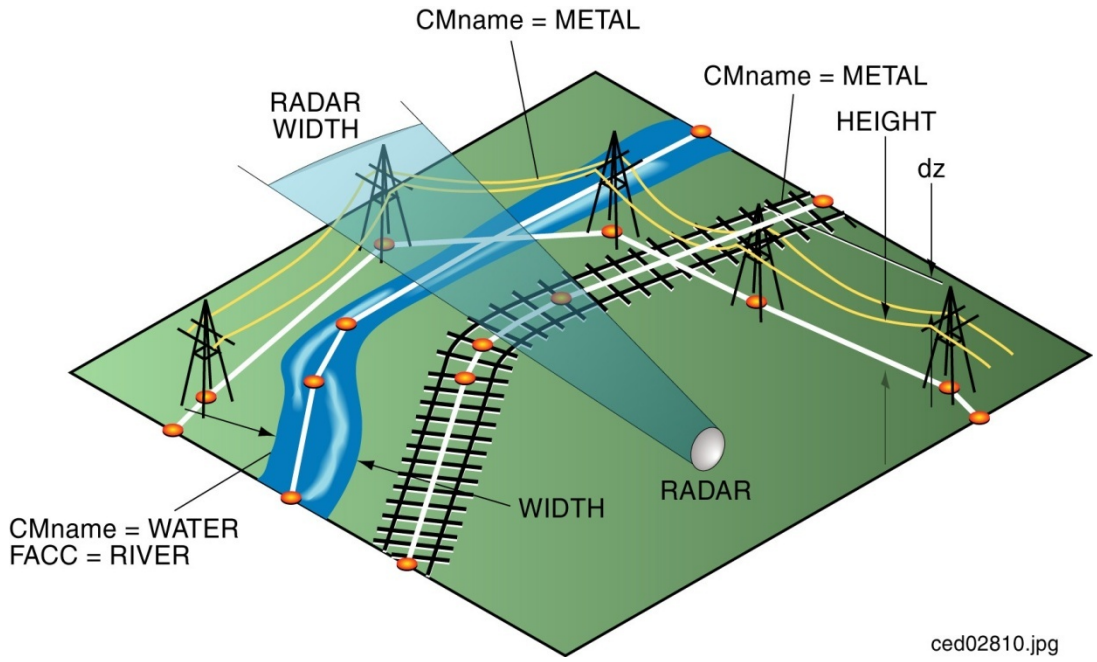


Figure A-32: Radar Beam Simulation

The radar then uses a beam simulation to process the above information and construct an image representing the content of each small beam sections. The intersection of the beam pie slice is compared with the lineal feature's position and converted into an image whose intensity is based on the computed RCS of the lineal features. As mentioned above, the RCS value (which is modeled internally in the radar simulation) takes into account the properties (which are derived from the attributes) of each lineal feature.

A.20.2 Road Following Example

Figure A-33: Network Dataset Used to Describe a Navigable Network, illustrates how lineal features can be used to describe a navigable network; the example could represent a network of roads. First, the application reads a lineal shape file describing the chains, then the point shape file describing the junction nodes. For each junction nodes the application makes up a list of attached chains ending up with a network as illustrated in Figure A-33: Network Dataset Used to Describe a Navigable Network, where there are six chains (labeled in this example as CSLID1 to

CSLID6) that are joined through intersection nodes (labeled in this example as CSZID1 to CSZID6). The small black dots represent points forming the segments of a chain; they are essentially used to describe the deviation from a straight line between nodes.

In Figure A-33: Network Dataset Used to Describe a Navigable Network, the green line shows an example of what a shortest path algorithm could determine if asked to find the shortest route between CSZID1 and CSZID5 based on the lengths of the chains. First, the algorithm would move to CSZID2 via the chain CSLID1; when at CSZID2 it has two alternatives, either take CSLID4 or use CSLID3 and CSLID6. In our example it would have determined that the latter alternative is the shortest path; the entity would then follow the path given by the green line going through all the segments in the chains.

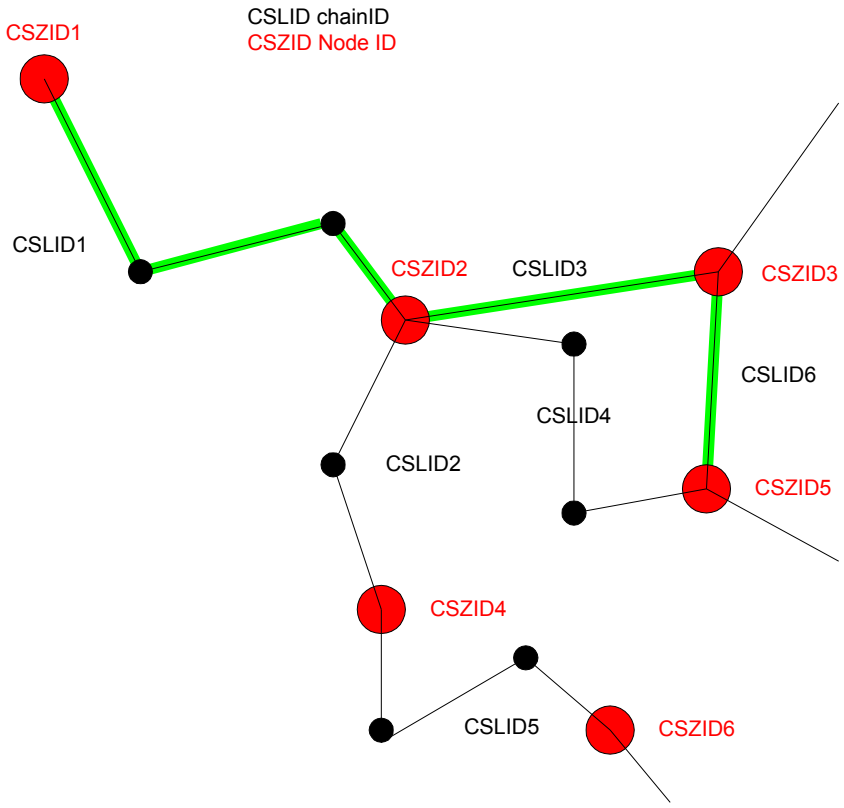


Figure A-33: Network Dataset Used to Describe a Navigable Network

A.20.3 Point Feature Radar Simulation Example

The following diagram represents a typical usage of a point feature as modeled in the CDB for a typical radar client-device. The radar client-device extracts the point feature from the database using the format described in the Specification and constructs an object. The constructed object will contain the necessary information for the radar to compute the equivalent radar image using the radar cross-section (RCS) of the object over the terrain; the RCS is derived from the point features characteristics.

NOTE: The example below illustrates the use of point-feature data by a radar client-device. However, we recommend that the radar client-devices use the modeled representation of the feature rather than the feature location, type and attribution data.

In Figure A-34: Objects Represented on a Terrain Tile, a series of different objects are represented on a terrain tile. The objects are modeled as single point of zero dimensions in radar. The radar will position the different points according to their geographic position and altitude. The height data corresponds to a height of the feature with respect to the terrain.

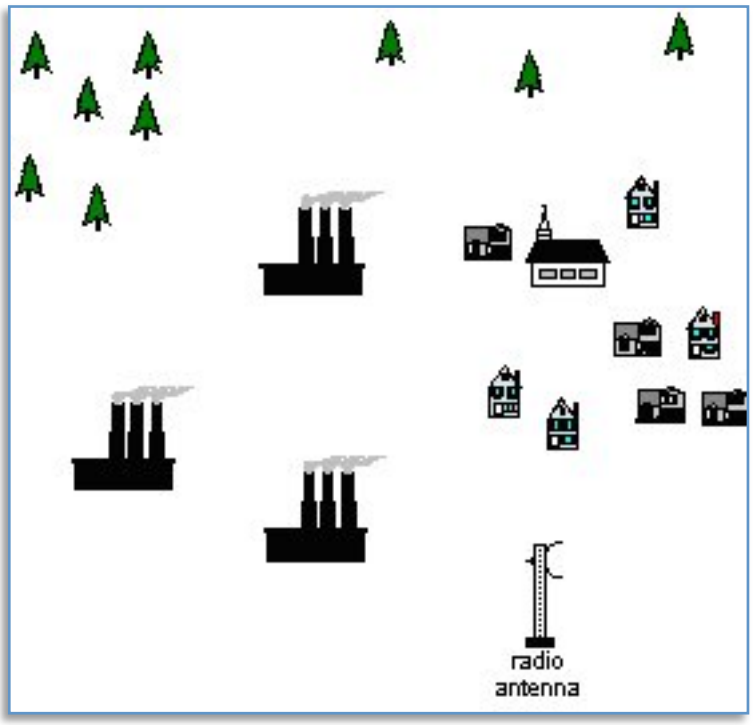


Figure A-34: Objects Represented on a Terrain Tile

The necessary information needed by radar is typically:

6. **Feature Attribution Code (FACC):**
   This information indicates if the feature is a tree, a pylon, or a church for example. In the above drawing this would mean a tree, an industry, a house or a radio station antenna.
7. **Composite Material IndeX (CMIX):**
   The Composite Material IndeX attribute points to the Composite Material Table and provides the Radar client-device with the type of material that the feature is made of. This information is used, in addition to the width

of the point feature, to provide a generic RCS of the target, which is proportional to the reflection value of the various materials. The RCS is then used by radar to determine the intensity of the radar image representing the point feature, based on the aspect and grazing angles to the Radar Beam. Each point contains a reference to a Composite Material.

8. **Bounding Sphere Radius (BSR):**
The radius of the point is also taken into consideration. This information is part of the shape data used to construct a 2D radar image of the terrain. The width is part of the point object attributes.

9. **Height (HGT):**
Since the radar sees the terrain with a perspective angle that can be computed using the radar altitude and the feature distance, the height of objects on the terrain becomes important to create the radar display image. This attribute of the point is used to indicate the differential height of each point with respect to the terrain height. In the example above, the trees all have the same average altitude. The other point features have different height.

10. **Position:**
The object point location in the CDB. The x and y coordinate of each point is extracted from those objects. The position when combined with the delta heights will create a pseudo-3D point feature object.

11. **Orientation (AO1):**
The radar needs the orientation of each point feature. This is needed because radar has a series of RCS tables, one for each of the Feature Identification Code. Those RCS tables give the RCS value for each incident angle of the radar beam. This angle is computed by taking into account the radar beam angle and the point feature orientation. For example, in the above drawing, the radio antenna has an orientation of 90 degrees. This means that if a radar beam comes from the right and points to the antenna at 270 degrees, the RCS value will be at maximum. The radar simulation would use a RCS table that represents the RCS with respect to the incident angle as follows:

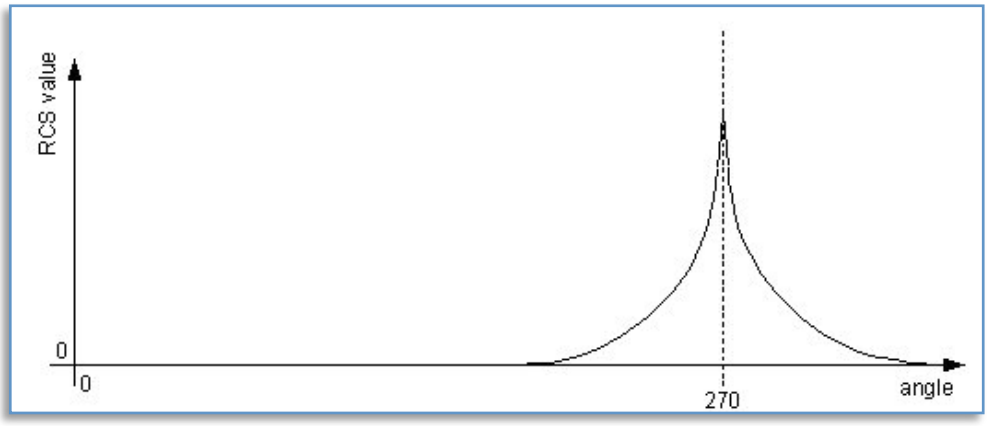


Figure A-35: Incident Angle

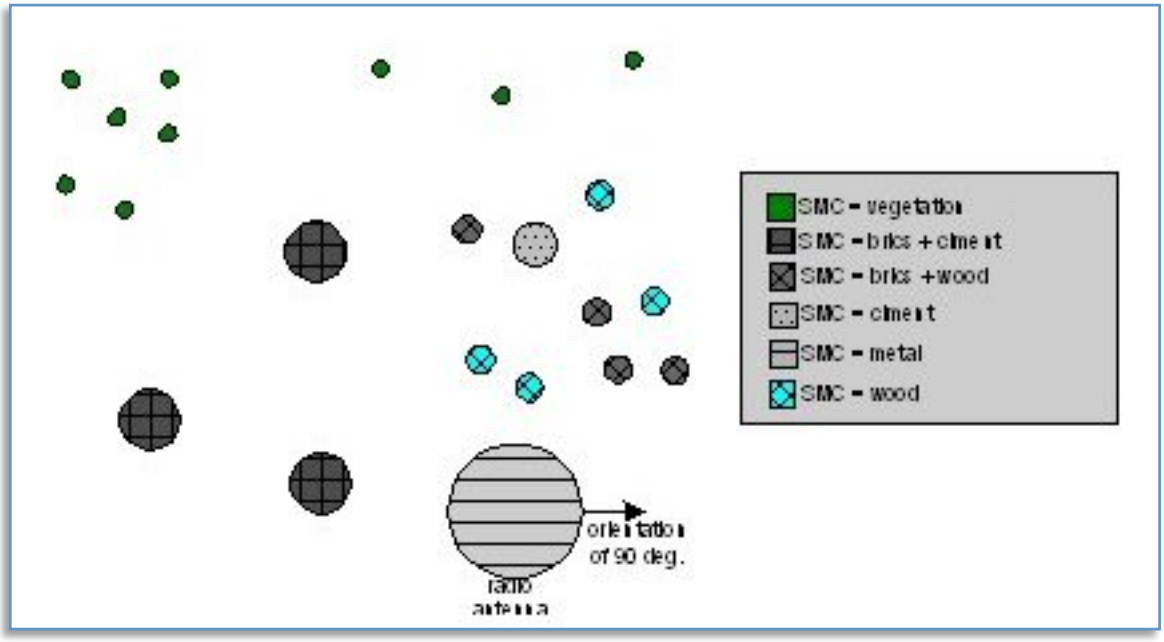


Figure A-36: Beam Simulation

The radar then uses a beam simulation to process the above information and construct an image representing the content of each small beam sections. The intersection of the beam pie slice is compared with the point's position and converted into an image whose intensity is based on the computed RCS of the points. As mentioned above, this RCS takes into account the attributes of each point feature.

If the size of the object referred to by the point feature is much larger than a specific threshold, the simulation could in addition use the MODL field of that point feature to extract a more precise geometrical 3D model from the CDB to increase the simulation fidelity.

A.20.4 Areal Feature Radar Simulation Example

The following diagram represents a typical usage of an areal feature as modeled in the CDB for a radar client-device.

NOTE: With the introduction of version 3.2 of the CDB Specification,we recommend that the terrain-conformal features be modeled using T2DModels and that radar client-devices use this modeled representation instead of the vector linear and areal features.

In a manner similar to the lineal example, the Radar extracts this areal (aka polygon or polyline) from the database using the Shape (*.shp) file and construct a tile in its memory. The constructed tile will contain the necessary information for the radar to compute the equivalent radar image using the radar cross-section (RCS) of the represented surface polygon over the terrain intersecting the Radar beam.

In Figure A-37: Four Areal Features Stored in the Tile, four areal features are stored in the tile with their surface material and feature classification attributes. Each of the features points to an array of segments.

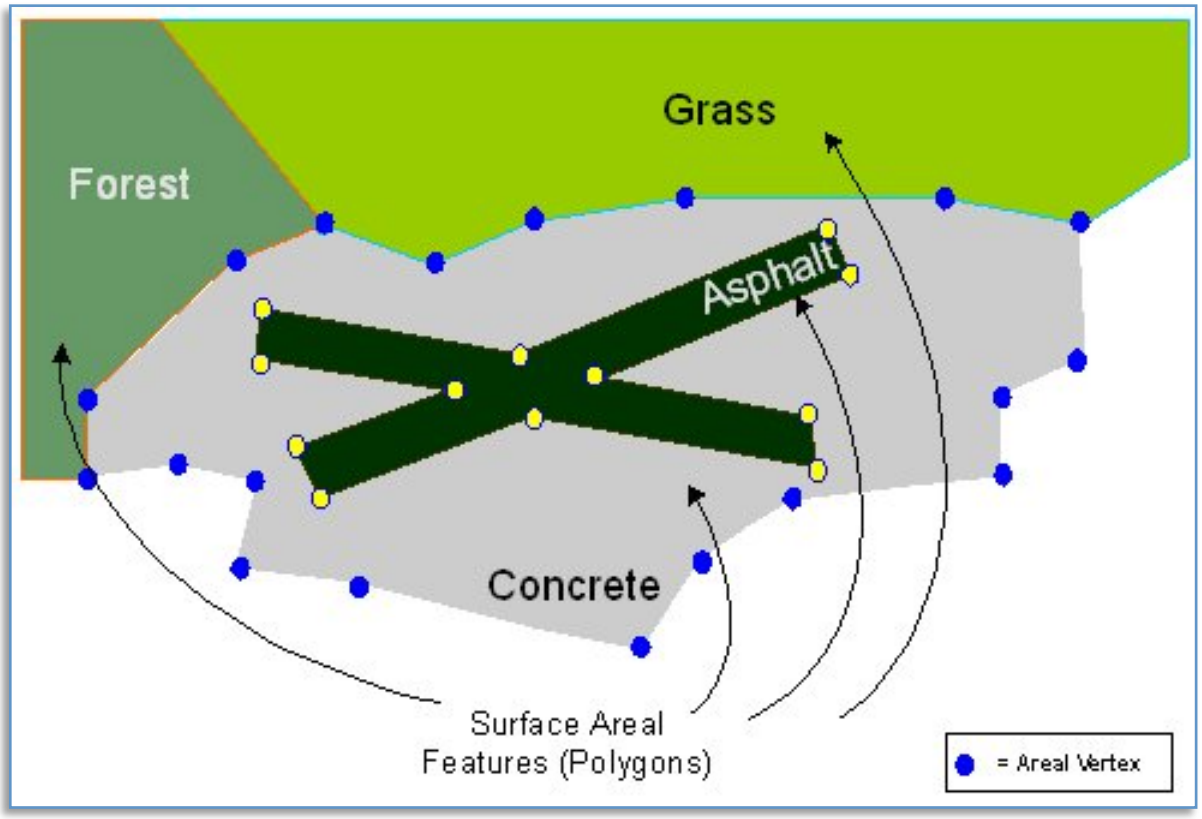


Figure A-37: Four Areal Features Stored in the Tile

The radar simulation uses the segment coordinates to construct a polygonal representation on the radar image. It extracts the areal feature attribute information from the segments and constructs its tile data in memory. The necessary information needed by the radar simulation is typically:

12. **Feature Classification Code (FACC):**
This information is the identification of the surface feature. It indicates if the areal feature represents a forest, a lake, or an airport runway for example. In the above drawing, this would translate to a forested area, a grassy area, a concrete section and a dual runway intersection.
13. **Composite Material IndeX (CMIX):**
The Composite Material IndeX Surface Material Code attribute points to the Composite Material Table and provides to the Radar with the type of material that the feature is made of. This information is used, in addition to the shape of the areal feature, to provide a generic RCS of the simulated area. This RCS is proportional to the reflection value of the various materials constituting the polygon or the simulated texture of its components (e.g., an industrial area made up of metallic roofs). The RCS

is then used by radar to determine the intensity of the radar image representing the areal feature, based on the aspect and grazing angles to the Radar Beam. Each chain will contain a reference to a material namespace object in the CDB.

14. **Height (HGT):**
Normally, the radar simulation sees the terrain with a perspective angle that can be computed using the radar altitude and the feature distance. Because of this angle, the height of objects on the terrain becomes important to create the image that the radar "sees". This attribute of the chain is used to indicate the average height of each areal object. In the example above, the forested area could be elevated to roughly 25 or 30 feet to produce a forest "canopy" which will look elevated to the radar.

The following image shows how a typical radar beam would intersect the different parts of areal features that are part of the terrain represented previously.

Figure A-38: Radar Beam Simulation, shows the radar using a beam simulation to extract the above information and construct an image representing the content of each small beam sections (aka bins). The intersection of the beam pie slice is performed against the areal feature polygons. Then the material of the polygon falling in the beam bin is converted into image intensity, which is relative to the computed RCS of that polygon's material. As mentioned above, this RCS takes into account the attributes of the segment of each areal feature.

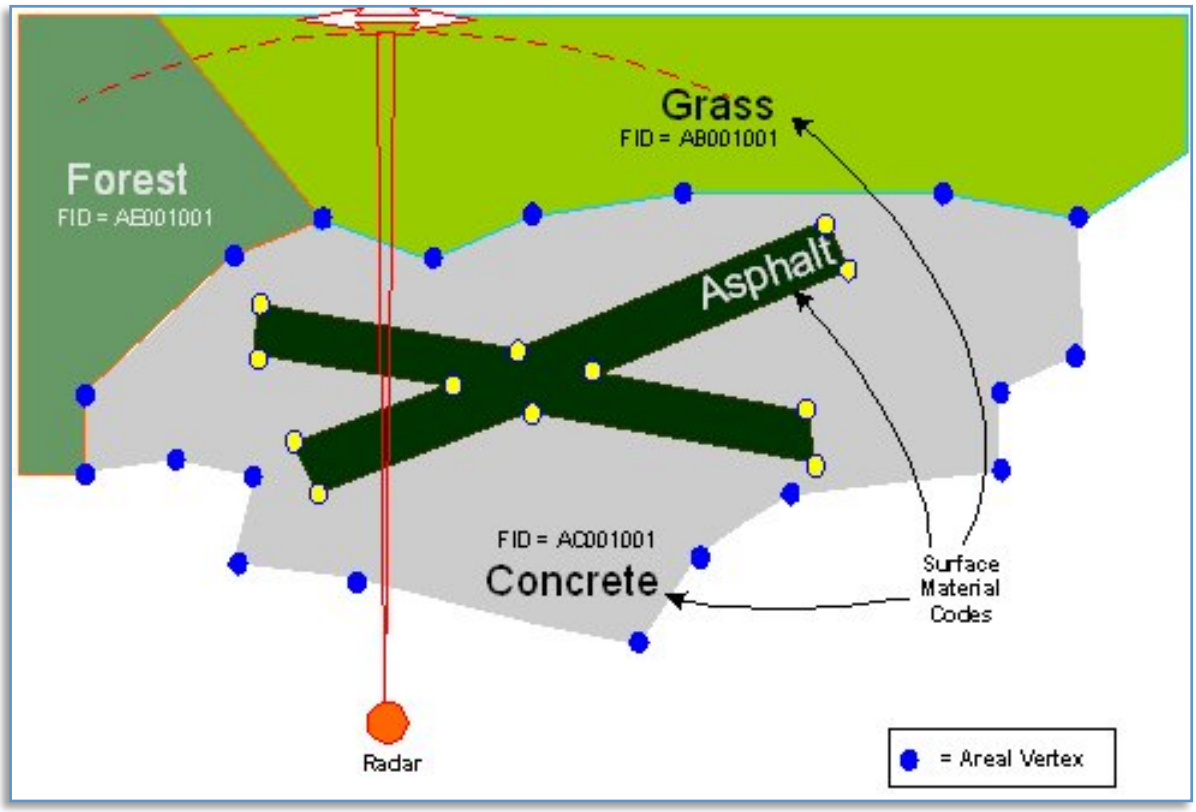


Figure A-38: Radar Beam Simulation

Appendix B

B. TIFF Specification 6.0 – Annotated

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

TIFF™

Revision 6.0

Final — June 3, 1992

Annotated with CDB conventions

Adobe Developers Association

Adobe Systems Incorporated
1585 Charleston Road
P.O. Box 7900
Mountain View, CA 94039-7900

E-Mail: devsup-person@adobe.com

A copy of this specification can be found in

http://www.adobe.com/Support/TechNotes.html

and

ftp://ftp.adobe.com/pub/adobe/DeveloperSupport/TechNotes/PDFfiles

Copyright

Licenses and Trademarks

PostScript is a trademark of Adobe Systems Incorporated. All instances of the name PostScript in the text are references to the PostScript language as defined by Adobe Systems Incorporated unless otherwise stated. The name PostScript also is used as a product trademark for Adobe Systems' implementation of the PostScript language interpreter.

Any references to a "PostScript printer," a "PostScript file," or a "PostScript driver" refer to printers, files, and driver programs (respectively) which are written in or support the PostScript language. The sentences in this specification that use "PostScript language" as an adjective phrase are so constructed to reinforce that the name refers to the standard language definition as set forth by Adobe Systems Incorporated.

PostScript, the PostScript logo, Display PostScript, Adobe, the Adobe logo, Adobe Illustrator, Aldus, PageMaker, TIFF, OPI, TrapWise, Tran-Script, Carta, and Sonata are trademarks of Adobe Systems Incorporated or its subsidiaries, and may be registered in some jurisdictions.

Apple, LaserWriter, and Macintosh are registered trademarks and Finder and System 7 are trademarks of Apple, Computer, Inc. Microsoft and MS-DOS are registered trademarks and Windows is a trademark of Microsoft Corporation. UNIX is a registered trademark of UNIX System Laboratories, Inc., a wholly owned subsidiary of Novell, Inc. All other trademarks are the property of their respective owners.

Production Notes

This document was created electronically using Adobe PageMaker® 6.0.

Contents

Introduction

About this Specification

This document describes TIFF, a tag-based file format for storing and interchanging raster images.

History

The first version of the TIFF specification was published by Aldus Corporation in the fall of 1986, after a series of meetings with various scanner manufacturers and software developers. It did not have a revision number but should have been labeled Revision 3.0 since there were two major earlier draft releases.

Revision 4.0 contained mostly minor enhancements and was released in April 1987. Revision 5.0, released in October 1988, added support for palette color images and LZW compression.

Scope

TIFF describes image data that typically comes from scanners, frame grabbers, and paint- and photo-retouching programs.

TIFF is not a printer language or page description language. The purpose of TIFF is to describe and store raster image data.

A primary goal of TIFF is to provide a rich environment within which applications can exchange image data. This richness is required to take advantage of the varying capabilities of scanners and other imaging devices.

Though TIFF is a rich format, it can easily be used for simple scanners and applications as well because the number of required fields is small.

TIFF will be enhanced on a continuing basis as new imaging needs arise. A high priority has been given to structuring TIFF so that future enhancements can be added without causing unnecessary hardship to developers.

Features

- TIFF is capable of describing bilevel, grayscale, palette-color, and full-color image data in several color spaces.
- TIFF includes a number of compression schemes that allow developers to choose the best space or time tradeoff for their applications.
- TIFF is not tied to specific scanners, printers, or computer display hardware.
- TIFF is portable. It does not favor particular operating systems, file systems, compilers, or processors.
- TIFF is designed to be extensible—to evolve gracefully as new needs arise.
- TIFF allows the inclusion of an unlimited amount of private or special-purpose information.

Revision Notes

Minor changes to TIFF 6.0, March 1995

Updated contact information and TIFF administration policies, since Aldus Corporation merged with Adobe Systems Incorporated on September 1, 1994.

The technical content and pagination are unchanged from the original June 3, 1992 release.

TIFF 5.0 to TIFF 6.0

This revision replaces TIFF Revision 5.0.

In the main body of the document, paragraphs that contain new or substantially-changed information are shown in italics.

New Features in Revision 6.0

Major enhancements to TIFF 6.0 are described in Part 2. They include:

- CMYK image definition
- A revised RGB Colorimetry section.
- YCbCr image definition
- CIE L*a*b* image definition
- Tiled image definition
- JPEG compression

Clarifications

- The LZW compression section more clearly explains when to switch the coding bit length.
- The interaction between Compression=2 (CCITT Huffman) and PhotometricInterpretation was clarified.
- The data organization of uncompressed data (Compression=1) when BitsPerSample is greater than 8 was clarified. See the Compression field description.
- The discussion of CCITT Group 3 and Group 4 bilevel image encodings was clarified and expanded, and Group3Options and Group4Options fields were renamed T4Options and T6Options. See Section 11.

Organizational Changes

- To make the organization more consistent and expandable, appendices were transformed into numbered sections.
- The document was divided into two parts—Baseline and Extensions—to help developers make better and more consistent implementation choices. Part 1, the Baseline section, describes those features that all general-purpose TIFF readers should support. Part 2, the Extensions section, describes a number of features that can be used by special or advanced applications.
- An index and table of contents were added.

Changes in Requirements

- To illustrate a Baseline TIFF file earlier in the document, the material from Appendix G ("TIFF Classes") in Revision 5 was integrated into the main body of the specification . As part of this integration, the TIFF Classes terminology was replaced by the more monolithic Baseline TIFF terminology. The intent was to further encourage all mainstream TIFF readers to support the Baseline TIFF requirements for bilevel, grayscale, RGB, and palette-color images.
- Due to licensing issues, LZW compression support was moved out of the "Part 1: Baseline TIFF" and into "Part 2: Extensions."
- Baseline TIFF requirements for bit depths in palette-color images were weakened a bit.

Changes in Terminology

In previous versions of the specification, the term "tag" reffered both to the identifying number of a TIFF field and to the entire field. In this version, the term "tag" refers only to the identifying number. The term "field" refers to the entire field, including the value.

Compatibility

Every attempt has been made to add functionality in such a way as to minimize compatibility problems with files and software that were based on earlier versions of the TIFF specification. The goal is that TIFF files should never become obsolete and that TIFF software should not have to be revised more frequently than absolutely necessary. In particular, Baseline TIFF 6.0 files will generally be readable even by older applications that assume TIFF 5.0 or an earlier version of the specification.

However, TIFF 6.0 files that use one of the major new extensions, such as a new compression scheme or color space, will not be successfully read by older software. In such cases, the older applications must gracefully give up and refuse to import the image, providing the user with a reasonably informative message.

TIFF Administration

Information and Support

The most recent version of the TIFF specification is available in PDF format on the Adobe WWW and ftp servers See the cover page of the specification for the required addresses.

Because of the widespread use of TIFF for in many environments, Adobe is unable to provide a general consulting service for TIFF implementors. TIFF developers are encouraged to study sample TIFF files, read TIFF documentation thoroughly, and work with developers of other products that are important to you.

If your TIFF question specifically concerns compatibility with an Adobe Systems product, please contact Adobe Developer Support at devsup-person@adobe.com.

Most companies that use TIFF can answer questions about support for TIFF in their products. Contact the appropriate product manager or developer support service group.

Private Fields and Values

An organization might wish to store information meaningful to only that organization in a TIFF file. Tags numbered 32768 or higher, sometimes called private tags, are reserved for that purpose.

Upon request, the TIFF administrator (send email to devsup-person@adobe.com) will allocate and register one or more private tags for an organization, to avoid possible conflicts with other organizations. You do not need to tell the TIFF administrator what you plan to use them for, but giving us this information may help other developers to avoid some duplication of effort. We will likely make the tag database public at some point.

Private enumerated values can be accommodated in a similar fashion. For example, you may wish to experiment with a new compression scheme within TIFF. Enumeration constants numbered 32768 or higher are reserved for private usage. Upon request, the administrator will allocate and register one or more enumerated values for a particular field (Compression, in our example), to avoid possible conflicts.

Tags and values allocated in the private number range are not prohibited from being included in a future revision of this specification. Several such instances exist in the current TIFF specification.

Do not choose your own tag numbers. Doing so could cause serious compatibility problems in the future. However, if there is little or no chance that your TIFF files will escape your private environment, please consider using TIFF tags in the "reusable" 65000-65535 range. You do not need to contact Adobe when using numbers in this range.

If you need more than 10 tags, we suggest that you reserve a single private tag, define it as a LONG TIFF data type, and use its value as a pointer (offset) to a private IFD or other data structure of your choosing. Within that IFD, you can use whatever tags you want, since no one else will know that it is an IFD unless you tell them.

Submitting a Proposal

Any person or group that wants to propose a change or addition to the TIFF specification should prepare a proposal that includes the following information:

- Name of the person or group making the request, and your affiliation.
- The reason for the request.
- A list of changes exactly as you propose that they appear in the specification. Use inserts, callouts, or other obvious editorial techniques to indicate areas of change, and number each change.
- Discussion of the potential impact on the installed base.
- A list of contacts outside your company that support your position. Include their affiliation.

Please send your proposal to devsup-person@adobe.com.

The TIFF Advisory Committee

The TIFF Advisory Committee is a working group of TIFF experts from a number of hardware and software manufacturers. It was formed in the spring of 1991 to provide a forum for debating and refining proposals for the 6.0 release of the TIFF specification.

If you are a TIFF expert and think you have the time and interest to work on this committee, contact devsup-person@adobe.com for further information. For the TIFF 6.0 release, the group met every two or three months, usually on the west coast of the U.S. Accessibility via Internet email is a requirement for membership, since that has proven to be an invaluable means for getting work done between meetings.

Other TIFF Extensions

~~The Aldus TIFF sections on CompuServe and AppleLink~~ (new location is under construction; check the Adobe WWW home page (http://www.adobe.com) for future developements) will contain proposed TIFF extensions from other companies that are not approved by Adobe as part of Baseline TIFF.

These proposals typically represent specialized uses of TIFF that do not fall within the domain of publishing or general graphics or picture interchange. Generally, these features will not be widely supported. If you do write files that incorporate these extensions, be sure to either not call them TIFF files or mark them in some way so that they will not be confused with mainstream TIFF files.

If you have such a document, send it to devsup-person@adobe.com. All submissions must be PDF documents or simple text. Be sure to include contact information—at least an email address.

Part 1: Baseline TIFF

The TIFF specification is divided into two parts. Part 1 describes *Baseline TIFF*. Baseline TIFF is the core of TIFF, the essentials that all mainstream TIFF developers should support in their products.

Section 1: Notation

Decimal and Hexadecimal

Unless otherwise noted, all numeric values in this document are expressed in decimal. (".H" is appended to hexidecimal values.)

Compliance

Is and *shall* indicate mandatory requirements. All compliant writers and readers must meet the specification.

Should indicates a recommendation.

May indicates an option.

Features designated 'not recommended for general data interchange' are considered extensions to Baseline TIFF. Files that use such features shall be designated "Extended TIFF 6.0" files, and the particular extensions used should be documented. A Baseline TIFF 6.0 reader is not required to support any extensions.

Section 2: TIFF Structure

CDB-compliant TIFF readers do not consider TIFF image and DEM data in big-endian byte order.

TIFF is an image file format. In this document, a *file* is defined to be a sequence of 8-bit bytes, where the bytes are numbered from 0 to N. The largest possible TIFF file is 2**32 bytes in length.

A TIFF file begins with an 8-byte *image file header* that points to an *image file directory* (*IFD*). An image file directory contains information about the image, as well as pointers to the actual image data.

The following paragraphs describe the image file header and IFD in more detail.

See Figure 1.

Image File Header

A TIFF file begins with an 8-byte image file header, containing the following information:

Bytes 0-1: The byte order used within the file. Legal values are:

"II" (4949.H)

"MM" (4D4D.H)

In the "II" format, byte order is always from the least significant byte to the most significant byte, for both 16-bit and 32-bit integers This is called *little-endian* byte order. In the "MM" format, byte order is always from most significant to least significant, for both 16-bit and 32-bit integers. This is called *big-endian* byte order.

Bytes 2-3 An arbitrary but carefully chosen number (42) that further identifies the file as a TIFF file.

The byte order depends on the value of Bytes 0-1.

Bytes 4-7 The offset (in bytes) of the first IFD. The directory may be at any location in the file after the header but *must begin on a word boundary*. In particular, an Image File Directory may follow the image data it describes. Readers must follow the pointers wherever they may lead.

The term *byte offset* is always used in this document to refer to a location with respect to the beginning of the TIFF file. The first byte of the file has an offset of 0.

Figure 1

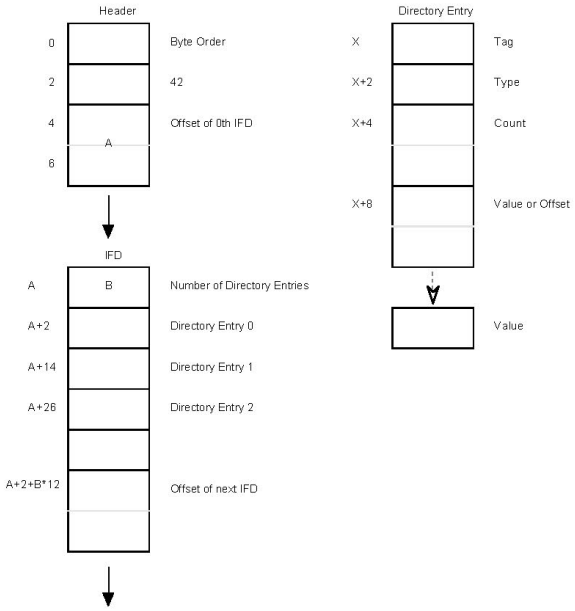


Image File Directory

An *Image File Directory* (*IFD*) consists of a 2-byte count of the number of directory entries (i.e., the number of fields), followed by a sequence of 12-byte field entries, followed by a 4-byte offset of the next IFD (or 0 if none). (Do not forget to write the 4 bytes of 0 after the last IFD.)

There must be at least 1 IFD in a TIFF file and each IFD must have at least one entry.

See Figure 1.

IFD Entry

Each 12-byte IFD entry has the following format:

Bytes 0-1 The Tag that identifies the field.

Bytes 2-3 The field Type.

Bytes 4-7 The number of values, *Count* of the indicated Type.

Bytes 8-11 The Value Offset, the file offset (in bytes) of the Value for the field. The Value is expected to begin on a word boundary; the corresponding Value Offset will thus be an even number. This file offset may point anywhere in the file, even after the image data.

IFD Terminology

A *TIFF field* is a logical entity consisting of TIFF tag and its value. This logical concept is implemented as an *IFD Entry*, plus the actual value if it doesn't fit into the value/offset part, the last 4 bytes of the IFD Entry. The terms *TIFF field* and *IFD entry* are interchangeable in most contexts.

Sort Order

The entries in an IFD must be sorted in ascending order by Tag. Note that this is not the order in which the fields are described in this document. The Values to which directory entries point need not be in any particular order in the file.

Value/Offset

To save time and space the Value Offset contains the Value instead of pointing to the Value if and only if the Value fits into 4 bytes. If the Value is shorter than 4 bytes, it is left-justified within the 4-byte Value Offset, i.e., stored in the lower-numbered bytes. Whether the Value fits within 4 bytes is determined by the Type and Count of the field.

Count

Count—called *Length* in previous versions of the specification—is the number of values. Note that Count is not the total number of bytes. For example, a single 16-bit word (SHORT) has a Count of 1; not 2.

Types

The field types and their sizes are:

1 = BYTE	8-bit unsigned integer.
2 = ASCII	8-bit byte that contains a 7-bit ASCII code; the last byte must be NUL (binary zero).
3 = SHORT	16-bit (2-byte) unsigned integer.
4 = LONG	32-bit (4-byte) unsigned integer.
5 = RATIONAL	Two LONGs: the first represents the numerator of a fraction; the second, the denominator.

The value of the Count part of an ASCII field entry includes the NUL. If padding is necessary, the Count does not include the pad byte. Note that there is no initial "count byte" as in Pascal-style strings.

Any ASCII field can contain multiple strings, each terminated with a NUL. A single string is preferred whenever possible. The Count for multi-string fields is the number of bytes in all the strings in that field plus their terminating NUL bytes. Only one NUL is allowed between strings, so that the strings following the first string will often begin on an odd byte.

The reader must check the type to verify that it contains an expected value. TIFF currently allows more than 1 valid type for some fields. For example, ImageWidth and ImageLength are usually specified as having type SHORT. But images with more than 64K rows or columns must use the LONG field type.

TIFF readers should accept BYTE, SHORT, or LONG values for any unsigned integer field. This allows a single procedure to retrieve any integer value, makes reading more robust, and saves disk space in some situations.

In TIFF 6.0, some new field types have been defined:

6 = SBYTE	An 8-bit signed (twos-complement) integer.
7 = UNDEFINED	An 8-bit byte that may contain anything, depending on the definition of the field.
8 = SSHORT	A 16-bit (2-byte) signed (twos-complement) integer.
9 = SLONG	A 32-bit (4-byte) signed (twos-complement) integer.
10 = SRATIONAL	Two SLONG's: the first represents the numerator of a fraction, the second the denominator.
11 = FLOAT	Single precision (4-byte) IEEE format.
12 = DOUBLE	Double precision (8-byte) IEEE format.

These new field types are also governed by the byte order (II or MM) in the TIFF header.

Warning: It is possible that other TIFF field types will be added in the future. Readers should skip over fields containing an unexpected field type.

Fields are arrays

Each TIFF field has an associated Count. This means that all fields are actually one-dimensional arrays, even though most fields contain only a single value.

For example, to store a complicated data structure in a single private field, use the UNDEFINED field type and set the Count to the number of bytes required to hold the data structure.

Multiple Images per TIFF File

TIFF/CDB Readers: **The CDB Specification does not take advantage of multiple images per TIFF file.**

There may be more than one IFD in a TIFF file. Each IFD defines a *subfile*. One potential use of subfiles is to describe related images, such as the pages of a fac-simile transmission. A Baseline TIFF reader is not required to read any IFDs beyond the first one.

Section 3: Bilevel Images

CDB-compliant TIFF readers do not consider bi-level image data.

Now that the overall TIFF structure has been described, we can move on to filling the structure with actual fields (tags and values) that describe raster image data.

To make all of this clearer, the discussion will be organized according to the four Baseline TIFF image types: bilevel, grayscale, palette-color, and full-color images. This section describes bilevel images.

Fields required to describe bilevel images are introduced and described briefly here. Full descriptions of each field can be found in Section 8.

Color

CDB-compliant TIFF readers do not consider WhiteIsZero image data.

A bilevel image contains two colors—black and white. TIFF allows an application to write out bilevel data in either a white-is-zero or black-is-zero format. The field that records this information is called PhotometricInterpretation.

PhotometricInterpretation

Tag = 262 (106.H)

Type = SHORT

Values:

0 = WhiteIsZero. For bilevel and grayscale images: 0 is imaged as white. The maximum value is imaged as black. This is the normal value for Compression=2.

1 = BlackIsZero. For bilevel and grayscale images: 0 is imaged as black. The maximum value is imaged as white. If this value is specified for Compression=2, the image should display and print reversed.

Compression

CDB-compliant TIFF readers do not consider compressed TIFF image.

Data can be stored either compressed or uncompressed.

Compression

Tag = 259 (103.H)

Type = SHORT

Values:

1 = No compression, but pack data into bytes as tightly as possible, leaving no unused bits (except at the end of a row). The component values are stored as an array of type BYTE. Each scan line (row) is padded to the next BYTE boundary.

2 = CCITT Group 3 1-Dimensional Modified Huffman run length encoding. See

Section 10 for a description of Modified Huffman Compression.

32773 = PackBits compression, a simple byte-oriented run length scheme. See the PackBits section for details.

Data compression applies only to raster image data. All other TIFF fields are unaffected.

Baseline TIFF readers must handle all three compression schemes.

Rows and Columns

The CDB specification recommends that the product of ImageLength and ImageWidth be less than 4,194,304 (2K x 2K).

CDB-compliant TIFF readers require this field to be a power of 2. ImageLength need not be equal to ImageWidth.

An image is organized as a rectangular array of pixels. The dimensions of this array are stored in the following fields:

ImageLength

Tag = 257 (101.H)

Type = SHORT or LONG

The number of rows (sometimes described as *scanlines*) in the image.

CDB-compliant TIFF readers require this field to be a power of 2. ImageWidth need not be equal to ImageLength.

ImageWidth

Tag = 256 (100.H)

Type = SHORT or LONG

The number of columns in the image, i.e., the number of pixels per scanline.

Physical Dimensions

Applications often want to know the size of the picture represented by an image. This information can be calculated from ImageWidth and ImageLength given the following resolution data:

CDB-compliant TIFF readers do not consider this TIFF tag.

ResolutionUnit

Tag = 296 (128.H)

Type = SHORT

Values:

1 = No absolute unit of measurement. Used for images that may have a non-square aspect ratio but no meaningful absolute dimensions.

2 = Inch.

3 = Centimeter.

Default = 2 (inch).

CDB-compliant TIFF readers do not consider this TIFF tag.

XResolution

Tag = 282 (11A.H)

Type = RATIONAL

The number of pixels per ResolutionUnit in the ImageWidth (typically, horizontal - see Orientation) direction.

CDB-compliant TIFF readers do not consider this TIFF tag.

YResolution

Tag = 283 (11B.H)

Type = RATIONAL

The number of pixels per ResolutionUnit in the ImageLength (typically, vertical) direction.

Location of the Data

Compressed or uncompressed image data can be stored almost anywhere in a TIFF file. TIFF also supports breaking an image into separate strips for increased editing flexibility and efficient I/O buffering. The location and size of each strip is given by the following fields:

RowsPerStrip

Tag = 278 (116.H)

Type = SHORT or LONG

The number of rows in each strip (except possibly the last strip.)

For example, if ImageLength is 24, and RowsPerStrip is 10, then there are 3 strips, with 10 rows in the first strip, 10 rows in the second strip, and 4 rows in the third strip. (The data in the last strip is not padded with 6 extra rows of dummy data.)

StripOffsets

Tag = 273 (111.H)

Type = SHORT or LONG

For each strip, the byte offset of that strip.

StripByteCounts

Tag = 279 (117.H)

Type = SHORT or LONG

For each strip, the number of bytes in that strip *after any compression*.

Putting it all together (along with a couple of less-important fields that are discussed later), a sample bilevel image file might contain the following fields:

A Sample Bilevel TIFF File

Offset (hex)	Description	Value (numeric values are expressed in hexadecimal notation)		
Header:				
0000	Byte Order	4D4D		
0002	42	002A		
0004	1st IFD offset	00000014		
IFD:				
0014	Number of Directory Entries	000C		
0016	NewSubfileType	00FE	0004	00000001 00000000
0022	ImageWidth	0100	0004	00000001 000007D0
002E	ImageLength	0101	0004	00000001 00000BB8
003A	Compression	0103	0003	00000001 8005 0000
0046	PhotometricInterpretation	0106	0003	00000001 0001 0000
0052	StripOffsets	0111	0004	000000BC 000000B6
005E	RowsPerStrip	0116	0004	00000001 00000010
006A	StripByteCounts	0117	0003	000000BC 000003A6
0076	XResolution	011A	0005	00000001 00000696
0082	YResolution	011B	0005	00000001 0000069E
008E	Software	0131	0002	0000000E 000006A6
009A	DateTime	0132	0002	00000014 000006B6
00A6	Next IFD offset	00000000		
Values longer than 4 bytes:				
00B6	StripOffsets	Offset0, Offset1, ... Offset187		
03A6	StripByteCounts	Count0, Count1, ... Count187		
0696	XResolution	0000012C 00000001		
069E	YResolution	0000012C 00000001		
06A6	Software	"PageMaker 4.0"		
06B6	DateTime	"1988:02:18 13:59:59"		
Image Data:				
00000700		Compressed data for strip 10		
xxxxxxxx		Compressed data for strip 179		
xxxxxxxx		Compressed data for strip 53		
xxxxxxxx		Compressed data for strip 160		
.				
.				

End of example

Comments on the Bilevel Image Example

- The IFD in this example starts at 14h. It could have started anywhere in the file providing the offset was an even number greater than or equal to 8 (since the TIFF header is always the first 8 bytes of a TIFF file).
- With 16 rows per strip, there are 188 strips in all.
- The example uses a number of optional fields such as DateTime. TIFF readers must safely skip over these fields if they do not understand or do not wish to use the information. Baseline TIFF readers must not require that such fields be present.
- To make a point, this example has highly-fragmented image data. The strips of the image are not in sequential order. The point of this example is to illustrate that strip offsets must not be ignored. Never assume that strip N+1 follows strip N on disk. It is not required that the image data follow the IFD information.

Required Fields for Bilevel Images

Here is a list of required fields for Baseline TIFF bilevel images. The fields are listed in numerical order, as they would appear in the IFD. Note that the previous example omits some of these fields. This is permitted because the fields that were omitted each have a default and the default is appropriate for this file.

TagName	Decimal	Hex	Type	Value
ImageWidth	256	100	SHORT or LONG	
ImageLength	257	101	SHORT or LONG	
Compression	259	103	SHORT	1, 2 or 32773
PhotometricInterpretation	262	106	SHORT	0 or 1
StripOffsets	273	111	SHORT or LONG	
RowsPerStrip	278	116	SHORT or LONG	
StripByteCounts	279	117	LONG or SHORT	
XResolution	282	11A	RATIONAL	
YResolution	283	11B	RATIONAL	
ResolutionUnit	296	128	SHORT	1, 2 or 3

Baseline TIFF bilevel images were called TIFF Class B images in earlier versions of the TIFF specification.

Section 4: Grayscale Images

Grayscale images are a generalization of bilevel images. Bilevel images can store only black and white image data, but grayscale images can also store shades of gray.

To describe such images, you must add or change the following fields. The other required fields are the same as those required for bilevel images.

Differences from Bilevel Images

CDB-compliant TIFF readers do not consider compressed TIFF image.

Compression = 1 *or 32773 (PackBits).* In Baseline TIFF, grayscale images can either be stored as uncompressed data or compressed with the PackBits algorithm.

Caution: PackBits is often ineffective on continuous tone images, including many grayscale images. In such cases, it is better to leave the image uncompressed.

BitsPerSample

Tag = 258 (102.H)

Type = SHORT

The number of bits per component.

Allowable values for Baseline TIFF grayscale images are **4** and **8**, allowing either 16 or 256 distinct shades of gray.

Required Fields for Grayscale Images

These are the required fields for grayscale images (in numerical order):

TagName	Decimal	Hex	Type	Value
ImageWidth	256	100	SHORT or LONG	
ImageLength	257	101	SHORT or LONG	
BitsPerSample	258	102	SHORT	4 or 8
Compression	259	103	SHORT	1 or 32773
PhotometricInterpretation	262	106	SHORT	0 or 1
StripOffsets	273	111	SHORT or LONG	
RowsPerStrip	278	116	SHORT or LONG	
StripByteCounts	279	117	LONG or SHORT	
XResolution	282	11A	RATIONAL	
YResolution	283	11B	RATIONAL	
ResolutionUnit	296	128	SHORT	1 or 2 or 3

Baseline TIFF grayscale images were called TIFF Class G images in earlier versions of the TIFF specification.

Section 5: Palette-color Images

CDB-compliant TIFF readers do not consider palette-color images, i.e. PhotometricInterpretation = 3 (Palette Color).

Palette-color images are similar to grayscale images. They still have one component per pixel, but the component value is used as an index into a full RGB-lookup table. To describe such images, you need to add or change the following fields. The other required fields are the same as those for grayscale images.

Differences from Grayscale Images

PhotometricInterpretation = 3 (Palette Color).

CDB-compliant TIFF readers do not consider palette-color images; as a result, this tag is ignored.

ColorMap

Tag = 320 (140.H)

Type = SHORT

N = 3 * (2**BitsPerSample)

This field defines a Red-Green-Blue color map (often called a lookup table) for palette color images. In a palette-color image, a pixel value is used to index into an RGB-lookup table. For example, a palette-color pixel having a value of 0 would be displayed according to the 0th Red, Green, Blue triplet.

In a TIFF ColorMap, all the Red values come first, followed by the Green values, then the Blue values. In the ColorMap, black is represented by 0,0,0 and white is represented by 65535, 65535, 65535.

Required Fields for Palette Color Images

These are the required fields for palette-color images (in numerical order):

TagName	Decimal	Hex	Type	Value
ImageWidth	256	100	SHORT or LONG	
ImageLength	257	101	SHORT or LONG	
BitsPerSample	258	102	SHORT	4 or 8
Compression	259	103	SHORT	1 or 32773
PhotometricInterpretation	262	106	SHORT	3
StripOffsets	273	111	SHORT or LONG	
RowsPerStrip	278	116	SHORT or LONG	
StripByteCounts	279	117	LONG or SHORT	
XResolution	282	11A	RATIONAL	
YResolution	283	11B	RATIONAL	
ResolutionUnit	296	128	SHORT	1 or 2 or 3
ColorMap	320	140	SHORT	

Baseline TIFF palette-color images were called TIFF Class P images in earlier versions of the TIFF specification.

Section 6: RGB Full Color Images

In an RGB image, each pixel is made up of three components: red, green, and blue. There is no ColorMap.

To describe an RGB image, you need to add or change the following fields and values. The other required fields are the same as those required for palette-color images.

Differences from Palette Color Images

BitsPerSample = 8,8,8. Each component is 8 bits deep in a Baseline TIFF RGB image.

PhotometricInterpretation = 2 (RGB).

There is no **ColorMap**.

SamplesPerPixel

Tag = 277 (115.H)

Type = SHORT

The number of components per pixel. This number is 3 for RGB images, unless extra samples are present. See the ExtraSamples field for further information.

Required Fields for RGB Images

These are the required fields for RGB images (in numerical order):

TagName	Decimal	Hex	Type	Value
ImageWidth	256	100	SHORT or LONG	
ImageLength	257	101	SHORT or LONG	
BitsPerSample	258	102	SHORT	8,8,8
Compression	259	103	SHORT	1 or 32773
PhotometricInterpretation	262	106	SHORT	2
StripOffsets	273	111	SHORT or LONG	
SamplesPerPixel	277	115	SHORT	3 or more
RowsPerStrip	278	116	SHORT or LONG	
StripByteCounts	279	117	LONG or SHORT	
XResolution	282	11A	RATIONAL	
YResolution	283	11B	RATIONAL	
ResolutionUnit	296	128	SHORT	1, 2 or 3

The BitsPerSample values listed above apply only to the main image data. If ExtraSamples are present, the appropriate BitsPerSample values for those samples must also be included.

Baseline TIFF RGB images were called TIFF Class R images in earlier versions of the TIFF specification.

Section 7: Additional Baseline TIFF Requirements

This section describes characteristics required of all Baseline TIFF files.

General Requirements

Options. Where there are options, TIFF writers can use whichever they want. Baseline TIFF readers must be able to handle all of them.

Defaults. TIFF writers may, but are not required to, write out a field that has a default value, if the default value is the one desired. TIFF readers must be prepared to handle either situation.

CDB-compliant TIFF readers consider only type 'II', (little-endian) byte ordered data.

Other fields. TIFF readers must be prepared to encounter fields other than those required in TIFF files. TIFF writers are allowed to write optional fields such as Make, Model, and DateTime, and TIFF readers may use such fields if they exist. TIFF readers must not, however, refuse to read the file if such optional fields do not exist. *TIFF readers must also be prepared to encounter and ignore private fields not described in the TIFF specification.*

'MM' and 'II' byte order. TIFF readers must be able to handle both byte orders. TIFF writers can do whichever is most convenient or efficient.

The CDB specification does not take advantage of multiple images per file.

Multiple subfiles. TIFF readers must be prepared for multiple images (subfiles) per TIFF file, although they are not required to do anything with images after the first one. TIFF writers are required to write a long word of 0 after the last IFD (to signal that this is the last IFD), as described earlier in this specification.

If multiple subfiles are written, the first one must be the full-resolution image. Subsequent images, such as reduced-resolution images, may be in any order in the TIFF file. If a reader wants to use such images, it must scan the corresponding IFD's before deciding how to proceed.

TIFF Editors. Editors—applications that modify TIFF files—have a few additional requirements:

- TIFF editors must be especially careful about subfiles. If a TIFF editor edits a full-resolution subfile, but does not update an accompanying reduced-resolution subfile, a reader that uses the reduced-resolution subfile for screen display will display the wrong thing. So TIFF editors must either create a new reduced-resolution subfile when they alter a full-resolution subfile or they must delete any subfiles that they aren't prepared to deal with.
- A similar situation arises with the fields in an IFD. It is unnecessary—and possibly dangerous—for an editor to copy fields it does not understand because the editor might alter the file in a way that is incompatible with the unknown fields.

No Duplicate Pointers. *No data should be referenced from more than one place. TIFF readers and editors are under no obligation to detect this condition and handle it properly. This would not be a problem if TIFF files were read-only enti-*

ties, but they are not. This warning covers both TIFF field value offsets and fields that are defined as offsets, such as StripOffsets.

Point to real data. *All strip offsets must reference valid locations. (It is not legal to use an offset of 0 to mean something special.)*

Beware of extra components. *Some TIFF files may have more components per pixel than you think. A Baseline TIFF reader must skip over them gracefully, using the values of the SamplesPerPixel and BitsPerSample fields. For example, it is possible that the data will have a PhotometricInterpretation of RGB but have 4 SamplesPerPixel. See ExtraSamples for further details.*

Beware of new field types. *Be prepared to handle unexpected field types such as floating-point data. A Baseline TIFF reader must skip over such fields gracefully. Do not expect that BYTE, ASCII, SHORT, LONG, and RATIONAL will always be a complete list of field types.*

Beware of new pixel types. *Some TIFF files may have pixel data that consists of something other than unsigned integers. If the SampleFormat field is present and the value is not 1, a Baseline TIFF reader that cannot handle the SampleFormat value must terminate the import process gracefully.*

Notes on Required Fields

CDB-compliant TIFF readers require ImageWidth and ImageLength fields to be a power of 2. ImageWidth need not be the same as ImageLength. CDB-compliant TIFF readers do not consider data that does not conform to this

ImageWidth, ImageLength. Both "SHORT" and "LONG" TIFF field types are allowed and must be handled properly by readers. TIFF writers can use either type. TIFF readers are not required to read arbitrarily large files however. Some readers will give up if the entire image cannot fit into available memory. (In such cases the reader should inform the user about the problem.) Others will probably not be able to handle ImageWidth greater than 65535.

RowsPerStrip. SHORT or LONG. Readers must be able to handle any value between 1 and 2**32-1. However, some readers may try to read an entire strip into memory at one time. If the entire image is one strip, the application may run out of memory. Recommendation: Set RowsPerStrip such that the size of each strip is about 8K bytes. Do this even for uncompressed data because it is easy for a writer and makes things simpler for readers. Note that extremely wide high-resolution images may have rows larger than 8K bytes; in this case, RowsPerStrip should be 1, and the strip will be larger than 8K.

StripOffsets. SHORT or LONG.

StripByteCounts. SHORT or LONG.

CDB-compliant TIFF readers do not consider the XResolution and YResolution TIFF tags.

XResolution, YResolution. RATIONAL. Note that the X and Y resolutions may be unequal. A TIFF reader must be able to handle this case. Typically, TIFF pixel-editors do not care about the resolution, but applications (such as page layout programs) do care.

ResolutionUnit. SHORT. TIFF readers must be prepared to handle all three values for ResolutionUnit.

CDB-compliant TIFF readers do not consider the ResolutionUnit TIFF tag.

Section 8: Baseline Field Reference Guide

This section contains detailed information about all the Baseline fields defined in this version of TIFF. A *Baseline field* is any field commonly found in a Baseline TIFF file, whether required or not.

For convenience, fields that were defined in earlier versions of the TIFF specification but are no longer generally recommended have also been included in this section.

New fields that are associated with optional features are not listed in this section. See Part 2 for descriptions of these new fields. There is a complete list of all fields described in this specification in Appendix A, and there are entries for all TIFF fields in the index.

More fields may be added in future versions. Whenever possible they will be added in a way that allows old TIFF readers to read newer TIFF files.

The documentation for each field contains:

- the name of the field
- the Tag number
- the field Type
- the required Number of Values (N); i.e., the Count
- comments describing the field
- the default, if any

If the field does not exist, readers must assume the default value for the field.

Most of the fields described in this part of the document are not required or are required only for particular types of TIFF files. See the preceding sections for lists of required fields.

Before defining the fields, you must understand these basic concepts: A Baseline TIFF *image* is defined to be a two-dimensional array of *pixels*, each of which consists of one or more color *components*. Monochromatic data has one color component per pixel, while RGB color data has three color components per pixel.

The Fields

CDB-compliant TIFF readers do not consider the Artist TIFF tag.

Artist

Person who created the image.

Tag = 315 (13B.H)

Type = ASCII

Note: some older TIFF files used this tag for storing Copyright information.

BitsPerSample

Number of bits per component.

Tag = 258 (102.H)

Type = SHORT

N = SamplesPerPixel

Note that this field allows a different number of bits per component for each component corresponding to a pixel. For example, RGB color data could use a different number of bits per component for each of the three color planes. Most RGB files will have the same number of BitsPerSample for each component. Even in this case, the writer must write all three values.

Default = 1. See also SamplesPerPixel.

CellLength

CDB-compliant TIFF readers do not consider the CellLength TIFF tag.
The length of the dithering or halftoning matrix used to create a dithered or halftoned bilevel file. Tag = 265 (109.H) Type = SHORT N = 1 This field should only be present if Threshholding = 2 No

The length of the dithering or halftoning matrix used to create a dithered or halftoned bilevel file.

Tag = 265 (109.H)

Type = SHORT

N = 1

This field should only be present if Threshholding = 2

No default. See also Threshholding.

CellWidth

CDB-compliant TIFF readers do not consider the CellWidth TIFF tag.

The width of the dithering or halftoning matrix used to create a dithered or halftoned bilevel file.Tag = 264 (108.H)

Type = SHORT

N = 1

No default. See also Threshholding.

ColorMap

CDB-compliant TIFF readers do not consider the ColorMap TIFF tag.

A color map for palette color images.

Tag = 320 (140.H)

Type = SHORT

N = 3 * (2**BitsPerSample)

This field defines a Red-Green-Blue color map (often called a lookup table) for palette-color images. In a palette-color image, a pixel value is used to index into an RGB lookup table. For example, a palette-color pixel having a value of 0 would be displayed according to the 0th Red, Green, Blue triplet.

In a TIFF ColorMap, all the Red values come first, followed by the Green values, then the Blue values. The number of values for each color is 2**BitsPerSample. Therefore, the ColorMap field for an 8-bit palette-color image would have 3 * 256 values.

The width of each value is 16 bits, as implied by the type of SHORT. 0 represents the minimum intensity, and 65535 represents the maximum intensity. Black is represented by 0,0,0, and white by 65535, 65535, 65535.

See also PhotometricInterpretation—palette color.

No default. ColorMap must be included in all palette-color images.

Compression

CDB-compliant TIFF readers do not consider compressed TIFF image.

Compression scheme used on the image data.

Tag = 259 (103.H)

Type = SHORT

N = 1

1 = No compression, but pack data into bytes as tightly as possible leaving no unused bits except at the end of a row.

If	Then the sample values are stored as an array of type:
BitsPerSample = 16 for all samples	SHORT
BitsPerSample = 32 for all samples	LONG
Otherwise	BYTE

Each row is padded to the next BYTE/SHORT/LONG boundary, consistent with the preceding BitsPerSample rule.

If the image data is stored as an array of SHORTs or LONGs, the byte ordering must be consistent with that specified in bytes 0 and 1 of the TIFF file header. Therefore, little-endian format files will have the least significant bytes preceding the most significant bytes, while big-endian format files will have the opposite order.

If the number of bits per component is not a power of 2, and you are willing to give up some space for better performance, use the next higher power of 2. For example, if your data can be represented in 6 bits, set BitsPerSample to 8 instead of 6, and then convert the range of the values from [0,63] to [0,255].

Rows must begin on byte boundaries. *(SHORT boundaries if the data is stored as SHORTs, LONG boundaries if the data is stored as LONGs).*

Some graphics systems require image data rows to be word-aligned or double-word-aligned, and padded to word-boundaries or double-word boundaries. Uncompressed TIFF rows will need to be copied into word-aligned or double-word-aligned row buffers before being passed to the graphics routines in these environments.

2 = CCITT Group 3 1-Dimensional Modified Huffman run-length encoding. See Section 10. BitsPerSample must be 1, since this type of compression is defined only for bilevel images.

32773 = PackBits compression, a simple byte-oriented run-length scheme. See Section 9 for details.

Data compression applies only to the image data, pointed to by StripOffsets.

Default = 1.

Copyright

CDB-compliant TIFF readers do not consider the Copyright TIFF tag.

Copyright notice.

Tag = 33432 (8298.H)

Type = ASCII

Copyright notice of the person or organization that claims the copyright to the image. The complete copyright statement should be listed in this field including any dates and statements of claims. For example, "Copyright, John Smith, 19xx. All rights reserved."

DateTime

CDB-compliant TIFF readers do not consider the DateTime TIFF tag.

Date and time of image creation.

Tag = 306 (132.H)

Type = ASCII

N = 20

The format is: "YYYY:MM:DD HH:MM:SS", with hours like those on a 24-hour clock, and one space character between the date and the time. The length of the string, including the terminating NUL, is 20 bytes.

ExtraSamples

Description of extra components.

Tag = 338 (152.H)

Type = SHORT

N = m

Specifies that each pixel has *m* extra components whose interpretation is defined by one of the values listed below. When this field is used, the SamplesPerPixel field has a value greater than the PhotometricInterpretation field suggests.

For example, full-color RGB data normally has SamplesPerPixel=3. If SamplesPerPixel is greater than 3, then the ExtraSamples field describes the meaning of the extra samples. If SamplesPerPixel is, say, 5 then ExtraSamples will contain 2 values, one for each extra sample.

ExtraSamples is typically used to include non-color information, such as opacity, in an image. The possible values for each item in the field's value are:

0 = Unspecified data

1 = Associated alpha data (with pre-multiplied color)

CDB-compliant TIFF readers do not consider unassociated alpha data.

2 = Unassociated alpha data

Associated alpha data is opacity information; it is fully described in Section 21. Unassociated alpha data is transparency information that logically exists independent of an image; it is commonly called a soft matte. Note that including both unassociated and associated alpha is undefined because associated alpha specifies that color components are pre-multiplied by the alpha component, while unassociated alpha specifies the opposite.

By convention, extra components that are present must be stored as the "last components" in each pixel. For example, if SamplesPerPixel is 4 and there is 1 extra component, then it is located in the last component location (SamplesPerPixel-1) in each pixel.

Components designated as "extra" are just like other components in a pixel. In particular, the size of such components is defined by the value of the BitsPerSample field.

With the introduction of this field, TIFF readers must not assume a particular SamplesPerPixel value based on the value of the PhotometricInterpretation field. For example, if the file is an RGB file, SamplesPerPixel may be greater than 3.

The default is no extra samples. This field must be present if there are extra samples.

See also SamplesPerPixel, AssociatedAlpha.

FillOrder

The logical order of bits within a byte.

Tag = 266 (10A.H)

Type = SHORT

N = 1

1 = pixels are arranged within a byte such that pixels with lower column values are stored in the higher-order bits of the byte.

1-bit uncompressed data example: Pixel 0 of a row is stored in the high-order bit of byte 0, pixel 1 is stored in the next-highest bit, ..., pixel 7 is stored in the low-order bit of byte 0, pixel 8 is stored in the high-order bit of byte 1, and so on.

CCITT 1-bit compressed data example: The high-order bit of the first compression code is stored in the high-order bit of byte 0, the next-highest bit of the first compression code is stored in the next-highest bit of byte 0, and so on.

2 = pixels are arranged within a byte such that pixels with lower column values are stored in the lower-order bits of the byte.

We recommend that FillOrder=2 be used only in special-purpose applications. It is easy and inexpensive for writers to reverse bit order by using a 256-byte lookup table. *FillOrder = 2 should be used only when BitsPerSample = 1 and the data is either uncompressed or compressed using CCITT 1D or 2D compression, to avoid potentially ambigous situations.*

Support for FillOrder=2 is not required in a Baseline TIFF compliant reader

Default is FillOrder = 1.

CDB-compliant TIFF readers do not consider the FreeByteCounts TIFF tag.

FreeByteCounts

For each string of contiguous unused bytes in a TIFF file, the number of bytes in the string.

Tag = 289 (121.H)

Type = LONG

Not recommended for general interchange.

See also FreeOffsets.

CDB-compliant TIFF readers do not consider the FreeOffsets TIFF tag.

FreeOffsets

For each string of contiguous unused bytes in a TIFF file, the byte offset of the string.

Tag = 288 (120.H)

Type = LONG

Not recommended for general interchange.

See also FreeByteCounts.

The CDB specification assumes that the gray response curve of TIFF image files to be linear. As a result, CDB-compliant TIFF readers do not consider the GrayResponseCurve Tag data.

GrayResponseCurve

For grayscale data, the optical density of each possible pixel value.

Tag = 291 (123.H)

Type = SHORT

N = 2**BitsPerSample

The 0th value of GrayResponseCurve corresponds to the optical density of a pixel having a value of 0, and so on.

This field may provide useful information for sophisticated applications, but it is currently ignored by most TIFF readers.

See also GrayResponseUnit, PhotometricInterpretation.

The CDB specification assumes that the gray response curve of TIFF image files to be linear. As a result, CDB-compliant TIFF readers do not consider the GrayResponseUnit Tag data.

GrayResponseUnit

The precision of the information contained in the GrayResponseCurve.

Tag = 290 (122.H)

Type = SHORT

N = 1

Because optical density is specified in terms of fractional numbers, this field is necessary to interpret the stored integer information. For example, if GrayScaleResponseUnits is set to 4 (ten-thousandths of a unit), and a GrayScaleResponseCurve number for gray level 4 is 3455, then the resulting actual value is 0.3455.

Optical densitometers typically measure densities within the range of 0.0 to 2.0.

1 = Number represents tenths of a unit.

2 = Number represents hundredths of a unit.

3 = Number represents thousandths of a unit.

4 = Number represents ten-thousandths of a unit.

5 = Number represents hundred-thousandths of a unit.

Modifies GrayResponseCurve.

See also GrayResponseCurve.

For historical reasons, the default is 2. However, for greater accuracy, 3 is recommended.

HostComputer

CDB-compliant TIFF readers do not consider the HostComputer TIFF tag.

The computer and/or operating system in use at the time of image creation.

Tag = 316 (13C.H)

Type = ASCII

See also Make, Model, Software.

ImageDescription

CDB-compliant TIFF readers do not consider the ImageDescription TIFF tag.

A string that describes the subject of the image.

Tag = 270 (10E.H)

Type = ASCII

For example, a user may wish to attach a comment such as "1988 company picnic" to an image.

ImageLength

CDB-compliant TIFF readers require ImageWidth and ImageLength fields to be a power of 2. ImageLength need not be the same as ImageWidth. CDB-compliant TIFF readers do not consider image that does not conform to this requirement.

The number of rows of pixels in the image.

Tag = 257 (101.H)

Type = SHORT or LONG

N = 1

No default. See also ImageWidth.

ImageWidth

The number of columns in the image, i.e., the number of pixels per row.

Tag = 256 (100.H)

Type = SHORT or LONG

N = 1

No default. See also ImageLength.

CDB-compliant Tiff readers do not consider the Make TIFF tag.

Make

The scanner manufacturer.

Tag = 271 (10F.H)

Type = ASCII

Manufacturer of the scanner, video digitizer, or other type of equipment used to generate the image. Synthetic images should not include this field.

See also Model, Software.

The CDB specification establishes that the MaxSampleValue to be always equal to the maximum value that can be represented by the number format representation. As a result, CDB-compliant TIFF readers do not consider

MaxSampleValue

The maximum component value used.

Tag = 281 (119.H)

Type = SHORT

N = SamplesPerPixel

This field is not to be used to affect the visual appearance of an image when it is displayed or printed. Nor should this field affect the interpretation of any other field; it is used only for statistical purposes.

Default is 2**(BitsPerSample) - 1.

The CDB specification establishes that the MinSampleValue to be always equal to 0 for image data and to minimum value that can be represented by the number format representation. As a result, CDB-compliant TIFF readers do not consider

MinSampleValue

The minimum component value used.

Tag = 280 (118.H)

Type = SHORT

N = SamplesPerPixel

See also MaxSampleValue.

Default is 0.

CDB-compliant TIFF readers do not consider the Model TIFF tag.

Model

The scanner model name or number.

Tag = 272 (110.H)

Type = ASCII

The model name or number of the scanner, video digitizer, or other type of equipment used to generate the image.

See also Make, Software.

The CDB specification assumes that the data is full or reduced resolution only. As a result, CDB-compliant TIFF readers only consider images and DEMs data whose PhotometricInterpretation Tag

NewSubfileType

A general indication of the kind of data contained in this subfile.

Tag = 254 (FE.H)

Type = LONG

N = 1

Replaces the old SubfileType field, due to limitations in the definition of that field.

NewSubfileType is mainly useful when there are multiple subfiles in a single TIFF file.

This field is made up of a set of 32 flag bits. Unused bits are expected to be 0. Bit 0 is the low-order bit.

Currently defined values are:

Bit 0 is 1 if the image is a reduced-resolution version of another image in this TIFF file; else the bit is 0.

Bit 1 is 1 if the image is a single page of a multi-page image (see the PageNumber field description); else the bit is 0.

Bit 2 is 1 if the image defines a transparency mask for another image in this TIFF file. The PhotometricInterpretation value must be 4, designating a transparency mask.

These values are defined as bit flags because they are independent of each other.

Default is 0.

The CDB specification assumes that the data is organized such that the 0th row represents the visual top of the grid data (or image), and the 0th column represents the visual left-hand side. As a result, CDB-compliant TIFF readers do not consider image and DEM data whose

Orientation

The orientation of the image with respect to the rows and columns.

Tag = 274 (112.H)

Type = SHORT

N = 1

1 = The 0th row represents the visual top of the image, and the 0th column represents the visual left-hand side.

2 = The 0th row represents the visual top of the image, and the 0th column represents the visual right-hand side.

3 = The 0th row represents the visual bottom of the image, and the 0th column represents the visual right-hand side.

4 = The 0th row represents the visual bottom of the image, and the 0th column represents the visual left-hand side.

5 = The 0th row represents the visual left-hand side of the image, and the 0th column represents the visual top.

6 = The 0th row represents the visual right-hand side of the image, and the 0th column represents the visual top.

7 = The 0th row represents the visual right-hand side of the image, and the 0th column represents the visual bottom.

8 = The 0th row represents the visual left-hand side of the image, and the 0th column represents the visual bottom.

Default is 1.

Support for orientations other than 1 is not a Baseline TIFF requirement.

PhotometricInterpretation

CDB-compliant TIFF readers only consider images whose PhotometricInterpretation Tag value is 1

CDB-compliant TIFF readers do not consider WhiteIsZero images.

CDB-compliant TIFF readers do not consider palette-color images, i.e. PhotometricInterpretation = 3 (Palette

CDB-compliant simulator TIFF readers do not consider transparency mask imagery data, i.e. PhotometricInterpretation = 4

The color space of the image data.

Tag = 262 (106.H)

Type = SHORT

N = 1

0 = WhiteIsZero. For bilevel and grayscale images: 0 is imaged as white. 2**BitsPerSample-1 is imaged as black. This is the normal value for Compression=2.

1 = BlackIsZero. For bilevel and grayscale images: 0 is imaged as black. 2**BitsPerSample-1 is imaged as white. If this value is specified for Compression=2, the image should display and print reversed.

2 = RGB. In the RGB model, a color is described as a combination of the three primary colors of light (red, green, and blue) in particular concentrations. For each of the three components, 0 represents minimum intensity, and 2**BitsPerSample - 1 represents maximum intensity. Thus an RGB value of (0,0,0) represents black, and (255,255,255) represents white, assuming 8-bit components. For PlanarConfiguration = 1, the components are stored in the indicated order: first Red, then Green, then Blue. For PlanarConfiguration = 2, the StripOffsets for the component planes are stored in the indicated order: first the Red component plane StripOffsets, then the Green plane StripOffsets, then the Blue plane StripOffsets.

3= Palette color. In this model, a color is described with a single component. The value of the component is used as an index into the red, green and blue curves in the ColorMap field to retrieve an RGB triplet that defines the color. When PhotometricInterpretation=3 is used, ColorMap must be present and SamplesPerPixel must be 1.

4 = Transparency Mask.

This means that the image is used to define an irregularly shaped region of another image in the same TIFF file. SamplesPerPixel and BitsPerSample must be 1. PackBits compression is recommended. The 1-bits define the interior of the region; the 0-bits define the exterior of the region.

A reader application can use the mask to determine which parts of the image to display. Main image pixels that correspond to 1-bits in the transparency mask are imaged to the screen or printer, but main image pixels that correspond to 0-bits in the mask are not displayed or printed.

The image mask is typically at a higher resolution than the main image, if the main image is grayscale or color so that the edges can be sharp.

There is no default for PhotometricInterpretation, *and it is required.* Do not rely on applications defaulting to what you want.

PlanarConfiguration

How the components of each pixel are stored.

Tag = 284 (11C.H)

Type = SHORT

N = 1

1 = *Chunky* format. The component values for each pixel are stored contiguously. The order of the components within the pixel is specified by PhotometricInterpretation. For example, for RGB data, the data is stored as RGBRGBRGB...

2 = *Planar* format. The components are stored in separate "component planes." The values in StripOffsets and StripByteCounts are then arranged as a 2-dimensional array, with SamplesPerPixel rows and StripsPerImage columns. (All of the columns for row 0 are stored first, followed by the columns of row 1, and so on.) PhotometricInterpretation describes the type of data stored in each component plane. For example, RGB data is stored with the Red components in one component plane, the Green in another, and the Blue in another.

PlanarConfiguration=2 is not currently in widespread use and it is not recommended for general interchange. It is used as an extension and Baseline TIFF readers are not required to support it.

If SamplesPerPixel is 1, PlanarConfiguration is irrelevant, and need not be included.

If a row interleave effect is desired, a writer might write out the data as PlanarConfiguration=2—separate sample planes—but break up the planes into multiple strips (one row per strip, perhaps) and interleave the strips.

Default is 1. See also BitsPerSample, SamplesPerPixel.

ResolutionUnit

The CDB specification establishes a series of conventions that govern the resolution of TIFF files. As a result, CDB-compliant TIFF readers do not consider the ResolutionUnit TIFF tag.

The unit of measurement for XResolution and YResolution.

Tag = 296 (128.H)

Type = SHORT

N = 1

To be used with XResolution and YResolution.

1 = No absolute unit of measurement. Used for images that may have a non-square aspect ratio, but no meaningful absolute dimensions.

The drawback of ResolutionUnit=1 is that different applications will import the image at different sizes. Even if the decision is arbitrary, it might be better to use dots per inch or dots per centimeter, and to pick XResolution and YResolution so that the aspect ratio is correct and the maximum dimension of the image is about four inches (the "four" is arbitrary.)

2 = Inch.

3 = Centimeter.

Default is 2.

RowsPerStrip

The number of rows per strip.

Tag = 278 (116.H)

Type = SHORT or LONG

N = 1

TIFF image data is organized into strips for faster random access and efficient I/O buffering.

RowsPerStrip and ImageLength together tell us the number of strips in the entire image. The equation is:

StripsPerImage = floor ((ImageLength + RowsPerStrip - 1) / RowsPerStrip).

StripsPerImage is *not* a field. It is merely a value that a TIFF reader will want to compute because it specifies the number of StripOffsets and StripByteCounts for the image.

Note that either SHORT or LONG values can be used to specify RowsPerStrip. SHORT values may be used for small TIFF files. It should be noted, however, that earlier TIFF specification revisions required LONG values and that some software may not accept SHORT values.

The default is 2**32 - 1, which is effectively infinity. That is, the entire image is one strip.

Use of a single strip is not recommended. Choose RowsPerStrip such that each strip is about 8K bytes, even if the data is not compressed, since it makes buffering simpler for readers. The "8K" value is fairly arbitrary, but seems to work well.

See also ImageLength, StripOffsets, StripByteCounts, TileWidth, TileLength, TileOffsets, TileByteCounts.

SamplesPerPixel

The number of components per pixel.

Tag = 277 (115.H)

Type = SHORT

N = 1

SamplesPerPixel is *usually* 1 for bilevel, grayscale, and palette-color images. SamplesPerPixel is *usually* 3 for RGB images.

Default = 1. See also BitsPerSample, PhotometricInterpretation, *ExtraSamples*.

Software

CDB-compliant TIFF readers do not consider the Software TIFF tag.

Name and version number of the software package(s) used to create the image.

Tag = 305 (131.H)

Type = ASCII

See also Make, Model.

StripByteCounts

For each strip, the number of bytes in the strip after compression.

Tag = 279 (117.H)

Type = SHORT or LONG

N = StripsPerImage for PlanarConfiguration equal to 1.

= SamplesPerPixel * StripsPerImage for PlanarConfiguration equal to 2

This tag is required for Baseline TIFF files.

No default.

See also StripOffsets, RowsPerStrip, TileOffsets, TileByteCounts.

StripOffsets

For each strip, the byte offset of that strip.

Tag = 273 (111.H)

Type = SHORT or LONG

N = StripsPerImage for PlanarConfiguration equal to 1.

= SamplesPerPixel * StripsPerImage for PlanarConfiguration equal to 2

The offset is specified with respect to the beginning of the TIFF file. Note that this implies that each strip has a location independent of the locations of other strips. This feature may be useful for editing applications. This required field is the only way for a reader to find the image data. *(Unless TileOffsets is used; see TileOffsets.)*

Note that either SHORT or LONG values may be used to specify the strip offsets. SHORT values may be used for small TIFF files. It should be noted, however, that earlier TIFF specifications required LONG strip offsets and that some software may not accept SHORT values.

For maximum compatibility with operating systems such as MS-DOS and Windows, the StripOffsets array should be less than or equal to 64K bytes in length, and the strips themselves, in both compressed and uncompressed forms, should not be larger than 64K bytes.

No default. See also StripByteCounts, RowsPerStrip, TileOffsets, TileByteCounts.

SubfileType

A general indication of the kind of data contained in this subfile.

Tag = 255 (FF.H)

Type = SHORT

N = 1

Currently defined values are:

1 = full-resolution image data

2 = reduced-resolution image data

3 = a single page of a multi-page image (see the PageNumber field description).

Note that several image types may be found in a single TIFF file, with each subfile described by its own IFD.

No default.

This field is deprecated. The NewSubfileType field should be used instead.

CDB-compliant TIFF readers do not consider image whose Subfile type = 3.

Threshholding

For black and white TIFF files that represent shades of gray, the technique used to convert from gray to black and white pixels.

Tag = 263 (107.H)

Type = SHORT

N = 1

CDB-compliant TIFF readers do not consider image data whose Thresholding TIFF tag is not equal 1.

1 = No dithering or halftoning has been applied to the image data.

2 = An ordered dither or halftone technique has been applied to the image data.

3 = A randomized process such as error diffusion has been applied to the image data.

Default is Threshholding = 1. See also CellWidth, CellLength.

XResolution

The number of pixels per ResolutionUnit in the ImageWidth direction.

Tag = 282 (11A.H)

Type = RATIONAL

N = 1

It is not mandatory that the image be actually displayed or printed at the size implied by this parameter. It is up to the application to use this information as it wishes.

The CDB specification establishes a series of conventions that govern the resolution of TIFF files. As a result, CDB-compliant TIFF readers do not consider this TIFF tag.

No default. See also YResolution, ResolutionUnit.

YResolution

The number of pixels per ResolutionUnit in the ImageLength direction.

Tag = 283 (11B.H)

Type = RATIONAL

N = 1

The CDB specification establishes a series of conventions that govern the resolution of TIFF files. As a result, CDB-compliant TIFF readers do not consider this TIFF tag.

No default. See also XResolution, ResolutionUnit.

Section 9: PackBits Compression

CDB-compliant TIFF readers do not consider PackBits compressed TIFF data. As a result, section 9 is not applicable to CDB-compliant TIFF readers.

This section describes TIFF compression type 32773, a simple byte-oriented run-length scheme.

Description

In choosing a simple byte-oriented run-length compression scheme, we arbitrarily chose the Apple Macintosh PackBits scheme. It has a good worst case behavior (at most 1 extra byte for every 128 input bytes). For Macintosh users, the toolbox utilities PackBits and UnPackBits will do the work for you, but it is easy to implement your own routines.

A pseudo code fragment to unpack might look like this:

```
Loop until you get the number of unpacked bytes you are expecting:
    Read the next source byte into n.
    If n is between 0 and 127 inclusive, copy the next n+1 bytes literally.
    Else if n is between -127 and -1 inclusive, copy the next byte -n+1
    times.
    Else if n is -128, noop.
Endloop
```

In the inverse routine, it is best to encode a 2-byte repeat run as a replicate run except when preceded and followed by a literal run. In that case, it is best to merge the three runs into one literal run. Always encode 3-byte repeats as replicate runs.

That is the essence of the algorithm. Here are some additional rules:

- Pack each row separately. Do not compress across row boundaries.
- The number of uncompressed bytes per row is defined to be (ImageWidth + 7) / 8. If the uncompressed bitmap is required to have an even number of bytes per row, decompress into word-aligned buffers.
- If a run is larger than 128 bytes, encode the remainder of the run as one or more additional replicate runs.

When PackBits data is decompressed, the result should be interpreted as per compression type 1 (no compression).

Section 10: Modified Huffman Compression

CDB-compliant TIFF readers do not consider Modified Huffman compressed TIFF data. As a result, section 10 is not applicable to CDB-compliant TIFF readers.

This section describes TIFF compression scheme 2, a method for compressing bilevel data based on the CCITT Group 3 1D facsimile compression scheme.

References

- "Standardization of Group 3 facsimile apparatus for document transmission," Recommendation T.4, Volume VII, Fascicle VII.3, Terminal Equipment and Protocols for Telematic Services, The International Telegraph and Telephone Consultative Committee (CCITT), Geneva, 1985, pages 16 through 31.
- "Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus," Recommendation T.6, Volume VII, Fascicle VII.3, Terminal Equipment and Protocols for Telematic Services, The International Telegraph and Telephone Consultative Committee (CCITT), Geneva, 1985, pages 40 through 48.

We do not believe that these documents are necessary in order to implement Compression=2. We have included (verbatim in most places) all the pertinent information in this section. However, if you wish to order the documents, you can write to ANSI, Attention: Sales, 1430 Broadway, New York, N.Y., 10018. Ask for the publication listed above—it contains both Recommendation T.4 and T.6.

Relationship to the CCITT Specifications

The CCITT Group 3 and Group 4 specifications describe communications protocols for a particular class of devices. They are not by themselves sufficient to describe a disk data format. Fortunately, however, the CCITT coding schemes can be readily adapted to this different environment. The following is one such adaptation. Most of the language is copied directly from the CCITT specifications.

See Section 11 for additional CCITT compression options.

Coding Scheme

A line (row) of data is composed of a series of variable length code words. Each code word represents a run length of all white or all black. (Actually, more than one code word may be required to code a given run, in a manner described below.) White runs and black runs alternate.

To ensure that the receiver (decompressor) maintains color synchronization, all data lines begin with a white run-length code word set. If the actual scan line begins with a black run, a white run-length of zero is sent (written). Black or white run-lengths are defined by the code words in Tables 1 and 2. The code words are of two types: Terminating code words and Make-up code words. Each run-length is represented by zero or more Make-up code words followed by exactly one Terminating code word.

Run lengths in the range of 0 to 63 pels (pixels) are encoded with their appropriate Terminating code word. Note that there is a different list of code words for black and white run-lengths.

Run lengths in the range of 64 to 2623 (2560+63) pels are encoded first by the Make-up code word representing the run-length that is nearest to, not longer than, that required. This is then followed by the Terminating code word representing the difference between the required run-length and the run-length represented by the Make-up code.

Run lengths in the range of lengths longer than or equal to 2624 pels are coded first by the Make-up code of 2560. If the remaining part of the run (after the first Make-up code of 2560) is 2560 pels or greater, additional Make-up code(s) of 2560 are issued until the remaining part of the run becomes less than 2560 pels. Then the remaining part of the run is encoded by Terminating code or by Make-up code plus Terminating code, according to the range mentioned above.

It is considered an unrecoverable error if the sum of the run-lengths for a line does not equal the value of the ImageWidth field.

New rows always begin on the next available byte boundary.

No EOL code words are used. No fill bits are used, except for the ignored bits at the end of the last byte of a row. RTC is not used.

An encoded CCITT string is self-photometric, defined in terms of white and black runs. Yet TIFF defines a tag called PhotometricInterpretation that also purports to define what is white and what is black. Somewhat arbitrarily, we adopt the following convention:

The "normal" PhotometricInterpretation for bilevel CCITT compressed data is WhiteIsZero. In this case, the CCITT "white" runs are to be interpretated as white, and the CCITT "black" runs are to be interpreted as black. However, if the PhotometricInterpretation is BlackIsZero, the TIFF reader must reverse the meaning of white and black when displaying and printing the image.

Table 1/T.4 Terminating codes

White run length	Code word	Black run length	Code word
0	00110101	0	0000110111
1	000111	1	010
2	0111	2	11
3	1000	3	10
4	1011	4	011
5	1100	5	0011
6	1110	6	0010
7	1111	7	00011
8	10011	8	000101
9	10100	9	000100
10	00111	10	0000100
11	01000	11	0000101
12	001000	12	0000111
13	000011	13	00000100
14	110100	14	00000111
15	110101	15	000011000
16	101010	16	0000010111
17	101011	17	0000011000
18	0100111	18	0000001000
19	0001100	19	00001100111
20	0001000	20	00001101000
21	0010111	21	00001101100
22	0000011	22	00000110111
23	0000100	23	00000101000
24	0101000	24	00000010111
25	0101011	25	00000011000
26	0010011	26	000011001010
27	0100100	27	000011001011
28	0011000	28	000011001100
29	00000010	29	000011001101
30	00000011	30	000001101000
31	00011010	31	000001101001
32	00011011	32	000001101010
33	00010010	33	000001101011
34	00010011	34	000011010010
35	00010100	35	000011010011
36	00010101	36	000011010100
37	00010110	37	000011010101
38	00010111	38	000011010110
39	00101000	39	000011010111
40	00101001	40	000001101100
41	00101010	41	000001101101
42	00101011	42	000011011010
43	00101100	43	000011011011
44	00101101	44	000001010100
45	00000100	45	000001010101
46	00000101	46	000001010110
47	00001010	47	000001010111
48	00001011	48	000001100100
49	01010010	49	000001100101
50	01010011	50	000001010010
51	01010100	51	000001010011

White run length	Code word	Black run length	Code word
52	01010101	52	000000100100
53	00100100	53	000000110111
54	00100101	54	000000111000
55	01011000	55	000000100111
56	01011001	56	000000101000
57	01011010	57	000001011000
58	01011011	58	000001011001
59	01001010	59	000000101011
60	01001011	60	000000101100
61	00110010	61	000001011010
62	00110011	62	000001100110
63	00110100	63	000001100111

Table 2/T.4 Make-up codes

White run length	Code word	Black run length	Code word
64	11011	64	0000001111
128	10010	128	000011001000
192	010111	192	000011001001
256	0110111	256	000001011011
320	00110110	320	000000110011
384	00110111	384	000000110100
448	01100100	448	000000110101
512	01100101	512	0000001101100
576	01101000	576	0000001101101
640	01100111	640	0000001001010
704	011001100	704	0000001001011
768	011001101	768	0000001001100
832	011010010	832	0000001001101
896	011010011	896	0000001110010
960	011010100	960	0000001110011
1024	011010101	1024	0000001110100
1088	011010110	1088	0000001110101
1152	011010111	1152	0000001110110
1216	011011000	1216	0000001110111
1280	011011001	1280	0000001010010
1344	011011010	1344	0000001010011
1408	011011011	1408	0000001010100
1472	010011000	1472	0000001010101
1536	010011001	1536	0000001011010
1600	010011010	1600	0000001011011
1664	011000	1664	0000001100100
1728	010011011	1728	0000001100101
EOL	000000000001	EOL	00000000000

Additional make-up codes

White and Black run length	Make-up code word
1792	00000001000
1856	00000001100
1920	00000001101
1984	000000010010
2048	000000010011
2112	000000010100
2176	000000010101
2240	000000010110
2304	000000010111
2368	000000011100
2432	000000011101
2496	000000011110
2560	000000011111

Part 2: TIFF Extensions

Part 2 contains extensions to Baseline TIFF. TIFF Extensions are TIFF features that may not be supported by all TIFF readers. TIFF creators who use these features will have to work closely with TIFF readers in their part of the industry to ensure successful interchange.

The features described in this part were either contained in earlier versions of the specification, or have been approved by the TIFF Advisory Committee.

Section 11: CCITT Bilevel Encodings

CDB-compliant TIFF readers do not consider CCITT Bi-level encoded TIFF data. As a result, section 11 is not applicable to CDB-compliant TIFF

The following fields are used when storing binary pixel arrays using one of the encodings adopted for raster-graphic interchange in numerous CCITT and ISO (International Organization for Standards) recommendations and standards. These encodings are often spoken of as "Group III compression" and "Group IV compression" because their application in facsimile transmission is the most widely known.

For the specialized use of these encodings in storing facsimile-transmission images, further guidelines can be obtained from the TIFF Class F document, available on-line in the same locations as this specification. This document is administered by another organization; paper copies are not available from Adobe.

Compression

Tag = 259 (103.H)

Type = SHORT

N = 1

3 = T4-encoding: CCITT T.4 bi-level encoding as specified in section 4, Coding, of CCITT Recommendation T.4: "Standardization of Group 3 Facsimile apparatus for document transmission." International Telephone and Telegraph Consultative Committee (CCITT, Geneva: 1988).

See the T4Options field for T4-encoding options such as 1D vs 2D coding.

4 = T6-encoding: CCITT T.6 bi-level encoding as specified in section 2 of CCITT Recommendation T.6: "Facsimile coding schemes and coding control functions for Group 4 facsimile apparatus." International Telephone and Telegraph Consultative Committee (CCITT, Geneva: 1988).

See the T6Options field for T6-encoding options such as escape into uncompressed mode to avoid negative-compression cases.

Application in Image Interchange

CCITT Recommendations T.4 and T.6 are specified in terms of the serial bit-by-bit creation and processing of a variable-length binary string that encodes bi-level (black and white) pixels of a rectangular image array. Generally, the encoding schemes are described in terms of bit-serial communication procedures and the end-to-end coordination that is required to gain reliable delivery over inherently unreliable data links. The Group 4 procedures, with their T6-encoding, represent a significant simplification because it is assumed that a reliable communication medium is employed, whether ISDN or X.25 or some other trustworthy transport vehicle. Because image-storage systems and computers achieve data integrity and communication reliability in other ways, the T6-encoding tends to be prefered for imaging applications. When computer storage and retrieval and interchange of facsimile material are of interest, the T4-encodings provide a better match to the

current generation of Group 3 facsimile products and their defenses against data corruption as the result of transmission defects.

Whichever form of encoding is preferable for a given application, there are a number of adjustments that need to be made to account for the capture of the CCITT binary-encoding strings as part of electronically-stored material and digital-image interchange.

PhotometricInterpretation. An encoded CCITT string is self-photometric, defined in terms of white and black runs. Yet TIFF defines a tag called PhotometricInterpretation that also purports to define what is white and what is black. Somewhat arbitrarily, we adopt the following convention:

The "normal" PhotometricInterpretation for bilevel CCITT compressed data is WhiteIsZero. In this case, the CCITT "white" runs are to be interpretated as white, and the CCITT "black" runs are to be interpreted as black. However, if the PhotometricInterpretation is BlackIsZero, the TIFF reader must reverse the meaning of white and black when displaying and printing the image.

FillOrder. When CCITT encodings are used directly over a typical serial communication link, the order of the bits in the encoded string is the sequential order of the string, bit-by-bit, from beginning to end. This poses the following question: In which order should consecutive blocks of eight bits be assembled into octets (standard data bytes) for use within a computer system? The answer differs depending on whether we are concerned about preserving the serial-transmission sequence or preserving only the format of byte-organized sequences in memory and in stored files.

From the perspective of electronic interchange, as long as a receiver's reassembly of bits into bytes properly mirrors the way in which the bytes were disassembled by the transmitter, no one cares which order is seen on the transmission link because each multiple of 8 bits is transparently transmitted.

Common practice is to record arbitrary binary strings into storage sequences such that the first sequential bit of the string is found in the high-order bit of the first octet of the stored byte sequence. This is the standard case specified by TIFF FillOrder = 1, used in most bitmap interchange and the only case required in Baseline TIFF. This is also the approach used for the octets of standard 8-bit character data, with little attention paid to the fact that the most common forms of data communication transmit and reassemble individual 8-bit frames with the low-order-bit first!

For bit-serial transmission to a distant unit whose approach to assembling bits into bytes is unknown and supposed to be irrelevant, it is necessary to satisfy the expected sequencing of bits over the transmission link. This is the normal case for communication between facsimile units and also for computers and modems emulating standard Group 3 facsimile units. In this case, if the CCITT encoding is captured directly off of the link via standard communication adapters, TIFF FillOrder = 2 will usually apply to that stored data form.

Consequently, different TIFF FillOrder cases may arise when CCITT encodings are obtained by synthesis within a computer (including Group 4 transmission, which is treated more like computer data) instead of by capture from a Group 3 facsimile unit.

Because this is such a subtle situation, with surprisingly disruptive consequences for FillOrder mismatches, the following practice is urged whenever CCITT bilevel encodings are used:

a. TIFF FillOrder (tag 266) should always be explicitly specified.

b. FillOrder = 1 should be employed wherever possible in persistent material that is intended for interchange. This is the only reliable case for widespread interchange among computer systems, and it is important to explicitly confirm the honoring of standard assumptions.

c. FillOrder = 2 should occur only in highly-localized and preferably-transient material, as in a facsimile server supporting group 3 facsimile equipment. The tag should be present as a safeguard against the CCITT encoding "leaking" into an unsuspecting application, allowing readers to detect and warn against the occurence.

There are interchange situations where fill order is not distinguished, as when filtering the CCITT encoding into a PostScript level 2 image operation. In this case, as in most other cases of computer-based information interchange, FillOrder=1 is assumed, and any padding to a multiple of 8 bits is accomplished by adding a sufficient number of 0-bits to the end of the sequence.

Strips and Tiles. When CCITT bi-level encoding is employed, interaction with stripping (Section 3) and tiling (Section 15) is as follows:

a. Decompose the image into segments—individual pixel arrays representing the desired strip or tile configuration. The CCITT encoding procedures are applied most flexibly if the segments each have a multiple of 4 lines.

b. Individually encode each segment according to the specified CCITT bi-level encoding, as if each segment is a separate raster-graphic image.

The reason for this general rule is that CCITT bi-level encodings are generally progressive. That is, the initial line of pixels is encoded, and then subsequent lines, according to a variety of options, are encoded in terms of changes that need to be made to the preceding (unencoded) line. For strips and tiles to be individually usable, they must each start as fresh, independent encodings.

Miscellaneous features. There are provisions in CCITT encoding that are mostly meaningful during facsimile-transmission procedures. There is generally no significant application when storing images in TIFF or other data interchange formats, although TIFF applications should be tolerant and flexible in this regard. These features tend to have significance only when facilitating transfer between facsimile and non-facsimile applications of the encoded raster-graphic images. Further considerations for fill sequences, end-of-line flags, return-to-control (end-of-block) sequences and byte padding are introduced in discussion of the individual encoding options.

T4Options

Tag = 292 (124.H)

Type = LONG

N = 1

See Compression=3. This field is made up of a set of 32 flag bits. Unused bits must be set to 0. Bit 0 is the low-order bit.

Bit 0 is 1 for 2-dimensional coding (otherwise 1-dimensional is assumed). For 2-D coding, if more than one strip is specified, each strip must begin with a 1-

dimensionally coded line. That is, RowsPerStrip should be a multiple of "Parameter K," as documented in the CCITT specification.

Bit 1 is 1 if uncompressed mode is used.

Bit 2 is 1 if fill bits have been added as necessary before EOL codes such that EOL always ends on a byte boundary, thus ensuring an EOL-sequence of 1 byte preceded by a zero nibble: xxxx-0000 0000-0001.

Default is 0, for basic 1-dimensional coding. See also Compression.

T6Options

Tag = 293 (125.H)

Type = LONG

N = 1

See *Compression = 4*. This field is made up of a set of 32 flag bits. Unused bits must be set to 0. Bit 0 is the low-order bit. The default value is 0 (all bits 0).

bit 0 is unused and always 0.

bit 1 is 1 if uncompressed mode is allowed in the encoding.

In earlier versions of TIFF, this tag was named Group4Options. The significance has not changed and the present definition is compatible. The name of the tag has been changed to be consistent with the nomenclature of other T.6-encoding applications.

Readers should honor this option tag, and only this option tag, whenever T.6-Encoding is specified for Compression.

For T.6-Encoding, each segment (strip or tile) is encoded as if it were a separate image. The encoded string from each segment starts a fresh byte.

There are no one-dimensional line encodings in T.6-Encoding. Instead, even the first row of the segment's pixel array is encoded two-dimensionally by always assuming an invisible preceding row of all-white pixels. The 2-dimensional procedure for encoding the body of individual rows is the same as that used for 2-dimensional T.4-encoding and is described fully in the CCITT specifications.

The beginning of the encoding for each row of a strip or tile is conducted as if there is an imaginary preceding (0-width) white pixel, that is as if a fresh run of white pixels has just commenced. The completion of each line is encoded as if there are imaginary pixels beyond the end of the current line, and of the preceding line, in effect, of colors chosen such that the line is exactly completable by a code word, making the imaginary next pixel a changing element that's not actually used.

The encodings of successive lines follow contiguously in the binary T.6-Encoding stream with no special initiation or separation codewords. There are no provisions for fill codes or explicit end-of-line indicators. The encoding of the last line of the pixel array is followed immediately, in place of any additional line encodings, by a 24-bit End-of-Facsimile Block (EOFB).

000000000001000000000001.B.

The EOFB sequence is immediately followed by enough 0-bit padding to fit the entire stream into a sequence of 8-bit bytes.

General Application. Because of the single uniform encoding procedure, without disruptions by end-of-line codes and shifts into one-dimensional encodings, T.6-encoding is very popular for compression of bi-level images in document imaging systems. T.6-encoding trades off redundancy for minimum encoded size, relying on the underlying storage and transmission systems for reliable retention and communication of the encoded stream.

TIFF readers will operate most smoothly by always ignoring bits beyond the EOFB. Some writers may produce additional bytes of pad bits beyond the byte containing the final bit of the EOFB. Robust readers will not be disturbed by this prospect.

It is not possible to correctly decode a T.6-Encoding without knowledge of the exact number of pixels in each line of the pixel array. ImageWidth (or TileWidth, if used) must be stated exactly and accurately. If an image or segment is overscanned, producing extraneous pixels at the beginning or ending of lines, these pixels must be counted. Any cropping must be accomplished by other means. It is not possible to recover from a pixel-count deviation, even when one is detected. Failure of any row to be completed as expected is cause for abandoning further decoding of the entire segment. There is no requirement that ImageWidth be a multiple of eight, of course, and readers must be prepared to pad the final octet bytes of decoded bitmap rows with additional bits.

If a TIFF reader encounters EOFB before the expected number of lines has been extracted, it is appropriate to assume that the missing rows consist entirely of white pixels. Cautious readers might produce an unobtrusive warning if such an EOFB is followed by anything other than pad bits.

Readers that successfully decode the RowsPerStrip (or TileLength or residual ImageLength) number of lines are not required to verify that an EOFB follows. That is, it is generally appropriate to stop decoding when the expected lines are decoded or the EOFB is detected, whichever occurs first. Whether error indications or warnings are also appropriate depends upon the application and whether more precise troubleshooting of encoding deviations is important.

TIFF writers should always encode the full, prescribed number of rows, with a proper EOFB immediately following in the encoding. Padding should be by the least number of 0-bits needed for the T.6-encoding to exactly occupy a multiple of 8 bits. Only 0-bits should be used for padding, and StripByteCount (or TileByteCount) should not extend to any bytes not containing properly-formed T.6-encoding. In addition, even though not required by T.6-encoding rules, successful interchange with a large variety of readers and applications will be enhanced if writers can arrange for the number of pixels per line and the number of lines per strip to be multiples of eight.

Uncompressed Mode. Although T.6-encodings of simple bi-level images result in data compressions of 10:1 and better, some pixel-array patterns have T.6-encodings that require more bits than their simple bi-level bitmaps. When such cases are detected by encoding procedures, there is an optional extension for shifting to a form of uncompressed coding within the T.6-encoding string.

Uncompressed mode is not well-specified and many applications discourage its usage, prefering alternatives such as different compressions on a segment-by-segment (strip or tile) basis, or by simply leaving the image uncompressed in its

entirety. The main complication for readers is in properly restoring T.6-encoding after the uncompressed sequence is laid down in the current row.

Readers that have no provision for uncompressed mode will generally reject any case in which the flag is set. Readers that are able to process uncompressed-mode content within T.6-encoding strings can safely ignore this flag and simply process any uncompressed-mode occurences correctly.

Writers that are unable to guarantee the absence of uncompressed-mode material in any of the T.6-encoded segments must set the flag. The flag should be cleared (or defaulted) only when absence of uncompressed-mode material is assured. Writers that are able to inhibit the generation of uncompressed-mode extensions are encouraged to do so in order to maximize the acceptability of their T.6-encoding strings in interchange situations.

Because uncompressed-mode is not commonly used, the following description is best taken as suggestive of the general machinery. Interpolation of fine details can easily vary between implementations.

Uncompressed mode is signalled by the occurence of the 10-bit extension code string

0000001111.B

outside of any run-length make-up code or extension. Original unencoded image information follows. In this unencoded information, a 0-bit evidently signifies a white pixel, a 1-bit signifies a black pixel, and the TIFF PhotometricInterpretation will influence how these bits are mapped into any final uncompressed bitmap for use. The only modification made to the unencoded information is insertion of a 1-bit after every block of five consecutive 0-bits from the original image information. This is a transparency device that allows longer sequencences of 0-bits to be reserved for control conditions, especially ending the uncompressed-mode sequence. When it is time to return to compressed mode, the 8-bit exit sequence

0000001t.B

is appended to the material. The 0-bits of the exit sequence are not considered in applying the 1-bit insertion rule; up to four information 0-bits can legally precede the exit sequence. The trailing bit, 't,' specifies the color (via 0 or 1) that is understood in the next run of compressed-mode encoding. This lets a color other than white be assumed for the 0-width pixel on the left of the edge between the last uncompressed pixel and the resumed 2-dimensional scan.

Writers should confine uncompressed-mode sequences to the interiors of individual rows, never attempting to "wrap" from one row to the next. Readers must operate properly when the only encoding for a single row consists of an uncompressed-mode escape, a complete row of (proper 1-inserted) uncompressed information, and the extension exit. Technically, the exit pixel, 't,' should probably then be the opposite color of the last true pixel of the row, but readers should be generous in this case.

In handling these complex encodings, the encounter of material from a defective source or a corrupted file is particularly unsettling and mysterious. Robust readers will do well to defend against falling off the end of the world; e.g., unexpected EOFB sequences should be handled, and attempted access to data bytes that are not within the bounds of the present segment (or the TIFF file itself) should be avoided.

Section 12: Document Storage and Retrieval

CDB-compliant TIFF readers do not consider all TIFF tags related to document storage and retrieval. As a result, section 12 is not applicable to CDB-compliant TIFF readers.

These fields may be useful for document storage and retrieval applications. They will very likely be ignored by other applications.

DocumentName

The name of the document from which this image was scanned.

Tag = 269 (10D.H)

Type = ASCII

See also PageName.

PageName

The name of the page from which this image was scanned.

Tag = 285 (11D.H)

Type = ASCII

See also DocumentName.

No default.

PageNumber

The page number of the page from which this image was scanned.

Tag = 297 (129.H)

Type = SHORT

N = 2

This field is used to specify page numbers of a multiple page (e.g. facsimile) document. PageNumber[0] is the page number; PageNumber[1] is the total number of pages in the document. If PageNumber[1] is 0, the total number of pages in the document is not available.

Pages need not appear in numerical order.

The first page is numbered 0 (zero).

No default.

XPosition

X position of the image.

Tag = 286 (11E.H)

Type = RATIONAL

N = 1

The X offset in ResolutionUnits of the left side of the image, with respect to the left side of the page.

No default. See also YPosition.

YPosition

Y position of the image.

Tag = 287 (11F.H)

Type = RATIONAL

N = 1

The Y offset in ResolutionUnits of the top of the image, with respect to the top of the page. In the TIFF coordinate scheme, the positive Y direction is down, so that YPosition is always positive.

No default. See also XPosition.

Section 13: LZW Compression

This section describes TIFF compression scheme 5, an adaptive compression scheme for raster images.

Restrictions

When LZW compression was added to the TIFF specification, in Revision 5.0, it was thought to be public domain. This is, apparently, not the case.

The following paragraph has been approved by the Unisys Corporation:

"The LZW compression method is said to be the subject of United States patent number 4,558,302 and corresponding foreign patents owned by the Unisys Corporation. Software and hardware developers may be required to license this patent in order to develop and market products using the TIFF LZW compression option. Unisys has agreed that developers may obtain such a license on reasonable, non-discriminatory terms and conditions. Further information can be obtained from: Welch Licensing Department, Office of the General Counsel, M/S C1SW19, Unisys Corporation, Blue Bell, Pennsylvania, 19424."

Reportedly, there are also other companies with patents that may affect LZW implementors.

Reference

Terry A. Welch, "A Technique for High Performance Data Compression", IEEE Computer, vol. 17 no. 6 (June 1984). Describes the basic Lempel-Ziv & Welch (LZW) algorithm in very general terms. The author's goal is to describe a hardware-based compressor that could be built into a disk controller or database engine and used on all types of data. There is no specific discussion of raster images. This section gives sufficient information so that the article is not required reading.

Characteristics

LZW compression has the following characteristics:

- LZW works for images of various bit depths.
- LZW has a reasonable worst-case behavior.
- LZW handles a wide variety of repetitive patterns well.
- LZW is reasonably fast for both compression and decompression.
- LZW does not require floating point software or hardware.

- LZW is lossless. All information is preserved. But if noise or information is removed from an image, perhaps by smoothing or zeroing some low-order bitplanes, LZW compresses images to a smaller size. Thus, 5-bit, 6-bit, or 7-bit data masquerading as 8-bit data compresses better than true 8-bit data. Smooth images also compress better than noisy images, and simple images compress better than complex images.
- LZW works quite well on bilevel images, too. On our test images, it almost always beat PackBits and generally tied CCITT 1D (Modified Huffman) compression. LZW also handles halftoned data better than most bilevel compression schemes.

The Algorithm

Each strip is compressed independently. We strongly recommend that RowsPerStrip be chosen such that each strip contains about 8K bytes before compression. We want to keep the strips small enough so that the compressed and uncompressed versions of the strip can be kept entirely in memory, even on small machines, but are large enough to maintain nearly optimal compression ratios.

The LZW algorithm is based on a translation table, or string table, that maps strings of input characters into codes. The TIFF implementation uses variable-length codes, with a maximum code length of 12 bits. This string table is different for every strip and does not need to be reatained for the decompressor. The trick is to make the decompressor automatically build the same table as is built when the data is compressed. We use a C-like pseudocode to describe the coding scheme:

```
InitializeStringTable();
WriteCode(ClearCode);
Ω = the empty string;
for each character in the strip {
        K = GetNextCharacter();
        if Ω+K is in the string table {
                Ω = Ω+K; /* string concatenation */
        } else {
                WriteCode (CodeFromString(Ω));
                AddTableEntry(Ω+K);
                Ω = K;
        }
} /* end of for loop */
WriteCode (CodeFromString(Ω));
WriteCode (EndOfInformation);
```

That's it. The scheme is simple, although it is challenging to implement efficiently. But we need a few explanations before we go on to decompression.

The "characters" that make up the LZW strings are bytes containing TIFF uncompressed (Compression=1) image data, in our implementation. For example, if BitsPerSample is 4, each 8-bit LZW character will contain two 4-bit pixels. If BitsPerSample is 16, each 16-bit pixel will span two 8-bit LZW characters.

It is also possible to implement a version of LZW in which the LZW character depth equals BitsPerSample, as described in Draft 2 of Revision 5.0. But there is a major problem with this approach. If BitsPerSample is greater than 11, we can not

use 12-bit-maximum codes and the resulting LZW table is unacceptably large. Fortunately, due to the adaptive nature of LZW, we do not pay a significant compression ratio penalty for combining several pixels into one byte before compressing. For example, our 4-bit sample images compressed about 3 percent worse, and our 1-bit images compressed about 5 percent better. And it is easier to write an LZW compressor that always uses the same character depth than it is to write one that handles varying depths.

We can now describe some of the routine and variable references in our pseudocode:

InitializeStringTable() initializes the string table to contain all possible single-character strings. There are 256 of them, numbered 0 through 255, since our characters are bytes.

WriteCode() writes a code to the output stream. The first code written is a ClearCode, which is defined to be code #256.

Ω is our "prefix string."

GetNextCharacter() retrieves the next character value from the input stream. This will be a number between 0 and 255 because our characters are bytes.

The "+" signs indicate string concatenation.

AddTableEntry() adds a table entry. (InitializeStringTable() has already put 256 entries in our table. Each entry consists of a single-character string, and its associated code value, which, in our application, is identical to the character itself. That is, the 0th entry in our table consists of the string <0>, with a corresponding code value of <0>, the 1st entry in the table consists of the string <1>, with a corresponding code value of <1> and the 255th entry in our table consists of the string <255>, with a corresponding code value of <255>.) So, the first entry that added to our string table will be at position 256, right? Well, not quite, because we reserve code #256 for a special "Clear" code. We also reserve code #257 for a special "EndOfInformation" code that we write out at the end of the strip. So the first multiple-character entry added to the string table will be at position 258.

For example, suppose we have input data that looks like this:

Pixel 0:<7>

Pixel 1:<7>

Pixel 2:<7>

Pixel 3:<8>

Pixel 4:<8>

Pixel 5:<7>

Pixel 6:<7>

Pixel 7:<6>

Pixel 8:<6>

First, we read Pixel 0 into K. ΩK is then simply <7>, because Ω is an empty string at this point. Is the string <7> already in the string table? Of course, because all single character strings were put in the table by InitializeStringTable(). So set Ω equal to <7>, and then go to the top of the loop.

Read Pixel 1 into K. Does ΩK (<7><7>) exist in the string table? No, so we write the code associated with Ω to output (write <7> to output) and add ΩK (<7><7>) to the table as entry 258. Store K (<7>) into Ω. Note that although we have added the string consisting of Pixel 0 and Pixel 1 to the table, we "re-use" Pixel 1 as the beginning of the next string.

Back at the top of the loop, we read Pixel 2 into K. Does ΩK (<7><7>) exist in the string table? Yes, the entry we just added, entry 258, contains exactly <7><7>. So we add K to the end of Ω so that Ω is now <7><7>.

Back at the top of the loop, we read Pixel 3 into K. Does ΩK (<7><7><8>) exist in the string table? No, so we write the code associated with Ω (<258>) to output and then add ΩK to the table as entry 259. Store K (<8>) into Ω.

Back at the top of the loop, we read Pixel 4 into K. Does ΩK (<8><8>) exist in the string table? No, so we write the code associated with Ω (<8>) to output and then add ΩK to the table as entry 260. Store K (<8>) into Ω.

Continuing, we get the following results:

After reading:	We write to output:	And add table entry:
Pixel 0		
Pixel 1	<7>	258: <7><7>
Pixel 2		
Pixel 3	<258>	259: <7><7><8>
Pixel 4	<8>	260:<8><8>
Pixel 5	<8>	261: <8><7>
Pixel 6		
Pixel 7	<258>	262: <7><7><6>
Pixel 8	<6>	263: <6><6>

WriteCode() also requires some explanation. In our example, the output code stream, <7><258><8><8><258><6> should be written using as few bits as possible. When we are just starting out, we can use 9-bit codes, since our new string table entries are greater than 255 but less than 512. *After adding table entry 511, switch to 10-bit codes (i.e., entry 512 should be a 10-bit code.) Likewise, switch to 11-bit codes after table entry 1023, and 12-bit codes after table entry 2047.* We will arbitrarily limit ourselves to 12-bit codes, so that our table can have at most 4096 entries. The table should not be any larger.

Whenever you add a code to the output stream, it "counts" toward the decision about bumping the code bit length. This is important when writing the last code word before an EOI code or ClearCode, to avoid code length errors.

What happens if we run out of room in our string table? This is where the ClearCode comes in. As soon as we use entry 4094, we write out a (12-bit) ClearCode. (If we wait any longer to write the ClearCode, the decompressor might try to interpret the ClearCode as a 13-bit code.) At this point, the compressor reinitializes the string table and then writes out 9-bit codes again.

Note that whenever you write a code and add a table entry, Ω is not left empty. It contains exactly one character. Be careful not to lose it when you write an end-of-table ClearCode. You can either write it out as a 12-bit code before writing the ClearCode, in which case you need to do it right after adding table entry 4093, or

you can write it as a 9-bit code after the ClearCode . Decompression gives the same result in either case.

To make things a little simpler for the decompressor, we will require that each strip begins with a ClearCode and ends with an EndOfInformation code. Every LZW-compressed strip must begin on a byte boundary. It need not begin on a word boundary. LZW compression codes are stored into bytes in high-to-low-order fashion, i.e., FillOrder is assumed to be 1. The compressed codes are written as bytes (not words) so that the compressed data will be identical whether it is an 'II' or 'MM' file.

Note that the LZW string table is a continuously updated history of the strings that have been encountered in the data. Thus, it reflects the characteristics of the data, providing a high degree of adaptability.

LZW Decoding

The procedure for decompression is a little more complicated:

```
while ((Code = GetNextCode()) != EoiCode) {
        if (Code == ClearCode) {
                InitializeTable();
                Code = GetNextCode();
                if (Code == EoiCode)
                        break;
                WriteString(StringFromCode(Code));
                OldCode = Code;
        } /* end of ClearCode case */
        else {
                if (IsInTable(Code)) {
                        WriteString(StringFromCode(Code));
                        AddStringToTable(StringFromCode(OldCode
)+FirstChar(StringFromCode(Code)));
                        OldCode = Code;
                } else {
                        OutString = StringFromCode(OldCode) +
FirstChar(StringFromCode(OldCode));
                        WriteString(OutString);
                        AddStringToTable(OutString);
                        OldCode = Code;
                }
        } /* end of not-ClearCode case */
} /* end of while loop */
```

The function GetNextCode() retrieves the next code from the LZW-coded data. It must keep track of bit boundaries. It knows that the first code that it gets will be a 9-bit code. We add a table entry each time we get a code. So, GetNextCode() must switch over to 10-bit codes as soon as string *#510* is stored into the table. *Similarly, the switch is made to 11-bit codes after #1022 and to 12-bit codes after #2046.*

The function StringFromCode() gets the string associated with a particular code from the string table.

The function AddStringToTable() adds a string to the string table. The "+" sign joining the two parts of the argument to AddStringToTable indicates string concatenation.

StringFromCode() looks up the string associated with a given code.

WriteString() adds a string to the output stream.

When SamplesPerPixel Is Greater Than 1

So far, we have described the compression scheme as if SamplesPerPixel were always 1, as is the case with palette-color and grayscale images. But what do we do with RGB image data?

Tests on our sample images indicate that the LZW compression ratio is nearly identical whether PlanarConfiguration=1 or PlanarConfiguration=2, for RGB images. So, use whichever configuration you prefer and simply compress the bytes in the strip.

Note: Compression ratios on our test RGB images were disappointingly low: between 1.1 to 1 and 1.5 to 1, depending on the image. Vendors are urged to do what they can to remove as much noise as possible from their images. Preliminary tests indicate that significantly better compression ratios are possible with less-noisy images. Even something as simple as zeroing-out one or two least-significant bitplanes can be effective, producing little or no perceptible image degradation.

Implementation

The exact structure of the string table and the method used to determine if a string is already in the table are probably the most significant design decisions in the implementation of a LZW compressor and decompressor. Hashing has been suggested as a useful technique for the compressor. We have chosen a tree-based approach, with good results. The decompressor is more straightforward and faster because no search is involved—strings can be accessed directly by code value.

LZW Extensions

Some images compress better using LZW coding if they are first subjected to a process wherein each pixel value is replaced by the difference between the pixel and the preceding pixel. See the following Section.

Acknowledgments

See the first page of this section for the LZW reference.

The use of ClearCode as a technique for handling overflow was borrowed from the compression scheme used by the Graphics Interchange Format (GIF), a small-color-paint-image-file format used by CompuServe that also uses an adaptation of the LZW technique.

Section 14: Differencing Predictor

CDB-compliant Tiff readers do not consider Differencing Predictor compressed TIFF data. As a result, section 14 is not applicable to CDB-compliant TIFF readers.

This section defines a Predictor that greatly improves compression ratios for some images.

Predictor

Tag = 317 (13D.H)

Type = SHORT

N = 1

A predictor is a mathematical operator that is applied to the image data before an encoding scheme is applied. Currently this field is used only with LZW (Compression=5) encoding because LZW is probably the only TIFF encoding scheme that benefits significantly from a predictor step. See Section 13.

The possible values are:

1 = No prediction scheme used before coding.

2 = Horizontal differencing.

Default is 1.

The algorithm

Make use of the fact that many continuous-tone images rarely vary much in pixel value from one pixel to the next. In such images, if we replace the pixel values by differences between consecutive pixels, many of the differences should be 0, plus or minus 1, and so on. This reduces the apparent information content and allows LZW to encode the data more compactly.

Assuming 8-bit grayscale pixels for the moment, a basic C implementation might look something like this:

```
char    image[ ][ ];
int     row, col;

/* take horizontal differences:
 */
for (row = 0; row < nrows; row++)
        for (col = ncols - 1; col >= 1; col--)
                image[row][col] -= image[row][col-1];
```

If we don't have 8-bit components, we need to work a little harder to make better use of the architecture of most CPUs. Suppose we have 4-bit components packed two per byte in the normal TIFF uncompressed (i.e., Compression=1) fashion. To find differences, we want to first expand each 4-bit component into an 8-bit byte, so that we have one component per byte, low-order justified. We then perform the horizontal differencing illustrated in the example above. Once the differencing has been completed, we then repack the 4-bit differences two to a byte, in the normal TIFF uncompressed fashion.

If the components are greater than 8 bits deep, expanding the components into 16-bit words instead of 8-bit bytes seems like the best way to perform the subtraction on most computers.

Note that we have not lost any data up to this point, nor will we lose any data later on. It might seem at first that our differencing might turn 8-bit components into 9-bit differences, 4-bit components into 5-bit differences, and so on. But it turns out that we can completely ignore the "overflow" bits caused by subtracting a larger number from a smaller number and still reverse the process without error. Normal two's complement arithmetic does just what we want. Try an example by hand if you need more convincing.

Up to this point we have implicitly assumed that we are compressing bilevel or grayscale images. An additional consideration arises in the case of color images.

If PlanarConfiguration is 2, there is no problem. Differencing works the same as it does for grayscale data.

If PlanarConfiguration is 1, however, things get a little trickier. If we didn't do anything special, we would subtract red component values from green component values, green component values from blue component values, and blue component values from red component values. This would not give the LZW coding stage much redundancy to work with. So, we will do our horizontal differences with an offset of SamplesPerPixel (3, in the RGB case). In other words, we will subtract red from red, green from green, and blue from blue. The LZW coding stage is identical to the SamplesPerPixel=1 case. We require that BitsPerSample be the same for all 3 components.

Results and Guidelines

LZW without differencing works well for 1-bit images, 4-bit grayscale images, and many palette-color images. But natural 24-bit color images and some 8-bit grayscale images do much better with differencing.

Although the combination of LZW coding with horizontal differencing does not result in any loss of data, it may be worthwhile in some situations to give up some information by removing as much noise as possible from the image data before doing the differencing, especially with 8-bit components. The simplest way to get rid of noise is to mask off one or two low-order bits of each 8-bit component. On our 24-bit test images, LZW with horizontal differencing yielded an average compression ratio of 1.4 to 1. When the low-order bit was masked from each component, the compression ratio climbed to 1.8 to 1; the compression ratio was 2.4 to 1 when masking two bits, and 3.4 to 1 when masking three bits. Of course, the more you mask, the more you risk losing useful information along with the noise. We encourage you to experiment to find the best compromise for your device. For some applications, it may be useful to let the user make the final decision.

Incidentally, we tried taking both horizontal and vertical differences, but the extra complexity of two-dimensional differencing did not appear to pay off for most of our test images. About one third of the images compressed slightly better with two-dimensional differencing, about one third compressed slightly worse, and the rest were about the same.

Section 15: Tiled Images

Introduction

Motivation

This section describes how to organize images into tiles instead of strips.

For low-resolution to medium-resolution images, the standard TIFF method of breaking the image into strips is adequate. However high-resolution images can be accessed more efficiently—and compression tends to work better—if the image is broken into roughly square tiles instead of horizontally-wide but vertically-narrow strips.

Relationship to existing fields

When the tiling fields described below are used, they replace the StripOffsets, StripByteCounts, and RowsPerStrip fields. Use of tiles will therefore cause older TIFF readers to give up because they will have no way of knowing where the image data is or how it is organized. **Do not** use both strip-oriented and tile-oriented fields in the same TIFF file.

Padding

Tile size is defined by TileWidth and TileLength. All tiles in an image are the same size; that is, they have the same pixel dimensions.

Boundary tiles are padded to the tile boundaries. For example, if TileWidth is 64 and ImageWidth is 129, then the image is 3 tiles wide and 63 pixels of padding must be added to fill the rightmost column of tiles. The same holds for TileLength and ImageLength. It doesn't matter what value is used for padding, because good TIFF readers display only the pixels defined by ImageWidth and ImageLength and ignore any padded pixels. Some compression schemes work best if the padding is accomplished by replicating the last column and last row instead of padding with 0's.

The price for padding the image out to tile boundaries is that some space is wasted. But compression usually shrinks the padded areas to almost nothing. Even if data is not compressed, remember that tiling is intended for large images. Large images have lots of comparatively small tiles, so that the percentage of wasted space will be very small, generally on the order of a few percent or less.

The advantages of padding an image to the tile boundaries are that implementations can be simpler and faster and that it is more compatible with tile-oriented compression schemes such as JPEG. See Section 22.

Tiles are compressed individually, just as strips are compressed. *That is, each row of data in a tile is treated as a separate "scanline" when compressing.* Compres-

sion includes any padded areas of the rightmost and bottom tiles so that all the tiles in an image are the same size when uncompressed.

All of the following fields are required for tiled images:

Fields

TileWidth

Tag = 322 (142.H)

Type = SHORT or LONG

N = 1

The tile width in pixels. This is the number of columns in each tile.

Assuming integer arithmetic, three computed values that are useful in the following field descriptions are:

TilesAcross = (ImageWidth + TileWidth - 1) / TileWidth

TilesDown = (ImageLength + TileLength - 1) / TileLength

TilesPerImage = TilesAcross * TilesDown

These computed values are not TIFF fields; they are simply values determined by the ImageWidth, TileWidth, ImageLength, and TileLength fields.

TileWidth and ImageWidth together determine the number of tiles that span the width of the image (TilesAcross). TileLength and ImageLength together determine the number of tiles that span the length of the image (TilesDown).

We recommend choosing TileWidth and TileLength such that the resulting tiles are about 4K to 32K bytes before compression. This seems to be a reasonable value for most applications and compression schemes.

TileWidth must be a multiple of 16. This restriction improves performance in some graphics environments and enhances compatibility with compression schemes such as JPEG.

Tiles need not be square.

Note that ImageWidth can be less than TileWidth, although this means that the tiles are too large or that you are using tiling on really small images, neither of which is recommended. The same observation holds for ImageLength and TileLength.

No default. See also TileLength, TileOffsets, TileByteCounts.

TileLength

Tag = 323 (143.H)

Type = SHORT or LONG

N = 1

The tile length (height) in pixels. This is the number of rows in each tile.

TileLength must be a multiple of 16 for compatibility with compression schemes such as JPEG.

Replaces RowsPerStrip in tiled TIFF files.

No default. See also TileWidth, TileOffsets, TileByteCounts.

TileOffsets

Tag = 324 (144.H)

Type = LONG

N = TilesPerImage for PlanarConfiguration = 1

= SamplesPerPixel * TilesPerImage for PlanarConfiguration = 2

For each tile, the byte offset of that tile, as compressed and stored on disk. The offset is specified with respect to the beginning of the TIFF file. Note that this implies that each tile has a location independent of the locations of other tiles.

Offsets are ordered left-to-right and top-to-bottom. For PlanarConfiguration = 2, the offsets for the first component plane are stored first, followed by all the offsets for the second component plane, and so on.

No default. See also TileWidth, TileLength, TileByteCounts.

TileByteCounts

Tag = 325 (145.H)

Type = SHORT or LONG

N = TilesPerImage for PlanarConfiguration = 1

= SamplesPerPixel * TilesPerImage for PlanarConfiguration = 2

For each tile, the number of (compressed) bytes in that tile.

See TileOffsets for a description of how the byte counts are ordered.

No default. See also TileWidth, TileLength, TileOffsets.

Section 16: CMYK Images

Motivation

CDB-compliant TIFF readers do not consider CMYK encoded color image TIFF image data. As a result, section 16 is not applicable to CDB-compliant TIFF

This section describes how to store separated (usually CMYK) image data in a TIFF file.

In a separated image, each pixel consists of N components. Each component represents the amount of a particular ink that is to be used to represent the image at that location, typically using a halftoning technique.

For example, in a CMYK image, each pixel consists of 4 components. Each component represents the amount of cyan, magenta, yellow, or black process ink that is to be used to represent the image at that location.

The fields described in this section can be used for more than simple 4-color process (CMYK) printing. They can also be used for describing an image made up of more than 4 inks, such an image made up of a cyan, magenta, yellow, red, green, blue, and black inks. Such an image is sometimes called a high-fidelity image and has the advantage of slightly extending the printed color gamut.

Since separated images are quite device-specific and are restricted to color prepress use, they should not be used for general image data interchange. Separated images are to be used only for prepress applications in which the imagesetter, paper, ink, and printing press characteristics are known by the creator of the separated image.

Note: there is no single method of converting RGB data to CMYK data and back. In a perfect world, something close to cyan = 255-red, magenta = 255-green, and yellow = 255-blue should work; but characteristics of printing inks and printing presses, economics, and the fact that the meaning of RGB itself depends on other parameters combine to spoil this simplicity.

Requirements

In addition to satisfying the normal Baseline TIFF requirements, a separated TIFF file must have the following characteristics:

- **SamplesPerPixel = N.** SHORT. The number of inks. (For example, N=4 for CMYK, because we have one component each for cyan, magenta, yellow, and black.)
- **BitsPerSample = 8,8,8,8 (for CMYK).** SHORT. For now, only 8-bit components are recommended. The value "8" is repeated SamplesPerPixel times.
- **PhotometricInterpretation = 5 (Separated - usually CMYK).** SHORT. The components represent the desired percent dot coverage of each ink, where the larger component values represent a higher percentage of ink dot coverage and smaller values represent less coverage.

Fields

In addition, there are some new fields, all of which are optional.

InkSet

Tag = 332 (14C.H)

Type = SHORT

N = 1

The set of inks used in a separated (PhotometricInterpretation=5) image.

1 = CMYK. The order of the components is cyan, magenta, yellow, black. Usually, a value of 0 represents 0% ink coverage and a value of 255 represents 100% ink coverage for that component, but see DotRange below. The InkNames field should not exist when InkSet=1.

2 = not CMYK. See the InkNames field for a description of the inks to be used.

Default is 1 (CMYK).

NumberOfInks

Tag = 334 (14E.H)

Type = SHORT

N = 1

The number of inks. Usually equal to SamplesPerPixel, unless there are extra samples.

See also ExtraSamples.

Default is 4.

InkNames

Tag = 333 (14D.H)

Type = ASCII

N = total number of characters in all the ink name strings, including the NULs.

The name of each ink used in a separated (PhotometricInterpretation=5) image, written as a list of concatenated, NUL-terminated ASCII strings. The number of strings must be equal to NumberOfInks.

The samples are in the same order as the ink names.

See also InkSet, NumberOfInks.

No default.

DotRange

Tag = 336 (150.H)

Type = BYTE or SHORT

N = 2, or 2*SamplesPerPixel

The component values that correspond to a 0% dot and 100% dot. DotRange[0] corresponds to a 0% dot, and DotRange[1] corresponds to a 100% dot.

If a DotRange pair is included for each component, the values for each component are stored together, so that the pair for Cyan would be first, followed by the pair for Magenta, and so on. *Use of multiple dot ranges is, however, strongly discouraged in the interests of simplicity and compatibility with ANSI IT8 standards.*

A number of prepress systems like to keep some "headroom" and "footroom" on both ends of the range. What to do with components that are less than the 0% aim point or greater than the 100% aim point is not specified and is application-dependent.

It is strongly recommended that a CMYK TIFF writer not attempt to use this field to reverse the sense of the pixel values so that smaller values mean more ink instead of less ink. That is, DotRange[0] should be less than DotRange[1].

DotRange[0] and DotRange[1] must be within the range [0, (2**BitsPerSample) - 1].

Default: a component value of 0 corresponds to a 0% dot, and a component value of 255 (assuming 8-bit pixels) corresponds to a 100% dot. That is, DotRange[0] = 0 and DotRange[1] = (2**BitsPerSample) - 1.

TargetPrinter

Tag = 337 (151.H)

Type = ASCII

N = any

A description of the printing environment for which this separation is intended.

History

This Section has been expanded from earlier drafts, with the addition of the **InkSet, InkNames, NumberOfInks, DotRange**, and **TargetPrinter**, but is backward-compatible with earlier draft versions.

Possible future enhancements: definition of the characterization information so that the CMYK data can be retargeted to a different printing environment and so that display on a CRT or proofing device can more accurately represent the color. ANSI IT8 is working on such a proposal.

Section 17: HalftoneHints

CDB-compliant TIFF readers do not consider any of the TIFF tags related to halftone hints. As a result, section 17 is not applicable to CDB-compliant TIFF

This section describes a scheme for properly placing highlights and shadows in halftoned images.

Introduction

The single most easily recognized failing of continuous tone images is the incorrect placement of highlight and shadow. It is critical that a halftone process be capable of printing the lightest areas of the image as the smallest halftone spot capable of the output device, at the specified printer resolution and screen ruling. Specular highlights (small ultra-white areas) as well as the shadow areas should be printable as paper only.

Consistency in highlight and shadow placement allows the user to obtain predictable results on a wide variety of halftone output devices. Proper implementation of theHalftoneHints field will provide a significant step toward device independent imaging, such that low cost printers may to be used as effective proofing devices for images which will later be halftoned on a high-resolution imagesetter.

The HalftoneHints Field

HalftoneHints

Tag = 321 (141.H)

Type = SHORT

N = 2

The purpose of the HalftoneHints field is to convey to the halftone function the range of gray levels within a colorimetrically-specified image that should retain tonal detail. The field contains two values of sixteen bits each and, therefore, is contained wholly within the field itself; no offset is required. The first word specifies the highlight gray level which should be halftoned at the lightest printable tint of the final output device. The second word specifies the shadow gray level which should be halftoned at the darkest printable tint of the final output device. Portions of the image which are whiter than the highlight gray level will quickly, if not immediately, fade to specular highlights. There is no default value specified, since the highlight and shadow gray levels are a function of the subject matter of a particular image.

Appropriate values may be derived algorithmically or may be specified by the user, either directly or indirectly.

The HalftoneHints field, as defined here, defines an achromatic function. It can be used just as effectively with color images as with monochrome images. When used with opponent color spaces such as CIE L*a*b* or YCbCr, it refers to the achromatic component only; L* in the case of CIELab, and Y in the case of

YCbCr. When used with tri-stimulus spaces such as RGB, it suggests to retain tonal detail for all colors with an NTSC gray component within the bounds of the R=G=B=Highlight to R=G=B=Shadow range.

Comments for TIFF Writers

TIFF writers are encouraged to include the HalftoneHints field in all color or grayscale images where BitsPerSample >1. Although no default value is specified, prior to the introduction of this field it has been common practice to implicitly specify the highlight and shadow gray levels as 1 and 2**BitsperSample-2 and manipulate the image data to this definition. There are some disadvantages to this technique, and it is not feasible for a fixed gamut colorimetric image type. Appropriate values may be derived algorithmically or may be specified by the user directly or indirectly. Automatic algorithms exist for analyzing the histogram of the achromatic intensity of an image and defining the minimum and maximum values as the highlight and shadow settings such that tonal detail is retained throughout the image. This kind of algorithm may try to impose a highlight or shadow where none really exists in the image, which may require user controls to override the automatic setting.

It should be noted that the choice of the highlight and shadow values is somewhat output dependent. For instance, in situations where the dynamic range of the output medium is very limited (as in newsprint and, to a lesser degree, laser output), it may be desirable for the user to clip some of the lightest or darkest tones to avoid the reduced contrast resulting from compressing the tone of the entire image. Different settings might be chosen for 150-line halftone printed on coated stock. Keep in mind that these values may be adjusted later (which might not be possible unless the image is stored as a colorimetric, fixed, full-gamut image), and that more sophisticated page-layout applications may be capable of presenting a user interface to consider these decisions at a point where the halftone process is well understood.

It should be noted that although CCDs are linear intensity detectors, TIFF writers may choose to manipulate the image to store gamma-compensated data. Gamma-compensated data is more efficient at encoding an image than is linear intensity data because it requires fewer BitsPerPixel to eliminate banding in the darker tones. It also has the advantage of being closer to the tone response of the display or printer and is, therefore, less likely to produce poor results from applications that are not rigorous about their treatment of images. Be aware that the PhotometricInterpretation value of 0 or 1 (grayscale) implies linear data because no gamma is specified. The PhotometricInterpretation value of 2 (RGB data) specifies the NTSC gamma of 2.2 as a default. If data is written as something other than the default, then a GrayResponseCurve field or a TransferFunction field must be present to define the deviation. For grayscale data, be sure that the densities in the GrayResponseCurve are consistent with the PhotometricInterpretation field and the HalftoneHints field.

Comments for TIFF Readers

TIFF readers that send a grayscale image to a halftone output device, whether it is a binary laser printer or a PostScript imagesetter should make an effort to maintain the highlight and shadow placement. This requires two steps. First, determine the highlight and shadow gray level of a particular image. Second, communicate that information to the halftone engine.

To determine the highlight and shadow gray levels, begin by looking for a HalftoneHints field. If it exists, it takes precedence. The first word represents the gray level of the highlight and the second word represents the gray level of the shadow. If the image is a colorimetric image (i.e. it has a GrayResponseCurve field or a TransferFunction field) but does not contain a HalftoneHints field, then the gamut mapping techniques described earlier should be used to determine the highlight and shadow values. If neither of these conditions are true, then the file should be treated as if a HalftoneHints field had indicated a highlight at gray level 1 and a shadow at gray level 2**BitsPerPixel-2 (or vice-versa depending on the PhotometricInterpretation field). Once the highlight and shadow gray levels have been determined, the next step is to communicate this information to the halftone module. The halftone module may exist within the same application as the TIFF reader, it may exist within a separate printer driver, or it may exist within the Raster Image Processor (RIP) of the printer itself. Whether the halftone process is a simple dither pattern or a general purpose spot function, it has some gray level at which the lightest printable tint will be rendered. The HalftoneHint concept is best implemented in an environment where this lightest printable tint is easily and consistently specified.

There are several ways in which an application can communicate the highlight and shadow to the halftone function. Some environments may allow the application to pass the highlight and shadow to the halftone module explicitly along with the image. This is the best approach, but many environments do not yet provide this capability. Other environments may provide fixed gray levels at which the highlight and shadow will be rendered. For these cases, the application should build a tone map that matches the highlight and shadow specified in the image to the highlight and shadow gray level of the halftone module. This approach requires more work by the application software, but will provide excellent results. Some environments will not have any consistent concept of highlight and shadow at all. In these environments, the best an application can do is characterize each of the supported printers and save the observed highlight and shadow gray levels. The application can then use these values to achieve the desired results, providing the environment doesn't change.

Once the highlight and shadow areas are selected, care should be taken to appropriately map intermediate gray levels to those expected by the halftone engine, which may or may not be linear Reflectance. Note that although CCDs are linear intensity detectors and many TIFF files are stored as linear intensity, most output devices require significant tone compensation (sometimes called gamma correction) to correctly display or print linear data. Be aware that the PhotometricInterpretation value of 0, 1 implies linear data because no gamma is specified. The PhotometricInterpretation value of 2 (RGB data) specifies the NTSC gamma of 2.2 as a default. If a GrayResponseCurve field or a TransferFunction field is present, it may define something other than the default.

Some Background on the Halftone Process

To obtain the best results when printing a continuous-tone raster image, it is seldom desirable to simply reproduce the tones of the original on the printed page. Most often there is some gamut mapping required. Often this is because the tonal range of the original extends beyond the tonal range of the output medium. In some cases, the tone range of the original is within the gamut of the output medium, but it may be more pleasing to expand the tone of the image to fill the range of the output. Given that the tone of the original is to be adjusted, there is a whole range of possibilities for the level of sophistication that may be undertaken by a software application.

Printing monochrome output is far less sophisticated than printing color output. For monochrome output the first priority is to control the placement of the highlight and the shadow. Ideally, a quality halftone will have sufficient levels of gray so that a standard observer cannot distinguish the interface between any two adjacent levels of gray. In practice, however, there is often a significant step between the tone of the paper and the tone of the lightest printable tint. Although usually less severe, the problem is similar between solid ink and the darkest printable tint. Since the dynamic range between the lightest printable tint and the darkest printable tint is usually less than one would like, it is common to maximize the tone of the image within these bounds. Not all images will have a highlight (an area of the image which is desirable to print as light as possible while still retaining tonal detail). If one exists, it should be carefully controlled to print at the lightest printable tint of the output medium. Similarly, the darkest areas of the image to retain tonal detail should be printed as the darkest printable tint of the output medium. Tones lighter or darker than these may be clipped at the limits of the paper and ink. Satisfactory results may be obtained in monochrome work by doing nothing more than a perceptually-linear mapping of the image between these rigorously controlled endpoints. This level of sophistication is sufficient for many mid-range applications, although the results often appear flatter (i.e. lower in contrast) than desired.

The next step is to increase contrast slightly in the tonal range of the image that contains the most important subject matter. To perform this step well requires considerably more information about the image and about the press. To know where to add contrast, the algorithm must have access to first the keyness of the image; the tone range which the user considers most important. To know how much contrast to add, the algorithm must have access to the absolute tone of the original and the dynamic range of the output device so that it may calculate the amount of tone compression to which the image is actually subjected.

Most images are called normal key. The important subject areas of a normal key image are in the midtones. These images do well when a so-called “sympathetic curve” is applied, which increases the contrast in midtones slightly at the expense of contrast in the lighter and darker tones. White china on a white tablecloth is an example of a high key image. High key images benefit from higher contrast in lighter tones, with less contrast needed in the midtones and darker tones. Low key images have important subject matter in the darker tones and benefit from increasing the contrast in the darker tones. Specifying the keyness of an image might be attempted by automatic techniques, but it will likely fail without user input. For example, a photo of a bride in a white wedding dress it may be a high key image if

you are selling wedding dresses, but may be a normal key image if you are the parents of the bride and are more interested in her smile.

Sophisticated color reproduction employs all of these principles, and then applies them in three dimensions. The mapping of the highlight and shadow becomes only one small, albeit critical, portion of the total issue of mapping colors that are too saturated for the output medium. Here again, automatic techniques may be employed as a first pass, with the user becoming involved in the clip or compress mapping decision. The HalftoneHints field is still useful in communicating which portions of the intensity of the image must be retained and which may be clipped. Again, a sophisticated application may override these settings if later user input is received.

Section 18: Associated Alpha Handling

This section describes a scheme for handling images with alpha data.

Introduction

A common technique in computer graphics is to assemble an image from one or more elements that are rendered separately. When elements are combined using compositing techniques, matte or coverage information must be present for each pixel to create a properly anti-aliased accumulation of the full image [Porter84]. This matting information is an example of additional per-pixel data that must be maintained with an image. This section describes how to use the ExtraSamples field to store the requisite matting information, commonly called the associated alpha or just alpha. This scheme enables efficient manipulation of image data during compositing operations.

Images with matting information are stored in their natural format but with an additional component per pixel. The ExtraSample field is included with the image to indicate that an extra component of each pixel contains associated alpha data. In addition, when associated alpha data are included with RGB data, the RGB components must be stored premultiplied by the associated alpha component and component values in the range [0,2**BitsPerSample-1] are implicitly mapped onto the [0,1] interval. That is, for each pixel (r,g,b) and opacity A, where r, g, b, and A are in the range [0,1], (A*r,A*g,A*b,A) must be stored in the file. If A is zero, then the color components should be interpreted as zero. Storing data in this pre-multiplied format, allows compositing operations to be implemented most efficiently. In addition, storing pre-multiplied data makes it possible to specify colors with components outside the normal [0,1] interval. The latter is useful for defining certain operations that effect only the luminescence [Porter84].

Fields

ExtraSamples

Tag = 338 (152.H)

Type = SHORT

N = 1

This field must have a value of 1 (associated alpha data with pre-multiplied color components). The associated alpha data stored in component SamplesPerPixel-1 of each pixel contains the opacity of that pixel, and the color information is pre-multiplied by alpha.

Comments

Associated alpha data is just another component added to each pixel. Thus, for example, its size is defined by the value of the BitsPerSample field.

Note that since data is stored with RGB components already multiplied by alpha, naive applications that want to display an RGBA image on a display can do so simply by displaying the RGB component values. This works because it is effectively the same as merging the image with a black background. That is, to merge one image with another, the color of resultant pixels are calculated as:

$$C_r = C_{over} * A_{over} + C_{under} * (1 - A_{over})$$

Since the "under image" is a black background, this equation reduces to

$$C_r = C_{over} * A_{over}$$

which is exactly the pre-multiplied color; i.e. what is stored in the image.

On the other hand, to print an RGBA image, one must composite the image over a suitable background page color. For a white background, this is easily done by adding 1 - A to each color component. For an arbitrary background color C_{back}, the *printed color* of each pixel is

$$C_{print} = C_{image} + C_{back} * (1 - A_{image})$$

(since C_{image} is pre-multiplied).

Since the ExtraSamples field is independent of other fields, this scheme permits alpha information to be stored in whatever organization is appropriate. In particular, components can be stored packed (PlanarConfiguration=1); this is important for good I/O performance and for good memory access performance on machines that are sensitive to data locality. However, if this scheme is used, TIFF readers must not derive the SamplesPerPixel from the value of the PhotometricInterpretation field (e.g., if RGB, then SamplesPerPixel is 3).

In addition to being independent of data storage-related fields, the field is also independent of the PhotometricInterpretation field. This means, for example, that it is easy to use this field to specify grayscale data and associated matte information. Note that a palette-color image with associated alpha will not have the colormap indices pre-multiplied; rather, the RGB colormap values will be pre-multiplied.

Unassociated Alpha and Transparency Masks

CDB-compliant TIFF readers do not consider unassociated alpha image data.

Some image manipulation applications support notions of transparency masks and soft-edge masks. The associated alpha information described in this section is different from this *unassociated alpha* information in many ways, most importantly:

- Associated alpha describes opacity or coverage at each pixel, while clipping-related alpha information describes a boolean relationship. That is, associated alpha can specify fractional coverage at a pixel, while masks specify either 0 or 100 percent coverage.
- Once defined, associated alpha is not intended to be removed or edited, except as a result of compositing the image; it is an integral part of an image.

Unassociated alpha, on the other hand, is designed as an ancillary piece of information.

References

[Porter84] "Compositing Digital Images". Thomas Porter, Tom Duff; Lucasfilm Ltd. ACM SIGGRAPH Proceedings Volume 18, Number 3. July, 1984.

Section 19: Data Sample Format

This section describes a scheme for specifying data sample type information.

TIFF implicitly types all data samples as unsigned integer values. Certain applications, however, require the ability to store image-related data in other formats such as floating point. This section presents a scheme for describing a variety of data sample formats.

Fields

The CDB specification establishes the conventions that govern the SampleFormat of TIFF image and DEM data. As a result, CDB-compliant TIFF readers do not consider image and DEM data when the value of the SampleFormat tag does not conform to CDB conventions.

SampleFormat

Tag = 339 (153.H)

Type = SHORT

N = SamplesPerPixel

This field specifies how to interpret each data sample in a pixel. Possible values are:

1 = unsigned integer data

2 = two's complement signed integer data

3 = IEEE floating point data [IEEE]

4 = undefined data format

Note that the SampleFormat field does not specify the size of data samples; this is still done by the BitsPerSample field.

A field value of "undefined" is a statement by the writer that it did not know how to interpret the data samples; for example, if it were copying an existing image. A reader would typically treat an image with "undefined" data as if the field were not present (i.e. as unsigned integer data).

Default is 1, unsigned integer data.

While the above-mentioned sample data formats are possible, CDB clients expect image and DEM data to be in the format as specified in the CDB conventions and constraints.

CDB-compliant TIFF readers do not consider the SMinSampleValue TIFF tag.

SMinSampleValue

Tag = 340 (154.H)

Type = the field type that best matches the sample data

N = SamplesPerPixel

This field specifies the minimum sample value. Note that a value should be given for each data sample. That is, if the image has 3 SamplesPerPixel, 3 values must be specified.

The default for SMinSampleValue and SMaxSampleValue is the full range of the data type.

CDB-compliant TIFF readers do not consider the SMaxSampleValue TIFF tag.

SMaxSampleValue

Tag = 341 (155.H)

Type = the field type that best matches the sample data

N = SamplesPerPixel

This new field specifies the maximum sample value.

Comments

The SampleFormat field allows more general imaging (such as image processing) applications to employ TIFF as a valid file format.

SMinSampleValue and SMaxSampleValue become more meaningful when image data is typed. The presence of these fields makes it possible for readers to assume that data samples are bound to the range [SMinSampleValue, SMaxSampleValue] without scanning the image data.

References

[IEEE] "IEEE Standard 754 for Binary Floating-point Arithmetic".

Section 20: RGB Image Colorimetry

CDB-compliant TIFF readers do not consider any of the TIFF tags describes in this section.

Without additional information, RGB data is device-specific; that is, without an absolute color meaning. This section describes a scheme for describing and characterizing RGB image data.

Introduction

Color printers, displays, and scanners continue to improve in quality and availability while they drop in price. Now the problem is to display color images so that they appear to be identical on different hardware.

The key to reproducing the same color on different devices is to use the CIE 1931 XYZ color-matching functions, the international standard for color comparison. Using CIE XYZ, an image's colorimetry information can fully describe its color interpretation. The approach taken here is essentially calibrated RGB. It implies a transformation from the RGB color space of the pixels to CIE 1931 XYZ.

The appearance of a color depends not only on its absolute tristimulus values, but also on the conditions under which it is viewed, including the nature of the surround and the adaptation state of the viewer. Colors having the same absolute tristimulus values appear the same in identical viewing conditions. The more complex issue of color appearance under different viewing conditions is addressed by [4]. The colorimetry information presented here plays an important role in color appearance under different viewing conditions.

Assuming identical viewing conditions, an application using the tags described below can display an image on different hardware and achieve colorimetrically identical results. The process of using this colorimetry information for displaying an image is straightforward on a color monitor but it is more complex for color printers. Also, the results will be limited by the color gamut and other characteristics of the display or printing device.

The following fields describe the image colorimetry information of a TIFF image:

WhitePoint	chromaticity of the white point of the image
PrimaryChromaticities	chromaticities of the primaries of the image
TransferFunction	transfer function for the pixel data
TransferRange	extends the range of the transfer function
ReferenceBlackWhite	pixel component headroom and footroom parameters

The TransferFunction, TransferRange, and ReferenceBlackWhite fields have defaults based on industry standards. An image has a colorimetric interpretation if and only if both the WhitePoint and PrimaryChromaticities fields are present. An image without these colorimetry fields will be displayed in an application and hardware dependent manner.

Note: In the following definitions, BitsPerSample is used as if it were a single number when in fact it is an array of SamplesPerPixel numbers. The elements of

this array may not always be equal, for example: 5/6/5 16-bit pixels. BitsPerSample should be interpreted as the BitsPerSample value associated with a particular component. In the case of unequal BitsPerSample values, the definitions below can be extended in a straightforward manner.

This section has the following differences with Appendix H in TIFF 5.0:

- removed the use of image colorimetry defaults
- renamed the ColorResponseCurves field as TransferFunction
- optionally allowed a single TransferFunction table to describe all three channels
- described the use of the TransferFunction field for YCbCr, Palette, WhiteIsZero and BlackIsZero PhotometricInterpretation types
- added the TransferRange tag to expand the range of the TransferFunction below black and above white
- added the ReferenceBlackWhite field
- addressed the issue of color appearance

Colorimetry Field Definitions

WhitePoint

Tag = 318 (13E.H)

Type = RATIONAL

N = 2

The chromaticity of the white point of the image. This is the chromaticity when each of the primaries has its ReferenceWhite value. The value is described using the 1931 CIE xy chromaticity diagram and only the chromaticity is specified. This value can correspond to the chromaticity of the alignment white of a monitor, the filter set and light source combination of a scanner or the imaging model of a rendering package. The ordering is white[x], white[y].

For example, the CIE Standard Illuminant D65 used by CCIR Recommendation 709 and Kodak PhotoYCC is:

3127/10000,3290/10000

No default.

PrimaryChromaticities

Tag =319 (13F.H)

Type = RATIONAL

N = 6

The chromaticities of the primaries of the image. This is the chromaticity for each of the primaries when it has its ReferenceWhite value and the other primaries have their ReferenceBlack values. These values are described using the 1931 CIE xy chromaticity diagram and only the chromaticities are specified. These values can correspond to the chromaticities of the phosphors of a monitor, the filter set and light source combination of a scanner or the imaging model of a rendering package. The ordering is red[x], red[y], green[x], green[y], blue[x], and blue[y].

For example the CCIR Recommendation 709 primaries are:

640/1000,330/1000,

300/1000, 600/1000,

150/1000, 60/1000

No default.

TransferFunction

Tag =301 (12D.H)

Type = SHORT

N = {1 or 3} * (1 << BitsPerSample)

Describes a transfer function for the image in tabular style. Pixel components can be gamma-compensated, companded, non-uniformly quantized, or coded in some other way. The TransferFunction maps the pixel components from a non-linear BitsPerSample (e.g. 8-bit) form into a 16-bit linear form without a perceptible loss of accuracy.

If N = 1 << BitsPerSample, the transfer function is the same for each channel and all channels share a single table. Of course, this assumes that each channel has the same BitsPerSample value.

If N = 3 * (1 << BitsPerSample), there are three tables, and the ordering is the same as it is for pixel components of the PhotometricInterpretation field. These tables are separate and not interleaved. For example, with RGB images all red entries come first, followed by all green entries, followed by all blue entries.

The length of each component table is 1 << BitsPerSample. The width of each entry is 16 bits as implied by the type SHORT. Normally the value 0 represents the minimum intensity and 65535 represents the maximum intensity and the values [0, 0, 0] represent black and [65535,65535, 65535] represent white. If the TransferRange tag is present then it is used to determine the minimum and maximum values, and a scaling normalization.

The TransferFunction can be applied to images with a PhotometricInterpretation value of RGB, Palette, YCbCr, WhiteIsZero, and BlackIsZero. The TransferFunction is not used with other PhotometricInterpretation types.

For RGB PhotometricInterpretation, ReferenceBlackWhite expands the coding range, TransferRange expands the range of the TransferFunction, and the TransferFunction tables decompand the RGB value. The WhitePoint and PrimaryChromaticities further describe the RGB colorimetry.

For Palette color PhotometricInterpretation, the Colormap maps the pixel into three 16-bit values that when scaled to BitsPerSample-bits serve as indices into the TransferFunction tables which decompand the RGB value. The WhitePoint and PrimaryChromaticities further describe the underlying RGB colorimetry.

A Palette value can be scaled into a TransferFunction index by:

index= (value * ((1 << BitsPerSample) - 1)) / 65535;

A TransferFunction index can be scaled into a Palette color value by:

value= (index * 65535L) / ((1 << BitsPerSample) - 1);

Be careful if you intend to create Palette images with a TransferFunction. If the Colormap tag is directly converted from a hardware colormap, it may have a device gamma already incorporated into the DAC values.

For YCbCr PhotometricInterpretation, ReferenceBlackWhite expands the coding range, the YCbCrCoefficients describe the decoding matrix to transform YCbCr into RGB, TransferRange expands the range of the TransferFunction, and the TransferFunction tables decompand the RGB value. The WhitePoint and PrimaryChromaticities fields provide further description of the underlying RGB colorimetry.

After coding range expansion by ReferenceBlackWhite and TransferFunction expansion by TransferRange, RGB values may be outside the domain of the TransferFunction. Also, the display device matrix can transform RGB values into display device RGB values outside the domain of the device. These values are handled in an application-dependent manner.

For RGB images with non-default ReferenceBlackWhite coding range expansion and for YCbCr images, the resolution of the TransferFunction may be insufficient. For example, after the YCbCr transformation matrix, the decoded RGB values must be rounded to index into the TransferFunction tables. Applications needing the extra accuracy should interpolate between the elements of the TransferFunction tables. Linear interpolation is recommended.

For WhiteIsZero and BlackIsZero PhotometricInterpretation, the TransferFunction decompands the grayscale pixel value to a linear 16-bit form. Note that a TransferFunction value of 0 represents black and 65535 represents white regardless of whether a grayscale image is WhiteIsZero or BlackIsZero. For example, the zeroth element of a WhiteIsZero TransferFunction table will likely be 65535. This extension of the TransferFunction field for grayscale images is intended to replace the GrayResponseCurve field.

The TransferFunction does not describe a transfer characteristic outside of the range for ReferenceBlackWhite.

Default is a single table corresponding to the NTSC standard gamma value of 2.2. This table is used for each channel. It can be generated by:

```
NValues = 1 << BitsPerSample;
for (TF[0]= 0, i = 1; i < NValues; i++)
            TF[i]= floor(pow(i / (NValues - 1.0), 2.2) * 65535 + 0.5);
```

TransferRange

Tag = 342 (156.H)

Type = SHORT

N = 6

Expands the range of the TransferFunction. The first value within a pair is associated with TransferBlack and the second is associated with TransferWhite. The ordering of pairs is the same as for pixel components of the PhotometricInterpretation type. By default, theTransferFunction is defined over a range from a minimum intensity, 0 or nominal black, to a maximum intensity,(1 << BitsPerSample) - 1 or nominal white. Kodak PhotoYCC uses an extended range TransferFunction in order to describe highlights, saturated colors and shadow detail beyond this range. The TransferRange expands the TransferFunction to support these values. It is defined only for RGB and YCbCr PhotometricInterpretations.

After ReferenceBlackWhite and/or YCbCr decoding has taken place, an RGB value can be represented as a real number. It is then rounded to create an index into the TransferFunctiontable. In the absence of a TransferRange tag, or if the tag has the default values, the rounded value is an index and the normalized intensity value is:

```
index = (int) (value + (value < 0.0? -0.5 : 0.5));
intensity = TF[index] / 65535;
```

If the TransferRange tag is present and has non-default values, it provides an offset to be used with the rounded index. It also describes a scaling. The normalized intensity value is:

```
index = (int) (value + (value < 0.0? -0.5 : 0.5));
intensity = (TF[index + TransferRange[Black]] -
          TF[TransferRange[Black]])
          / (TF[TransferRange[White]] - TF[TransferRange[Black]]);
```

An application can write a TransferFunction with a non-defaultTransferRange as follows:

```
black_offset = scale_factor * Transfer(-TransferRange[Black]ar /
          (TransferRange[White] - TransferRange[Black]));
for (i = 0; i < (1 << BitsPerSample); i++)
          TF[i] = floor(0.5 - black_offset + scale_factor
                    * Transfer((i - TransferRange[Black])
                    / (TransferRange[White] - TransferRange[Black])));
```

The TIFF writer chooses scale_factor such that the TransferFunction fits into a 16-bit unsigned short, and chooses the TransferRange so that the most important part of the TransferFunction fits into the table.

Default is [0, NV, 0, NV, 0, NV] where NV = (1 <<BitsPerSample) - 1.

ReferenceBlackWhite

Tag =532 (214.H)

Type = RATIONAL

N = 6

Specifies a pair of headroom and footroom image data values (codes) for each pixel component. The first component code within a pair is associated with ReferenceBlack, and the second is associated with ReferenceWhite. The ordering of pairs is the same as those for pixel components of the PhotometricInterpretation type. ReferenceBlackWhite can be applied to images with a PhotometricInterpretation value of RGB or YCbCr. ReferenceBlackWhite is not used with other PhotometricInterpretation values.

Computer graphics commonly places black and white at the extremities of the binary representation of image data; for example, black at code 0 and white at code 255. In other disciplines, such as printing, film, and video, there are practical reasons to provide footroom codes below ReferenceBlack and headroom codes above ReferenceWhite.

In film applications, they correspond to the densities Dmax and Dmin. In video applications, ReferenceBlack corresponds to 7.5 IRE and 0 IRE in systems with and without setup respectively, and ReferenceWhite corresponds to 100 IRE units.

Using YCbCr (See Section 21) and the CCIR Recommendation 601.1 video standard as an example, code 16 represents ReferenceBlack, and code 235 represents ReferenceWhite for the luminance component (Y). For the chrominance components, Cb and Cr, code 128 represents ReferenceBlack, and code 240 represents ReferenceWhite. With Cb and Cr, the ReferenceWhite value is used to code reference blue and reference red respectively.

The full range component value is converted from the code by:

```
FullRangeValue = (code - ReferenceBlack) * CodingRange
        / (ReferenceWhite - ReferenceBlack);
```

The code is converted from the full-range component value by:

```
code = (FullRangeValue * (ReferenceWhite - ReferenceBlack)
        / CodingRange) + ReferenceBlack;
```

For RGB images and the Y component of YCbCr images, CodingRange is defined as:

```
CodingRange = 2 ** BitsPerSample - 1;
```

For the Cb and Cr components of YCbCr images, CodingRange is defined as:

```
CodingRange = 127;
```

For RGB images, in the default special case of no headroom or footroom, this conversion can be skipped because the scaling multiplier equals 1.0 and the value equals the code.

For YCbCr images, in the case of no headroom or footroom, the conversion for Y can be skipped because the value equals the code. For Cb and Cr, ReferenceBlack must still be subtracted from the code. In the general case, the scaling multiplication for the Cb and Cr component codes can be factored into the YCbCr transform matrix.

Useful ReferenceBlackWhite values for YCbCr images are:

[0/1, 255/1,128/1, 255/1, 128/1, 255/1]

no headroom/footroom

[15/1, 235/1, 128/1, 240/1, 128/1, 240/1]

CCIR Recommendation 601.1 headroom/footroom

Useful ReferenceBlackWhite values for BitsPerSample = 8,8,8 Class R images are:

[0/1, 255/1,0/1, 255/1, 0/1, 255/1]

no headroom/footroom

[16/1, 235/1, 16/1, 235/1, 16/1, 235/1]

CCIR Recommendation 601.1 headroom/footroom

Default is [0/,NV/1, 0/1, NV/1, 0/1, NV/1] where NV = 2 ** BitsPerSample - 1.

References

[1] *The Reproduction of Colour in Photography, Printing and Television,* R. W. G. Hunt, Fountain Press, Tolworth, England,1987.

[2] *Principles of Color Technology*, Billmeyer and Saltzman, Wiley-Interscience, New York, 1981.

[3] *Colorimetric Properties of Video Displays*, William Cowan, University of Waterloo, Waterloo, Canada, 1989.

[4] *TIFF Color Appearance Guidelines*, Dave Farber, Eastman Kodak Company, Rochester, New York.

Section 21: YC_bC_r Images

Introduction

CDB-compliant TIFF readers do not consider YC_bC_r color encoded TIFF image data. As a result, section 21 is not applicable to CDB-compliant TIFF

Digitizers of video sources that create RGB data are becoming more capable and less expensive. The RGB color space is adequate for this purpose. However, for both digital video and image compression applications a color difference color space is needed. The television industry depends on YC_bC_r for digital video. For image compression, subsampling the chrominance components allows for greater compression. TIFF YC_bC_r (which we shall call *Class Y*) supports these images and applications.

Class Y is based on CCIR Recommendation 601-1, "Encoding Parameters of Digital Television for Studios." Class Y also has parameters that allow the description of related standards such as CCIR Recommendation 709 and technological variations such as component-sample positioning.

YC_bC_r is a distinct PhotometricInterpretation type. RGB pixels are converted to and from YC_bC_r for storage and display.

Class Y defines the following fields:

YC_bC_rCoefficients	transformation from RGB to YC_bC_r
YC_bC_rSubSampling	subsampling of the chrominance components
YC_bC_rPositioning	positioning of chrominance component samples relative to the luminance samples

In addition, ReferenceBlackWhite, which specifies coding range expansion, is required by Class Y. See Section 20.

Class Y YC_bC_r images have three components: Y, the luminance component, and C_b and C_r, two chrominance components. Class Y uses the international standard notation YC_bC_r for color-difference component coding. This is often incorrectly called YUV, which properly applies only to composite coding.

The transformations between YC_bC_r and RGB are linear transformations of uninterpreted RGB sample data, typically gamma-corrected values. The YC_bC_rCoefficients field describes the parameters of this transformation.

Another feature of Class Y comes from subsampling the chrominance components. A Class Y image can be compressed by reducing the spatial resolution of chrominance components. This takes advantage of the relative insensitivity of the human visual system to chrominance detail. The YC_bC_rSubSampling field describes the degree of subsampling which has taken place.

When a Class Y image is subsampled, each C_b and C_r sample is associated with a group of luminance samples. The YC_bC_rPositioning field describes the position of the chrominance component samples relative to the group of luminance samples: centered or cosited.

Class Y requires use of the ReferenceBlackWhite field. This field expands the coding range by describing the reference black and white values for the different components that allow headroom and footroom for digital video images. Since the

default for ReferenceBlackWhite is inappropriate for Class Y, it must be used explicitly.

At first, it might seem that the information conveyed by Class Y and the RGB Colorimetry section is redundant. However, decoding YC_bC_r to RGB primaries requires the YC_bC_r fields, and interpretation of the resulting RGB primaries requires the colorimetry and transfer function information. See the RGB Colorimetry section for details.

Extensions to Existing Fields

Class Y images use a distinct PhotometricInterpretation Field value:

PhotometricInterpretation

Tag = 262 (106.H)

Type = SHORT

N = 1

This Field indicates the color space of the image. The new value is:

6 = YC_bC_r

A value of 6 indicates that the image data is in the YC_bC_r color space. TIFF uses the international standard notation YC_bC_r for color-difference sample coding. Y is the luminance component. C_b and C_r are the two chrominance components. RGB pixels are converted to and from YC_bC_r form for storage and display.

Fields Defined in Class Y

YC_bC_r Coefficients

Tag = 529 (211.H)

Type = RATIONAL

N = 3

The transformation from RGB to YC_bC_r image data. The transformation is specified as three rational values that represent the coefficients used to compute luminance, Y.

The three rational coefficient values, *LumaRed*, *LumaGreen* and *LumaBlue*, are the proportions of red, green, and blue respectively in luminance, Y.

Y, C_b, and C_r may be computed from RGB using the luminance coefficients specified by this field as follows:

$Y = (LumaRed * R + LumaGreen * G + LumaBlue * B)$

$C_b = (B - Y) / (2 - 2 * LumaBlue)$

$C_r = (R - Y) / (2 - 2 * LumaRed)$

R, G, and B may be computed from YC_bC_r as follows:

$R = C_r * (2 - 2 * LumaRed) + Y$

$G = (Y - LumaBlue * B - LumaRed * R) / LumaGreen$

$B = C_b * (2 - 2 * LumaBlue) + Y$

In disciplines such as printing, film, and video, there are practical reasons to provide footroom codes below the ReferenceBlack code and headroom codes above ReferenceWhite code. In such cases the values of the transformation matrix used to convert from YC_bC_r to RGB must be multiplied by a scale factor to produce full-range RGB values. These scale factors depend on the reference ranges specified by the ReferenceBlackWhite field. See the ReferenceBlackWhite and TransferFunction fields for more details.

The values coded by this field will typically reflect the transformation specified by a standard for YC_bC_r encoding. The following table contains examples of commonly used values.

Standard	*LumaRed*	*LumaGreen*	*LumaBlue*
CCIR Recommendation 601-1	299 / 1000	587 / 1000	114 / 1000
CCIR Recommendation 709	2125 / 10000	7154 / 10000	721 / 10000

The default values for this field are those defined by CCIR Recommendation 601-1: 299/1000, 587/1000 and 114/1000, for *LumaRed*, *LumaGreen* and *LumaBlue*, respectively.

YC_bC_rSubSampling

Tag = 530 (212.H)

Type = SHORT

N = 2

Specifies the subsampling factors used for the chrominance components of a YC_bC_r image. The two fields of this field, *YC_bC_rSubsampleHoriz* and *YC_bC_rSubsampleVert*, specify the horizontal and vertical subsampling factors respectively.

The two fields of this field are defined as follows:

Short 0: *YC_bC_rSubsampleHoriz*:

1 = ImageWidth of this chroma image is equal to the ImageWidth of the associated luma image.

2 = ImageWidth of this chroma image is halfthe ImageWidth of the associated luma image.

4 = ImageWidth of this chroma image is one-quarter the ImageWidth of the associated luma image.

Short 1: *YC_bC_rSubsampleVert*:

1 = ImageLength (height) of this chroma image is equal to the ImageLength of the associated luma image.

2 = ImageLength (height) of this chroma image is half the ImageLength of the associated luma image.

4 = ImageLength (height) of this chroma image is one-quarter the ImageLength of the associated luma image.

Both C_b and C_r have the same subsampling ratio. Also, *YC_bC_rSubsampleVert* shall always be less than or equal to *YC_bC_rSubsampleHoriz*.

ImageWidth and ImageLength are constrained to be integer multiples of *YC_bC_rSubsampleHoriz* and *YC_bC_rSubsampleVert* respectively. TileWidth and TileLength have the same constraints. RowsPerStrip must be an integer multiple of *YC_bC_rSubsampleVert*.

The default values of this field are [2, 2].

YC_bC_rPositioning

Tag = 531 (213.H)

Type = SHORT

N = 1

Specifies the positioning of subsampled chrominance components relative to luminance samples.

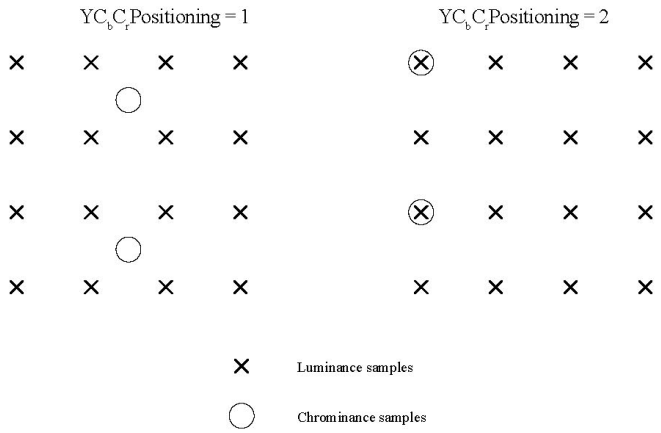
Specification of the spatial positioning of pixel samples relative to the other samples is necessary for proper image post processing and accurate image presentation. In Class Y files, the position of the subsampled chrominance components are defined with respect to the luminance component. Because components must be sampled orthogonally (along rows and columns), the spatial position of the samples in a given subsampled component may be determined by specifying the horizontal and vertical offsets of the first sample (i.e. the sample in the upper-left corner) with respect to the luminance component. The horizontal and vertical offsets of the first chrominance sample are denoted Xoffset[0,0] and Yoffset[0,0] respectively. Xoffset[0,0] and Yoffset[0,0] are defined in terms of the number of samples in the luminance component.

The values for this field are defined as follows:

Tag value	YC_bC_r Positioning	X and Y offsets of first chrominance sample
1	centered	Xoffset[0,0] = *ChromaSubsampleHoriz* / 2 - 0.5 Yoffset[0,0] = *ChromaSubsampleVert* / 2 - 0.5
2	cosited	Xoffset[0,0] = 0 Yoffset[0,0] = 0

Field value 1 (centered) must be specified for compatibility with industry standards such as PostScript Level 2 and QuickTime. Field value 2 (cosited) must be specified for compatibility with most digital video standards, such as CCIR Recommendation 601-1.

As an example, for *ChromaSubsampleHoriz* = 4 and *ChromaSubsampleVert* = 2, the centers of the samples are positioned as illustrated below:



Proper subsampling of the chrominance components incorporates an anti-aliasing filter that reduces the spectral bandwidth of the full-resolution samples. The type of filter used for subsampling determines the value of the YC_bC_rPositioning field.

For YC_bC_rPositioning = 1 (centered), subsampling of the chrominance components can easily be accomplished using a symmetrical digital filter with an even number of taps (coefficients). A commonly used filter for 2:1 subsampling utilizes two taps (1/2,1/2).

For YC_bC_rPositioning = 2 (cosited), subsampling of the chrominance components can easily be accomplished using a symmetrical digital filter with an odd number of taps. A commonly used filter for 2:1 subsampling utilizes three taps (1/4,1/2,1/4).

The default value of this field is 1.

Ordering of Component Samples

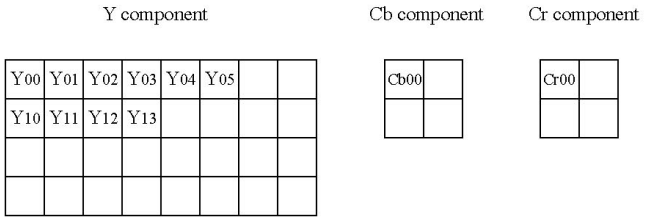
This section defines the ordering convention used for Y, C_b, and C_r component samples when the PlanarConfiguration field value = 1 (interleaving). For PlanarConfiguration = 2, component samples are stored as 3 separate planes, and the ordering is the same as that used for other PhotometricInterpretation field values.

For PlanarConfiguration = 1, the component sample order is based on the subsampling factors, *ChromaSubsampleHoriz* and *ChromaSubsampleVert,* defined by the YC_bC_rSubSampling field. The image data within a TIFF file is comprised of one or more "data units", where a data unit is defined to be a sequence of samples:

- one or more Y samples
- a C_b sample
- a C_r sample

The Y samples within a data unit are specified as a two-dimensional array having *ChromaSubsampleVert* rows of *ChromaSubsampleHoriz* samples.

Expanding on the example in the previous section, consider a YC_bC_r image having *ChromaSubsampleHoriz* = 4 and *ChromaSubsampleVert* = 2:



For PlanarConfiguration = 1, the sample order is:

$$Y_{00}, Y_{01}, Y_{02}, Y_{03}, Y_{10}, Y_{11}, Y_{12}, Y_{13}, Cb_{00}, Cr_{00}, Y_{04}, Y_{05} \ldots$$

Minimum Requirements for YCbCr Images

In addition to satisfying the general Baseline TIFF requirements, a YCbCr file must have the following characteristics:

- SamplesPerPixel = 3. SHORT. Three components representing Y, Cb and Cr.
- BitsPerSample = 8,8,8. SHORT.
- Compression = none (1), LZW (5) or JPEG (6). SHORT.
- PhotometricInterpretation = YC_bC_r (6). SHORT.
- ReferenceBlackWhite = 6 RATIONALS. Specify the reference values for black and white.

If the conversion from RGB is not according to CCIR Recommendation 601-1, code YC_bC_r Coefficients.

CDB-compliant TIFF readers do not consider JPEG color encoded TIFF image data. As a result, section 22 is not applicable to CDB-compliant TIFF readers.

Section 22: JPEG Compression

Introduction

Image compression reduces the storage requirements of pictorial data. In addition, it reduces the time required for access to, communication with, and display of images. To address the standardization of compression techniques an international standards group was formed: the Joint Photographic Experts Group (JPEG). JPEG has as its objective to create a joint ISO/CCITT standard for continuous tone image compression (color and grayscale).

JPEG decided that because of the broad scope of the standard, no one algorithmic procedure was able to satisfy the requirements of all applications. It was decided to specify different algorithmic processes, where each process is targeted to satisfy the requirements of a class of applications. Thus, the JPEG standard became a "toolkit" whereby the particular algorithmic "tools" are selected according to the needs of the application environment.

The algorithmic processes fall into two classes: lossy and lossless. Those based on the Discrete Cosine Transform (DCT) are lossy and typically provide for substantial compression without significant degradation of the reconstructed image with respect to the source image.

The simplest DCT-based coding process is the baseline process. It provides a capability that is sufficient for most applications. There are additional DCT-based processes that extend the baseline process to a broader range of applications.

The second class of coding processes is targeted for those applications requiring lossless compression. The lossless processes are not DCT-based and are utilized independently of any of the DCT-based processes.

This Section describes the JPEG baseline, the JPEG lossless processes, and the extensions to TIFF defined to support JPEG compression.

JPEG Baseline Process

The baseline process is a DCT-based algorithm that compresses images having 8 bits per component. The baseline process operates only in sequential mode. In sequential mode, the image is processed from left to right and top to bottom in a single pass by compressing the first row of data, followed by the second row, and continuing until the end of image is reached. Sequential operation has minimal buffering requirements and thus permits inexpensive implementations.

The JPEG baseline process is an algorithm which inherently introduces error into the reconstructed image and cannot be utilized for lossless compression. The algorithm accepts as input only those images having 8 bits per component. Images with fewer than 8 bits per component may be compressed using the baseline process algorithm by left justifying each input component within a byte before compression.

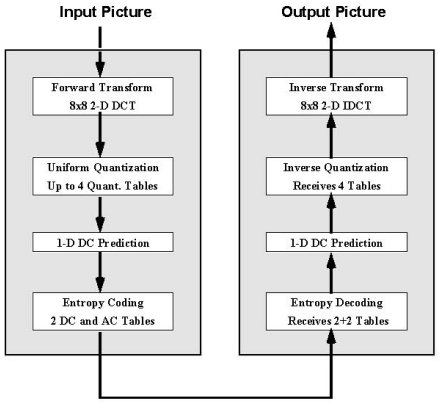


Figure 1. Baseline Process Encoder and Decoder

A functional block diagram of the Baseline encoding and decoding processes is contained in Figure 1. Encoder operation consists of dividing each component of the input image into 8x8 blocks, performing the two-dimensional DCT on each block, quantizing each DCT coefficient uniformly, subtracting the quantized DC coefficient from the corresponding term in the previous block, and then entropy coding the quantized coefficients using variable length codes (VLCs). Decoding is performed by inverting each of the encoder operations in the reverse order.

The DCT

Before performing the foward DCT, input pixels are level-shifted so that they range from -128 to +127. Blocks of 8x8 pixels are transformed with the two-dimensional 8x8 DCT:

$$F(u,v) = \frac{1}{4} C(u)C(v) \sum\sum f(x,y) \cos\frac{\pi(2x+1)u}{16} \cos\frac{\pi(2y+1)v}{16}$$

and blocks are inverse transformed by the decoder with the Inverse DCT:

$$f(x,y) = \frac{1}{4} \sum\sum C(u)C(v) F(u,v) \cos\frac{\pi(2x+1)u}{16} \cos\frac{\pi(2y+1)v}{16}$$

with $u, v, x, y = 0, 1, 2, \ldots 7$

where x, y = spatial coordinates in the pel domain
u, v = coordinates in the transform domain

$$C(u), C(v) = \begin{cases} 1/\text{sqrt}(2) & \text{for } u, v = 0 \\ 1 & \text{otherwise} \end{cases}$$

Although the exact method for computation of the DCT and IDCT is not subject to standardization and will not be specified by JPEG, it is probable that JPEG will adopt DCT-conformance specifications that designate the accuracy to which the DCT must be computed. The DCT-conformance specifications will assure that any two JPEG implementations will produce visually-similar reconstructed images.

Quantization

The coefficients of the DCT are quantized to reduce their magnitude and increase the number of zero-value coefficients. The DCT coefficients are independently quantized by uniform quantizers. A uniform quantizer divides the real number line into steps of equal size, as shown in Figure 2. The quantization step-size applied to each coefficient is determined from the contents of a 64-element quantization table.

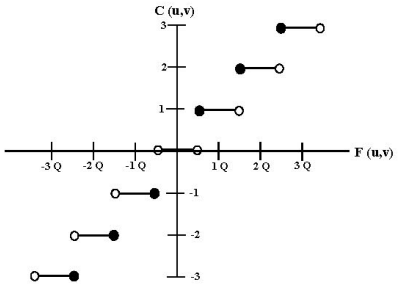


Figure 2. Uniform Quantization

The baseline process provides for up to 4 different quantization tables to be defined and assigned to separate interleaved components within a single scan of the input image. Although the values of each quantization table should ideally be determined through rigorous subjective testing which estimates the human psycho-visual thresholds for each DCT coefficient and for each color component of the input image, JPEG has developed quantization tables which work well for CCIR 601 resolution images and has published these in the informational section of the proposed standard.

DC Prediction

The DCT coefficient located in the upper-left corner of the transformed block represents the average spatial intensity of the block and is referred to as the "DC coefficient". After the DCT coefficients are quantized, but before they are entropy coded, DC prediction is performed. DC prediction simply means that the DC term of the previous block is subtracted from the DC term of the current block prior to encoding.

Zig-Zag Scan

Prior to entropy coding, the DCT coefficients are ordered into a one-dimensional sequence according to a "zig-zag" scan. The DC coefficient is coded first, followed by AC coefficient coding, proceeding in the order illustrated in Figure 3.

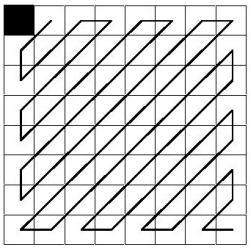


Figure 3. Zig-Zag Scan of DCT Coefficients

Entropy Coding

The quantized DCT coefficients are further compressed using entropy coding. The baseline process performs entropy coding using variable length codes (VLCs) and variable length integers (VLIs).

VLCs, commonly known as Huffman codes, compress data symbols by creating shorter codes to represent frequently-occurring symbols and longer codes for occasionally-occurring symbols. One reason for using VLCs is that they are easily implemented by means of lookup tables.

Separate code tables are provided for the coding of DC and AC coefficients. The following paragraphs describe the respective coding methods used for coding DC and AC coefficients.

DC Coefficient Coding

DC prediction produces a "differential DC coefficient" that is typically small in magnitude due to the high correlation of neighboring DC coefficients. Each differential DC coefficient is encoded by a VLC which represents the number of significant bits in the DC term followed by a VLI representing the value itself. The VLC is coded by first determining the number of significant bits, SSSS, in the differential DC coefficient through the following table:

SSSS	Differential DC Value
0	0
1	-1, 1
2	-3,-2, 2,3
3	-7..-4, 4..7
4	-15..-8, 8..15
5	-31..-16, 16..31

6	-63..-32, 32..63
7	-127..-64, 64..127
8	-255..-128, 128..255
9	-511..-256, 256..511
10	-1023..-512, 512..1023
11	-2047..-1024, 1024..2047
12	-4095..-2048, 2048..4095

SSSS is then coded from the selected DC VLC table. The VLC is followed by a VLI having SSSS bits that represents the value of the differential DC coefficient itself. If the coefficient is positive, the VLI is simply the low-order bits of the coefficient. If the coefficient is negative, then the VLI is the low-order bits of the coefficient-1.

AC Coefficient Coding

In a similar fashion, AC coefficients are coded with alternating VLC and VLI codes. The VLC table, however, is a two-dimensional table that is indexed by a composite 8-bit value. The lower 4 bits of the 8-bit value, i.e. the column index, is the number of significant bits, SSSS, of a non-zero AC coefficient. SSSS is computed through the same table as that used for coding the DC coefficient. The higher-order 4 bits, the row index, is the number of zero coefficients, NNNN, that precede the non-zero AC coefficient. The first column of the two-dimensional coding table contains codes that represent control functions. Figure 4 illustrates the general structure of the AC coding table.

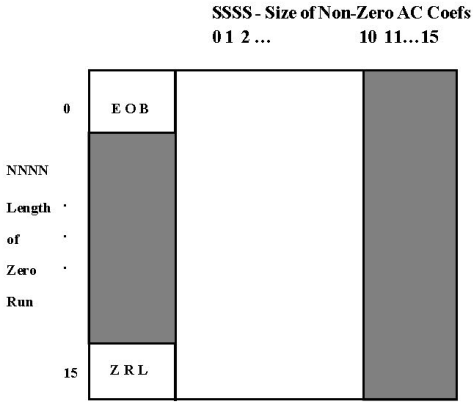


Figure 4. 2-D Run-Size Value Array for AC Coefs
The shaded portions are undefined in the baseline process

The flow chart in Figure 5 specifies the AC coefficient coding procedure. AC coefficients are coded by traversing the block in the zig-zag sequence and count-

ing the number of zero coefficients until a non-zero AC coefficient is encountered. If the count of consecutive zero coefficients exceeds 15, then a ZRL code is coded and the zero run-length count is reset. When a non-zero AC coefficient is found, the number of significant bits in the non-zero coefficient, SSSS, is combined with the zero run-length that precedes the coefficient, NNNN, to form an index into the two-dimensional VLC table. The selected VLC is then coded. The VLC is followed by a VLI that represents the value of the AC coefficient. This process is repeated until the end of the block is reached. If the last AC coefficient is zero, then an End of Block (EOB) VLC is encoded.

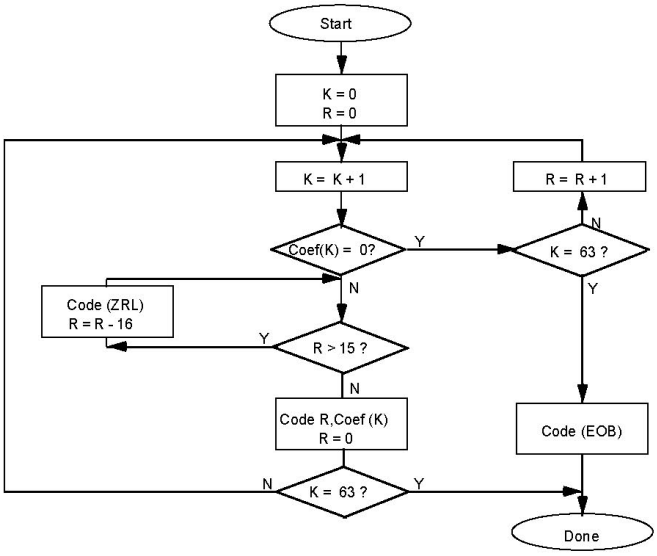


Figure 5. Encoding Procedure for AC Coefs

JPEG Lossless Processes

The JPEG lossless coding processes utilize a spatial-prediction algorithm based upon a two-dimensional Differential Pulse Code Modulation (DPCM) technique. They are compatible with a wider range of input pixel precision than the DCT-based algorithms (2 to 16 bits per component). Although the primary motivation for specifying a spatial algorithm is to provide a method for lossless compression, JPEG allows for quantization of the input data, resulting in lossy compression and higher compression rates.

Although JPEG provides for use of either the Huffman or Arithmetic entropy-coding models by the processes for lossless coding, only the Huffman coding model is supported by this version of TIFF. The following is a brief overview of the lossless process with Huffman coding.

Control Structure

Much of the control structure developed for the sequential DCT procedures is also used for sequential lossless coding. Either interleaved or non-interleaved data ordering may be used.

Coding Model

The coding model developed for coding the DC coefficients of the DCT is extended to allow a number of one-dimensional and two-dimensional predictors for the lossless coding function. Each component uses an independent predictor.

Prediction

Figure 6 shows the relationship between the neighboring values used for prediction and the sample being coded.

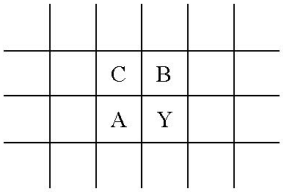


Figure 6. Relationship between sample and prediction samples

Y is the sample to be coded and A, B, and C are the samples immediately to the left, immediately above, and diagonally to the left and above.

The allowed predictors are listed in the following table.

Selection-value	Prediction
0	no prediction (differential coding)
1	A
2	B
3	C
4	A+B-C
5	A+((B-C)/2)
6	B+((A-C)/2)
7	(A+B)/2

Selection-value 0 shall only be used for differential coding in the hierarchical mode. Selections 1, 2 and 3 are one-dimensional predictors and selections 4, 5, 6, and 7 are two dimensional predictors. The divide by 2 in the prediction equations is done by a arithmetic-right-shift of the integer values.

The difference between the prediction value and the input is calculated modulo 2**16. Therefore, the prediction can also be treated as a modulo 2**16 value. In the decoder the difference is decoded and added, modulo 2**16, to the prediction.

Huffman Coding of the Prediction Error

The Huffman coding procedures defined for coding the DC coefficients are used to code the modulo 2**16 differences. The table for DC coding is extended to 17 entries that allows for coding of the modulo 2**16 differences.

Point Transformation Prior to Lossless Coding

For the lossless processes only, the input image data may optionally be scaled (quantized) prior to coding by specifying a nonzero value in the point transformation parameter. Point transformation is defined to be division by a power of 2.

If the point transformation field is nonzero for a component, a point transformation of the input is performed prior to the lossless coding. The input is divided by 2**Pt, where Pt is the value of the point transform signaling field. The output of the decoder is rescaled to the input range by multiplying by 2**Pt. Note that the scaling of input and output can be performed by arithmetic shifts.

Overview of the JPEG Extension to TIFF

In extending the TIFF definition to include JPEG compressed data, it is necessary to note the following:

- JPEG is effective only on continuous-tone color spaces:

Grayscale	(Photometric Interpretation = 1)	
RGB	(Photometric Interpretation = 2)	
CMYK	(Photometric Interpretation = 5)	(See the CMYK Images section.)
YC_bC_r	(Photometric Interpretation = 6)	(See the YCbCr images section.)

- Color conversion to YC_bC_r is often used as part of the compression process because the chrominance components can be subsampled and compressed to a greater degree without significant visual loss of quality. Fields are defined to describe how this conversion has taken place and the degree of subsampling employed (see the YCbCr Images section).

- New fields have been defined to specify the JPEG parameters used for compression and to allow quantization tables and Huffman code tables to be incorporated into the TIFF file.

- TIFF is compatible with compressed image data that conforms to the syntax of the JPEG interchange format for compressed image data. Fields are defined that may be utilized to facilitate conversion from TIFF to interchange format.

- The PlanarConfiguration Field is used to specify whether or not the compressed data is interleaved as defined by JPEG. For any of the JPEG DCT-based processes, the interleaved data units are coded 8x8 blocks rather than component samples.

- Although JPEG codes consecutive image blocks in a single contiguous bitstream, it is extremely useful to employ the concept of tiles in an image. The TIFF Tiles section defines some new fields for tiles. These fields should be stored in place of the older fields for strips. The concept of tiling an image in both dimensions is important because JPEG hardware may be limited in the size of each block that is handled.

- Note that the nomenclature used in the TIFF specification is different from the JPEG Draft International Standarditee Draft (ISO DIS 10918-1) in some respects. The following terms should be equated when reading this Section:

TIFF name	JPEG DIS name
ImageWidth	Number of Pixels
ImageLength	Number of Lines
SamplesPerPixel	Number of Components
JPEGQTable	Quantization Table
JPEGDCTable	Huffman Table for DC coefficients
JPEGACTable	Huffman Table for AC coefficients

Strips and Tiles

The JPEG extension to TIFF has been designed to be consistent with the existing TIFF strip and tile structures and to allow quick conversion to and from the stream-oriented compressed image format defined by JPEG.

Compressed images conforming to the syntax of the JPEG interchange format can be converted to TIFF simply by defining a single strip or tile for the entire image and then concatenating the TIFF image description fields to the JPEG compressed image data. The strip or tile offset field points directly to the start of the entropy coded data (not to a JPEG marker).

Multiple strips or tiles are supported in JPEG compressed images using restart markers. Restart markers, inserted periodically into the compressed image data, delineate image segments known as restart intervals. At the start of each restart interval, the coding state is reset to default values, allowing every restart interval to be decoded independently of previously decoded data. TIFF strip and tile offsets shall always point to the start of a restart interval. Equivalently, each strip or

tile contains an integral number of restart intervals. Restart markers need not be present in a TIFF file; they are implicitly coded at the start of every strip or tile.

To maximize interchangeability of TIFF files with other formats, a restriction is placed on tile height for files containing JPEG-compressed image data conforming to the JPEG interchange format syntax. The restriction, imposed only when the tile width is shorter than the image width and when the JPEGInterchangeFormat Field is present and non-zero, states that the tile height must be equal to the height of one JPEG Minimum Coded Unit (MCU). This restriction ensures that TIFF files may be converted to JPEG interchange format without undergoing decompression.

Extensions to Existing Fields

Compression

Tag = 259 (103.H)

Type = SHORT

N = 1

This Field indicates the type of compression used. The new value is:

6 = JPEG

JPEG Fields

JPEGProc

Tag = 512 (200.H)

Type = SHORT

N = 1

This Field indicates the JPEG process used to produce the compressed data. The values for this field are defined to be consistent with the numbering convention used in ISO DIS 10918-2. Two values are defined at this time.

1= Baseline sequential process

14= Lossless process with Huffman coding

When the lossless process with Huffman coding is selected by this Field, the Huffman tables used to encode the image are specified by the JPEGDCTables field, and the JPEGACTables field is not used.

Values indicating JPEG processes other than those specified above will be defined in the future.

Not all of the fields described in this section are relevant to the JPEG process selected by this Field. The following table specifies the fields that are applicable to each value defined by this Field.

Tag Name	JPEGProc =1	JPEGProc =14
JPEGInterchangeFormat	X	X
JPEGInterchangeFormatLength	X	X
JPEGRestart Interval	X	X
JPEGLosslessPredictors		X
JPEGPointTransforms		X
JPEGQTables	X	
JPEGDCTables	X	X
JPEGACTables	X	

This Field is mandatory whenever the Compression Field is JPEG (no default).

JPEGInterchangeFormat

Tag = 513 (201.H)

Type = LONG

N = 1

This Field indicates whether a JPEG interchange format bitstream is present in the TIFF file. If a JPEG interchange format bitstream is present, then this Field points to the Start of Image (SOI) marker code.

If this Field is zero or not present, a JPEG interchange format bitstream is not present.

JPEGInterchangeFormatLength

Tag = 514 (202.H)

Type = LONG

N = 1

This Field indicates the length in bytes of the JPEG interchange format bitstream. This Field is useful for extracting the JPEG interchange format bitstream without parsing the bitstream.

This Field is relevant only if the JPEGInterchangeFormat Field is present and is non-zero.

JPEGRestartInterval

Tag = 515 (203.H)

Type = SHORT

N = 1

This Field indicates the length of the restart interval used in the compressed image data. The restart interval is defined as the number of Minimum Coded Units (MCUs) between restart markers.

Restart intervals are used in JPEG compressed images to provide support for multiple strips or tiles. At the start of each restart interval, the coding state is reset to default values, allowing every restart interval to be decoded independently of previously decoded data. TIFF strip and tile offsets shall always point to the start of a restart interval. Equivalently, each strip or tile contains an integral number of restart intervals. Restart markers need not be present in a TIFF file; they are implicitly coded at the start of every strip or tile.

See the JPEG Draft International Standard (ISO DIS 10918-1) for more information about the restart interval and restart markers.

If this Field is zero or is not present, the compressed data does not contain restart markers.

JPEGLosslessPredictors

Tag = 517 (205.H)

Type = SHORT

N = SamplesPerPixel

This Field points to a list of lossless predictor-selection values, one per component.
The allowed predictors are listed in the following table.

Selection-value	Prediction
1	A
2	B
3	C
4	A+B-C
5	A+((B-C)/2)
6	B+((A-C)/2)
7	(A+B)/2

A, B, and C are the samples immediately to the left, immediately above, and diagonally to the left and above the sample to be coded, respectively.

See the JPEG Draft International Standard (ISO DIS 10918-1) for more details.

This Field is mandatory whenever the JPEGProc Field specifies one of the lossless processes (no default).

JPEGPointTransforms

Tag = 518 (206.H)

Type = SHORT

N = SamplesPerPixel

This Field points to a list of point transform values, one per component. This Field is relevant only for lossless processes.

If the point transformation value is nonzero for a component, a point transformation of the input is performed prior to the lossless coding. The input is divided by 2**Pt, where Pt is the point transform value. The output of the decoder is rescaled to the input range by multiplying by 2**Pt. Note that the scaling of input and output can be performed by arithmetic shifts.

See the JPEG Draft International Standard (ISO DIS 10918-1) for more details. The default value of this Field is 0 for each component (no scaling).

JPEGQTables

Tag = 519 (207.H)

Type = LONG

N = SamplesPerPixel

This Field points to a list of offsets to the quantization tables, one per component. Each table consists of 64 BYTES (one for each DCT coefficient in the 8x8 block). The quantization tables are stored in zigzag order.

See the JPEG Draft International Standard (ISO DIS 10918-1) for more details.

It is strongly recommended that, within the TIFF file, each component be assigned separate tables. This Field is mandatory whenever the JPEGProc Field specifies a DCT-based process (no default).

JPEGDCTables

Tag = 520 (208.H)

Type = LONG

N = SamplesPerPixel

This Field points to a list of offsets to the DC Huffman tables or the lossless Huffman tables, one per component.

The format of each table is as follows:

16 BYTES of "BITS", indicating the number of codes of lengths 1 to 16;

Up to 17 BYTES of "VALUES", indicating the values associated with those codes, in order of length.

See the JPEG Draft International Standard (ISO DIS 10918-1) for more details.

It is strongly recommended that, within the TIFF file, each component be assigned separate tables. This Field is mandatory for all JPEG processes (no default).

JPEGACTables

Tag = 521 (209.H)

Type = LONG

N = SamplesPerPixel

This Field points to a list of offsets to the Huffman AC tables, one per component. The format of each table is as follows:

16 BYTES of "BITS", indicating the number of codes of lengths 1 to 16;

Up to 256 BYTES of "VALUES", indicating the values associated with those codes, in order of length.

See the JPEG Draft International Standard (ISO DIS 10918-1) for more details.

It is strongly recommended that, within the TIFF file, each component be assigned separate tables. This Field is mandatory whenever the JPEGProc Field specifies a DCT-based process (no default).

Minimum Requirements for TIFF with JPEG Compression

The table on the following page shows the minimum requirements of a TIFF file that uses tiling and contains JPEG data compressed with the Baseline process.

Tag = NewSubFileType (254) Type = Long Length = 1 Value = 0	Single image
Tag = ImageWidth (256) Type = Long Length = 1 Value = ?	
Tag = ImageLength (257) Type = Long Length = 1 Value = ?	
Tag = BitsPerSample (258) Type = Short Length = SamplesPerPixel Value = ?	8 : Monochrome 8,8,8 : RGB 8,8,8 : YCbCr 8,8,8,8 : CMYK
Tag = Compression (259) Type = Long Length = 1 Value = 6	6 : JPEG compression
Tag = PhotometricInterpretation (262) Type = Short Length = 1 Value = ?	0,1 : Monochrome 2 : RGB 5 : CMYK 6 : YCbCr
Tag = SamplesPerPixel (277) Type = Short Length = 1 Value = ?	1 : Monochrome 3 : RGB 3 : YCbCr 4 : CMYK
Tag = XResolution (282) Type = Rational Length = 1 Value = ?	
Tag = YResolution (283) Type = Rational Length = 1 Value = ?	
Tag = PlanarConfiguration (284) Type = Short Length = 1 Value = ?	1 : Block Interleaved 2 : Not interleaved
Tag = ResolutionUnit (296) Type = Short Length = 1 Value = ?	
Tag = TileWidth (322) Type = Short Length = 1 Value = ?	Multiple of 8
Tag = TileLength (323) Type = Short Length = 1 Value = ?	Multiple of 8
Tag = TileOffsets (324) Type = Long Length = Number of tiles Value = ?	
Tag = TileByteCounts (325) Type = Long Length = Number of tiles Value = ?	
Tag = JPEGProc (512) Type = Short Length = 1 Value = ?	1 : Baseline process
Tag = JPEGQTables (519) Type = Long Length = SamplesPerPixel Value = ?	Offsets to tables
Tag = JPEGDCTables (520) Type = Long Length = SamplesPerPixel Value = ?	Offsets to tables
Tag = JPEGACTables (521) Type = Long Length = SamplesPerPixel Value = ?	Offsets to tables

References

[1] Wallace, G., "Overview of the JPEG Still Picture Compression Algorithm", Electronic Imaging East '90.

[2] ISO/IEC DIS 10918-1, "Digital Compression and Coding of Continuous-tone Still Images", Sept. 1991.

Section 23: CIE L*a*b* Images

What is CIE L*a*b*?

CDB-compliant TIFF readers do not consider CIE L*a*b* color encoded TIFF image data. As a result, section 23 is not applicable to CDB-compliant TIFF

CIE La*b* is a color space that is colorimetric, has separate lightness and chroma channels, and is approximately perceptually uniform. It has excellent applicability for device-independent manipulation of continuous tone images. These attributes make it an excellent choice for many image editing functions.

1976 CIEL*a*b* is represented as a Euclidean space with the following three quantities plotted along axes at right angles: L^* representing lightness, a^* representing the red/green axis, and b^* representing the yellow/blue axis. The formulas for 1976 CIE $L^*a^*b^*$ follow:

$$L^* = 116(Y/Y_n)^{1/3} - 16 \qquad \text{for } Y/Y_n > 0.008856$$

$$L^* = 903.3(Y/Y_n) \qquad \text{for } Y/Y_n <= 0.008856 \qquad \text{*see note below.}$$

$$a^* = 500\left[(X/X_n)^{1/3} - (Y/Y_n)^{1/3}\right]$$

$$b^* = 200\left[(Y/Y_n)^{1/3} - (Z/Z_n)^{1/3}\right].$$

where X_n, Y_n, and Z_n are the CIE X, Y, and Z tristimulus values of an *appropriate* reference white. Also, if any of the ratios X/X_n, Y/Y_n, or Z/Z_n is equal to or less than 0.008856, it is replaced in the formulas with

$$7.787F + 16/116,$$

where F is X/X_n, Y/Y_n, or Z/Z_n, as appropriate (note: these low-light conditions are of no relevance for most document-imaging applications). Tiff is defined such that each quantity be encoded with 8 bits. This provides 256 levels of L^* lightness; 256 levels (+/- 127) of a^*, and 256 levels (+/- 127) of b^*. Dividing the 0-100 range of L^* into 256 levels provides lightness steps that are less than half the size of a "just noticeable difference". This eliminates banding, even under conditions of substantial tonal manipulation. Limiting the theoretically unbounded a^* and b^* ranges to +/- 127 allows encoding in 8 bits without eliminating any but the most saturated self-luminous colors. It is anticipated that the rare specialized applications requiring support of these extreme cases would be unlikely to use CIELAB anyway. All object colors, in fact all colors within the theoretical MacAdam limits, fall within the +/- 127 a^*/b^* range.

The TIFF CIELAB Fields

PhotometricInterpretation

Tag = 262 (106.H)

Type = SHORT

N = 1

8 = 1976 CIE *L*a*b**

Usage of other Fields.

BitsPerSample: 8

SamplesPerPixel - ExtraSamples: 3 for *L*a*b**, 1 implies *L** only, for monochrome data.

Compression: same as other multi-bit formats. JPEG compression applies.

PlanarConfiguration: both chunky and planar data could be supported.

WhitePoint: does not apply

PrimaryChromaticities: does not apply.

TransferFunction: does not apply

Alpha Channel information will follow the lead of other data types.

The reference white for this data type is the *perfect reflecting diffuser* (100% diffuse reflectance at all visible wavelengths). The *L** range is from 0 (perfect absorbing black) to 100 (perfect reflecting diffuse white). The *a** and *b** ranges will be represented as signed 8 bit values having the range -127 to +127.

Converting between RGB and CIELAB, a Caveat

The above CIELAB formulae are derived from CIE *XYZ*. Converting from CIELAB to *RGB* requires an additional set of formulae for converting between *RGB* and *XYZ*. For standard NTSC primaries these are:

0.60	0.17	0.20	R	X
0.29	0.58	0.11	G =	Y
0.00	0.06	1.11	B	Z

Generally, D65 illumination is used and a perfect reflecting diffuser is used for the reference white.

Since CIELAB is not a directly displayable format, some conversion to RGB will be required. While look-up table accelerated CIELAB to RGB conversion is certainly possible and fast, TIFF writers may choose to include a low resolution RGB subfile as an integral part of TIFF CIELAB.

Color Difference Measurements in CIELAB

The differences between two colors in L^*, a^*, and b^* are denoted by DL^*, Da^*, and Db^*, respectively, with the total (3-dimensional) color difference represented as:

$$\Delta E^*_{ab} = \left[(\Delta E^*)^2+(\Delta a^*)^2+(\Delta b^*)^2\right]^{1/2}.$$

This color difference can also be expressed in terms of L^*, C^*, and a measure of hue. In this case, h_{ab} is *not* used because it is an angular measure and cannot be combined with L^* and C^* directly. A linear-distance form of hue is used instead:

CIE 1976 *a,b hue-difference*, ΔH^*_{ab}

$$\Delta H^*_{ab} = \left[(\Delta E^*)^2-(\Delta L^*)^2-(\Delta C^*)^2\right]^{1/2}.$$

where DC^* is the chroma difference between the two colors. The total color difference expression using this hue-difference is:

$$\Delta E^*_{ab} = \left[(\Delta L^*)^2+(\Delta H^*)^2+(\Delta b^*)^2\right]^{1/2}.$$

It is important to remember that color difference is 3-dimensional: much more can be learned from a DL*a*b* triplet than from a single DE value. The DL*C*H* form is often the most useful since it gives the error information in a form that has more familiar perception correlates. Caution is in order, however, when using DH* for large hue differences since it is a straight-line approximation of a curved hue distance.

The Merits of CIELAB

Colorimetric.

First and foremost, CIELAB is colorimetric. It is traceable to the internationally-recognized standard CIE 1931 Standard Observer. This insures that it encodes color in a manner that is accurately modeled after the human vision system. Colors seen as matching are encoded identically, and colors seen as not matching are encoded differently. CIELAB provides an unambiguous definition of color without the necessity of additional information such as with RGB (primary chromaticities, white point, and gamma curves).

Device Independent.

Unlike RGB spaces which associate closely with physical phosphor colors, CIELAB contains no device association. CIELAB is not tailored for one device or device type at the expense of all others.

Full Color Gamut.

Any one image or imaging device usually encounters a very limited subset of the entire range of humanly-perceptible color. Collectively, however, these images and devices span a much larger gamut of color. A truly versatile exchange color space should encompass all of these colors, ideally providing support for all visible color. RGB, PhotoYCC, YCbCr, and other display spaces suffer from gamut limitations that exclude significant regions of easily printable colors. CIELAB is defined for all visible color.

Efficiency

A good exchange space will maximize accuracy of translations between itself and other spaces. It will represent colors compactly for a given accuracy. These attributes are provided through visual uniformity. One of the greatest disadvantages of the classic CIE system (and RGB systems as well) is that colors within it are not equally spaced visually. Encoding full-color images in a linear-intensity space, such as the typical RGB space or, especially, the XYZ space, requires a very large range (greater than 8-bits/primary) to eliminate banding artifacts. Adopting a *non*-linear RGB space improves the efficiency but not nearly to the extent as with a perceptually uniform space where these problems are nearly eliminated. A uniform space is also more efficiently compressed (see below).

Public Domain / Single Standard

CIELAB maintains no preferential attachments to any private organization. Its existence as a single standard leaves no room for ambiguity. Since 1976, CIELAB has continually gained popularity as a widely-accepted and heavily-used standard.

Luminance/Chrominance Separation.

The advantages for image size compression made possible by having a separate lightness or luminance channel are immense. Many such spaces exist. The degree to which the luminance information is fully-isolated into a single channel is an important consideration. Recent studies (Kasson and Plouffe of IBM) support CIELAB as a leading candidate placing it above CIELUV, YIQ, YUV, YCC, and XYZ.

Other advantages support a separate lightness or luminance channel. Tone and contrast editing and detail enhancement are most easily accomplished with such a channel. Conversion to a black and white representation is also easiest with this type of space.

When the chrominance channels are encoded as opponents as with CIELAB, there are other compression, image manipulation, and white point handling advantages.

Compressibility (Data).

Opponent spaces such as CIELAB are inherently more compressible than tristimulus spaces such as RGB. The chroma content of an image can be compressed to a greater extent, without objectionable loss, than can the lightness content. The opponent arrangement of CIELAB allows for spatial subsampling and efficent compression using JPEG.

Compressibility (Gamut).

Adjusting the color range of an image to match the capabilities of the intended output device is a critical function within computational color reproduction. Luminance/chrominance separation, especially when provided in a polar form, is desirable for facilitating gamut compression. Accurate gamut compression in a tri-linear color space is difficult.

CIELAB has a polar form (*metric hue angle,* and *metric chroma*, described below) that serves compression needs fairly well. Because CIELAB is not perfectly uniform, problems can arise when compressing along constant hue lines. Noticeable hue errors are sometimes introduced. This problem is no less severe with other contending color spaces.

This polar form also provides advantages for local color editing of images. The polar form is not proposed as part of the TIFF addition.

Getting the Most from CIELAB

Image Editors

The advantages of image editing within a perceptually uniform polar color space are tremendous. A detailed description of these advantages is beyond the scope of this section. As previously mentioned, many common tonal manipulation tasks are most efficiently performed when only a single channel is affected. Edge enhancement, contrast adjustment, and general tone-curve manipulation all ideally affect only the lightness component of an image.

A perceptual polar space works excellently for specifying a color range for masking purposes. For example, a red shirt can be quickly changed to a green shirt without drawing an outline mask. The operation can be performed with a loosely, quickly-drawn mask region combined with a hue (and perhaps chroma) range that encompasses the shirt's colors. The hue component of the shirt can then be adjusted, leaving the lightness and chroma detail in place.

Color cast adjustment is easily realized by shifting either or both of the chroma channels over the entire image or blending them over the region of interest.

Converting from CIELAB to a device specific space

For fast conversion to an RGB display, CIELAB can be decoded using 3x3 matrixing followed by gamma correction. The computational complexity required

for accurate CRT display is the same with CIELAB as with extended luminance-chrominance spaces.

Converting CIELAB for accurate printing on CMYK devices requires computational complexity no greater than with *accurate* conversion from any other colorimetric space. Gamut compression becomes one of the more significant tasks for any such conversion.

Part 3: Appendices

Part 3 contains additional information that is not part of the TIFF specification, but may be of use to developers.

Appendix A: TIFF Tags Sorted by Number

TagName	Decimal	Hex	Type	Number of values
NewSubfileType	254	FE	LONG	1
SubfileType	255	FF	SHORT	1
ImageWidth	256	100	SHORT or LONG	1
ImageLength	257	101	SHORT or LONG	1
BitsPerSample	258	102	SHORT	SamplesPerPixel
Compression	259	103	SHORT	1
Uncompressed	1			
CCITT 1D	2			
Group 3 Fax	3			
Group 4 Fax	4			
LZW	5			
JPEG	6			
PackBits	32773			
PhotometricInterpretation	262	106	SHORT	1
WhiteIsZero	0			
BlackIsZero	1			
RGB	2			
RGB Palette	3			
Transparency mask	4			
CMYK	5			
YCbCr	6			
CIELab	8			
Threshholding	263	107	SHORT	1
CellWidth	264	108	SHORT	1
CellLength	265	109	SHORT	1
FillOrder	266	10A	SHORT	1
DocumentName	269	10D	ASCII	
ImageDescription	270	10E	ASCII	
Make	271	10F	ASCII	
Model	272	110	ASCII	
StripOffsets	273	111	SHORT or LONG	StripsPerImage
Orientation	274	112	SHORT	1
SamplesPerPixel	277	115	SHORT	1
RowsPerStrip	278	116	SHORT or LONG	1
StripByteCounts	279	117	LONG or SHORT	StripsPerImage
MinSampleValue	280	118	SHORT	SamplesPerPixel
MaxSampleValue	281	119	SHORT	SamplesPerPixel
XResolution	282	11A	RATIONAL	1
YResolution	283	11B	RATIONAL	1
PlanarConfiguration	284	11C	SHORT	1
PageName	285	11D	ASCII	
XPosition	286	11E	RATIONAL	
YPosition	287	11F	RATIONAL	
FreeOffsets	288	120	LONG	
FreeByteCounts	289	121	LONG	
GrayResponseUnit	290	122	SHORT	1

GrayResponseCurve	291	123	SHORT	2**BitsPerSample
T4Options	292	124	LONG	1
T6Options	293	125	LONG	1
ResolutionUnit	296	128	SHORT	1
PageNumber	297	129	SHORT	2
TransferFunction	301	12D	SHORT	{1 or SamplesPerPixel}* 2** BitsPerSample
Software	305	131	ASCII	
DateTime	306	132	ASCII	20
Artist	315	13B	ASCII	
HostComputer	316	13C	ASCII	
Predictor	317	13D	SHORT	1
WhitePoint	318	13E	RATIONAL	2
PrimaryChromaticities	319	13F	RATIONAL	6
ColorMap	320	140	SHORT	3 * (2**BitsPerSample)
HalftoneHints	321	141	SHORT	2
TileWidth	322	142	SHORT or LONG	1
TileLength	323	143	SHORT or LONG	1
TileOffsets	324	144	LONG	TilesPerImage
TileByteCounts	325	145	SHORT or LONG	TilesPerImage
InkSet	332	14C	SHORT	1
InkNames	333	14D	ASCII	total number of characters in all ink name strings, including zeros
NumberOfInks	334	14E	SHORT	1
DotRange	336	150	BYTE or SHORT	2, or 2* NumberOfInks
TargetPrinter	337	151	ASCII	any
ExtraSamples	338	152	BYTE	number of extra components per pixel
SampleFormat	339	153	SHORT	SamplesPerPixel
SMinSampleValue	340	154	Any	SamplesPerPixel
SMaxSampleValue	341	155	Any	SamplesPerPixel
TransferRange	342	156	SHORT	6
JPEGProc	512	200	SHORT	1
JPEGInterchangeFormat	513	201	LONG	1
JPEGInterchangeFormatLngth	514	202	LONG	1
JPEGRestartInterval	515	203	SHORT	1
JPEGLosslessPredictors	517	205	SHORT	SamplesPerPixel
JPEGPointTransforms	518	206	SHORT	SamplesPerPixel
JPEGQTables	519	207	LONG	SamplesPerPixel
JPEGDCTables	520	208	LONG	SamplesPerPixel
JPEGACTables	521	209	LONG	SamplesPerPixel
YCbCrCoefficients	529	211	RATIONAL	3
YCbCrSubSampling	530	212	SHORT	2
YCbCrPositioning	531	213	SHORT	1
ReferenceBlackWhite	532	214	LONG	2*SamplesPerPixel
Copyright	33432	8298	ASCII	Any

Appendix B: Operating System Considerations

Extensions and Filetypes

The recommended MS-DOS, UNIX, and OS/2 file extension for TIFF files is ".TIF".

On an Apple Macintosh computer, the recommended Filetype is "TIFF". It is a good idea to also name TIFF files with a ".TIF" extension so that they can easily imported if transferred to a different operating system.

Index

Appendix C

C. OpenFlight v16.0 Technical Description - Annotated

This document has been annotated to reflect the conventions established by the CDB Specification. 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

OpenFlight Scene Description Database Specification, version 16.0. November, 2004

Printed in the U.S.A.
November 2004

ADOPTER REGISTRATION AGREEMENTFOROPENFLIGHT® SCENE DESCRIPTION DATABASE SPECIFICATION

The purpose of this agreement is to enable third parties who agree to adopt the MultiGen-Paradigm OpenFlight Scene Description Database Specification, on a non-exclusive basis, to receive ongoing access to technical updates to OpenFlight. Registered "Adopters" may also receive marketing support on MultiGen-Paradigm, Inc.'s World Wide Web (WWW) site once your product is completed. To become a registered adopter, complete and sign this registration form and return to MultiGen-Paradigm, Inc., 550 S. Winchester Blvd, Ste 500, San Jose, CA 95128, Attn.: OpenFlight Registration c/o Todd Griffith, or fax the completed form to (408) 878-0895.

_____, whose place of business is

_____, (hereinafter "User")

desires to obtain a copy of the OpenFlight Scene Description Database Specification (hereinafter "OpenFlight"). OpenFlight contains information belonging to MultiGen-Paradigm, Inc., a California corporation located in San Jose, California. The parties wish to define their rights with respect to OpenFlight. Therefore, it is agreed as follows:

OpenFlight is the property of MultiGen-Paradigm, Inc. and is protected under the copyright and trademark laws of the United States of America. MultiGen-Paradigm, Inc. hereby grants to User a non-exclusive, non-transferable limited right to use OpenFlight as follows:

a. For reading OpenFlight into a computer program or database for in-house use, or as a feature of a commercial product.

b. For writing data from a computer program or database into OpenFlight for in-house use or as a feature of a commercial product.

Any attempt to sub-license, assign, or transfer all or any part of the OpenFlight Specification is prohibited without the prior written consent of MultiGen-Paradigm, Inc.

OpenFlight, MultiGen and Creator are registered trademarks of MultiGen-Paradigm, Inc. User agrees to indicate that MultiGen-Paradigm, Inc. is owner of its own trademarks in any of User's published references to such trademarks. User shall not at any time use any name or trademark which is confusingly similar to a MultiGen-Paradigm, Inc. trademark.

OPENFLIGHT IS OFFERED FOR USE BY USER "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. ALL SUCH WARRANTIES ARE EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL MULTIGEN-PARADIGM, INC. BE RESPONSIBLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF USE OF OPENFLIGHT.

Adopter (User):

Signature _____ Date _____

Print Name _____ Title _____

OpenFlight® Scene Description Database Specification version 16.0 (November, 2004)

Contents

1 *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

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

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

❸ = 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.

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

This document describes the OpenFlight Scene Description Database Specification, commonly referred to as simply "OpenFlight". OpenFlight is a 3D scene description file format that was created and is maintained by MultiGen-Paradigm, Inc. While OpenFlight databases are typically created and edited using MultiGen-Paradigm software tools, the format is widely adopted and as a result, many tools exist to read and write OpenFlight database files.

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.

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.

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.

Container nodes are nodes that impose some logical grouping or behavior on the set of nodes it contains. The group node, for example allows you to “collect” similar nodes under one common parent for whatever reason your application needs. You might choose to group your nodes spatially or by some other criteria important to your application. Another common container node, the level of detail node, imposes a particular visual behavior on the nodes it contains. It defines a range of distances inside which the nodes it contains are visible.

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

Vertex nodes are the building blocks of geometry nodes. Individually, a vertex node represents a discrete point in space. Collected together under a geometry node such as a face node, a set of vertex nodes define a closed (or unclosed) loop. A closed loop of vertex nodes defines a face (or polygon). The “front” side of the face is determined by the ordering in which the vertex nodes appear under the face node. An unclosed loop of vertex nodes defines a set of line segments, again oriented according to the order in which the vertex nodes appear.

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

Header: There is one header node per database file. It is always the first node in the file and represents the top of the database hierarchy and tree structure. For more information, see “Header Record” on page 19.

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" on page 22.

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" on page 25.

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" on page 26.

Mesh: A mesh node defines geometric primitives that share attributes and vertices. See For more information, see "Mesh Nodes" on page 28.

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" on page 34.

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" on page 37.

Subface: A subface node is a face node that is assumed to be coplanar to, and drawn on top of, its superface. Subfaces can themselves be superfaces to allow multiple levels of "nesting". This construct resolves the display of coplanar faces. A subface is introduced, after a face node, by a push subface control record and concluded by a pop subface control record. Note that the OpenFlight format does not enforce a subface to be coplanar with its superface but this is recommended.

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" on page 44.

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" on page 43.

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" on page 47.

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" on page 39, "Morph Vertex List Record" on page 39 and "Vertex Palette Records" on page 66.

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 For more information, see "Morph Vertex List Record" on page 39.

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" on page 47.

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" on page 37.

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" on page 41.

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" on page 49.

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" on page 41.

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.

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.

Database palettes are not limited to supporting indirect attribute mapping. The vertex palette for example, defines a set of "shared" vertex nodes that can be indirectly referenced by multiple faces and/or light point nodes in the database. Similar to the space savings achieved by attribute palettes, the vertex palette also saves much disk space in the OpenFlight file when many geometry nodes share references to the same exact point in space (vertex).

All the database palettes supported by OpenFlight are described in "Palette Records" on page 66. Specific palettes in OpenFlight include:

•"Color Palette Record" on page 70
•"Material Palette Record" on page 71
•"Texture Palette Record" on page 73
•"Texture Mapping Palette Record" on page 86
•"Sound Palette Record" on page 80
•"Line Style Palette Record" on page 85
•"Light Source Palette Record" on page 81
•"Light Point Appearance Palette Record" on page 82
•"Light Point Animation Palette Record" on page 85
•"Vertex Palette Records" on page 66
•"Name Table Record" on page 71
•"Eyepoint and Trackplane Palette Record" on page 73
•"Linkage Palette Record" on page 77

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" on page 18. External reference records are described in "External Reference Record" on page 41.

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.

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.

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" on page 54 for more information.

2 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 an 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" on page 65**.**

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.

Continuation records are used to "continue" a record in the OpenFlight Scene Description file stream, when the original record is larger than 65535 bytes. The continuation record appears in the stream immediately following the record that it "continues". 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. Multiple continuation records may follow a record, in which case all continuation records would be appended (in sequence) to the original record

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.

The fields within each OpenFlight record are stored in big-endian byte order. OpenFlight database files have the extension ".flt" by convention.

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	❶
Unsigned Int	22	2	Vertex reference index; -1 if not vertex extension	❶

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	❶
Unsigned Int	22	2	Vertex reference index; -1 if not vertex extension	❶

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	❷

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	❶
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	❶
Int	6	2	Instance definition number	✓

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	❸
12	Flight V12	❸
14	OpenFlight v14.0 and v14.1	❸
1420	OpenFlight v14.2	❸
1510	OpenFlight v15.1	❸
1540	OpenFlight v15.4	❸
1550	OpenFlight v15.5	❸
1560	OpenFlight v15.6	❸
1570	OpenFlight v15.7	❸
1580	OpenFlight v15.8	❸
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.

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	✓
Char	4	8	7 char ASCII ID; 0 terminates (usually set to “db”)	✓
Int	12	4	Format revision level	✓
Int	16	4	Edit revision level	❶
Char	20	32	Date and time of last revision	❶
Int	52	2	Next Group node ID number	❶
Int	54	2	Next LOD node ID number	❶
Int	56	2	Next Object node ID number	❶
Int	58	2	Next Face node ID number	❶
Int	60	2	Unit multiplier (always 1)	❶
Int	62	1	Vertex coordinate units	✓
			0 = Meters	✓
			1 = Kilometers	❸
			4 = Feet	❸
			5 = Inches	❸
			8 = Nautical miles	❸
Int	63	1	if TRUE set texwhite on new faces	❶
Int	64	4	Flags (bits, from left to right)	❶
			0 = Save vertex normals	❶
			1 = Packed Color mode	❶
			2 = CAD View mode	❶
			3-31 = Spare	❶
Int	68	4*6	Reserved	❶
Int	92	4	Projection type	✓
			0 = Flat earth	✓
			1 = Trapezoidal	❶
			2 = Round earth	❶
			3 = Lambert	❶
			4 = UTM	❶
			5 = Geodetic	✓
			6 = Geocentric	❶
Int	96	4*7	Reserved	❶
Int	124	2	Next DOF node ID number	❶
Int	126	2	Vertex storage type	❶
			1 = Double precision float - should always be 1	❶

Header Record (Continued)

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	128	4	Database origin	❶
			100 = OpenFlight	❸
			200 = DIG I/DIG II	❸
			300 = Evans and Sutherland CT5A/CT6	❸
			400 = PSP DIG	❸
			600 = General Electric CIV/CV/PT2000	❸
			700 = Evans and Sutherland GDF	❸
Double	132	8	Southwest database coordinate x	❶
Double	140	8	Southwest database coordinate y	❶
Double	148	8	Delta x to place database	❶
Double	156	8	Delta y to place database	❶
Int	164	2	Next sound node ID number	❶
Int	166	2	Next path node ID number	❶
Int	168	4*2	Reserved	❶
Int	176	2	Next Clip node ID number	❶
Int	178	2	Next Text node ID number	❶
Int	180	2	Next BSP node ID number	❶
Int	182	2	Next Switch node ID number	❶
Int	184	4	Reserved	❶
Double	188	8	Southwest corner latitude	❶
Double	196	8	Southwest corner longitude	❶
Double	204	8	Northeast corner latitude	❶
Double	212	8	Northeast corner longitude	❶
Double	220	8	Origin latitude	❶
Double	228	8	Origin longitude	❶
Double	236	8	Lambert upper latitude	❶
Double	244	8	Lambert lower latitude	❶
Int	252	2	Next Light source node ID number	❶
Int	254	2	Next Light point node ID number	❶
Int	256	2	Next Road node ID number	❶
Int	258	2	Next CAT node ID number	❶
Int	260	2	Reserved	❶
Int	262	2	Reserved	❶
Int	264	2	Reserved	❶
Int	266	2	Reserved	❶
Int	268	4	Earth ellipsoid model	❶
			0 = WGS 1984	❸

			1 = WGS 1972	❸
			2 = Bessel	❸
			3 = Clarke 1866	❸
			4 = NAD 1927	❸
			-1 = User defined ellipsoid	❸
Int	272	2	Next Adaptive node ID number	❶
Int	274	2	Next Curve node ID number	❶
Int	276	2	UTM zone (for UTM projections - negative value means Southern hemisphere)	❶
Char	278	6	Reserved	❶

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)	❶
Double	292	8	Radius (distance from database origin to farthest corner)	❶
Unsigned int	300	2	Next Mesh node ID number	❶
Unsigned int	302	2	Next Light Point System ID number	❶
Int	304	4	Reserved	❶
Double	308	8	Earth major axis (for user defined ellipsoid) in meters	❶
Double	316	8	Earth minor axis (for user defined ellipsoid) in meters	❶

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.**

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	❶
Int	16	4	Flags (bits, from left to right)	✓
			0 = Reserved	❸
			1 = Forward animation	✓
			2 = Swing animation	✓
			3 = Bounding box follows	❸
			4 = Freeze bounding box	❸
			5 = Default parent	❸
			6 = Backward animation	✓
			7-31 = Spare	❸
Int	20	2	Special effect ID1 - application defined	❶
Int	22	2	Special effect ID2 - application defined	❶
Int	24	2	Significance	✓per CDB convention
Int	26	1	Layer code	❶
Int	27	1	Reserved	❶
Int	28	4	Reserved	❶
Int	32	4	Loop count	✓
Float	36	4	Loop duration in seconds	✓
Float	40	4	Last frame duration in seconds	✓

Here **are some examples that show how the values of the animation flags (forward animation, backward animation and swing animation) affect the animation. Note that these flags define how one "loop" of the animation sequence behaves.**

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.**

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.
T_1	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 it 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.

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	❸
Int	16	2	Relative priority	✓
Unsigned Int	18	2	Transparency	✓
			0 = Opaque	

			65535 = Totally clear	
Int	20	2	Special effect ID1 - application defined	❶
Int	22	2	Special effect ID2 - application defined	❶
Int	24	2	Significance	✓ Per CDB conventions
Int	26	2	Reserved	❶

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" on page 71**. 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" on page 25, 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.

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	❶
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)	✓
			2 = Draw wireframe and close	❸
			3 = Draw wireframe	❸
			4 = Surround with wireframe in alternate color	❸
			8 = Omnidirectional light	❸
			9 = Unidirectional light	❸
			10 = Bidirectional light	❸
Int	19	1	Texture white = if TRUE, draw textured face white	❶
Unsigned Int	20	2	Color name index	✓
Unsigned Int	22	2	Alternate color name index	✓
Int	24	1	Reserved	❶
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	❶ 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)	❶
Int	36	4	IR material code	❶
Unsigned Int	40	2	Transparency	✓
			0 = Opaque	
			65535 = Totally clear	
Unsigned Int	42	1	LOD generation control	❹

Unsigned Int	43	1	Line style index	❶
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	❶
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	✓
			3 = Use vertex colors and vertex normals	✓
Char	49	7	Reserved	❶
Unsigned Int	56	4	Packed color, primary (a, b, g, r) - only b, g, r used	✓
Unsigned Int	60	4	Packed color, alternate (a, b, g, r) - only b, g, r used	✓
Int	64	2	Texture mapping index	❶
Int	66	2	Reserved	❶
Unsigned Int	68	4	Primary color index	✓
Unsigned Int	72	4	Alternate color index	✓
Int	76	2	Reserved	❶
Int	78	2	Shader index, -1 if none	❶

Mesh Nodes

A mesh node defines a set of geometric primitives that share attributes and vertices. Prior to OpenFlight version 15.7, the fundamental geometric construct was the face (polygon) which was represented by a unique set of attributes and vertices. Meshes, by contrast, 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 consists of 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
```

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" on page 26.

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	❶
Int	16	4	IR color code	❶
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	❸
			3 = Draw wireframe	❸
			4 = Surround with wireframe in alternate color	❸
			8 = Omnidirectional light	❸
			9 = Unidirectional light	❸
			10 = Bidirectional light	❸
Int	23	1	Texture white = if TRUE, draw textured face white	❶
Unsigned Int	24	2	Color name index	✓
Unsigned Int	26	2	Alternate color name index	✓
Int	28	1	Reserved	❶
Int	29	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	30	2	Detail texture pattern index, -1 if none	❶ Note: Detail textures are IRIS GL specific
Int	32	2	Texture pattern index, -1 if none	✓
Int	34	2	Material index, -1 if none	✓

Int	36	2	Surface material code (for DFAD)	✓ (tentative)
Int	38	2	Feature ID (for DFAD)	❶
Int	40	4	IR material code	❶
Unsigned Int	44	2	Transparency	✓
			0 = Opaque	
			65535 = Totally clear	
Unsigned Int	46	1	LOD generation control	❹
Unsigned Int	47	1	Line style index	❶
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	❶
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	❶
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	❶
Int	70	2	Reserved	❶
Unsigned Int	72	4	Primary color index	✓
Unsigned Int	76	4	Alternate color index	✓
Int	80	2	Reserved	❶
Int	82	2	Shader index, -1 if none	❶

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.

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 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" on page 65.)	✓
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	✓
			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	❶

Local Vertex Pool Record (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.				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	colorN - Color for vertex N - present if Attribute mask includes Has Color Index or Has RGBA Color. If Has Color Index, lower 3 bytes specify color table index, upper 1 byte is Alpha. If Has RGBA Color, 4 bytes specify (a, b, g, r) values.	✓
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.

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	$Index_N$ - 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.

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.

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	❶
Int	24	4	Draw order (for calligraphic lights)	❶
Int	28	4	Reserved	❶

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.

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

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 = 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	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.

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.

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	❶
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)	✓

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	❸
			10 = Reserved	❸
			11-31 = Spare	❸
Int	380	4	Reserved	❶

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" on page 66. **A vertex node is a leaf node in the database and therefore cannot have any children.**

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	$Offset_N$ - 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" on page 66. **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.

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 0_N - Offset into vertex palette of Nth 0% vertex.	✓
Int	8+(N* 8)	4	Offset 100_N - Offset into vertex palette of Nth 100% vertex.	✓

Binary Separating Plane Record

The binary separating plane (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.

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	❶
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.

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	❶
Int	208	4	Flags (bits, from left to right)	❶
			0 = Color palette override	❶
			1 = Material palette override	❶
			2 = Texture and texture mapping palette override	❶
			3 = Line style palette override	❶
			4 = Sound palette override	❶
			5 = Light source palette override	❶
			6 = Light point palette override	❶
			7 = Shader palette override	❶
			8-31 = Spare	❶
Int	212	2	View as bounding box	❶
			0 = View external reference normally	❶
			1 = View external reference as bounding box	❶

Int	214	2	Reserved	1

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.

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	❶
Double	16	8	Switch-in distance	✓
Double	24	8	Switch-out distance	✓
Int	32	2	Special effect ID1 - application defined	❶
Int	34	2	Special effect ID2 - application defined	❶
Int	36	4	Flags (bits, from left to right)	✓
			0 = Use previous slant range	✓
			1 = Reserved	❸
			2 = Freeze center (don't recalculate)	❹
			3-31 = Spare	❸
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.

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).

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.

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.

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

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," page 99**.**

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.

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:
			a_0, a_1, a_2, a_3, a_4
			b_0, b_1, b_2, b_3, b_4
			c_0, c_1, c_2, c_3, c_4
			d_0, d_1, d_2, d_3, d_4

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:

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" on page 53.

Switch Record

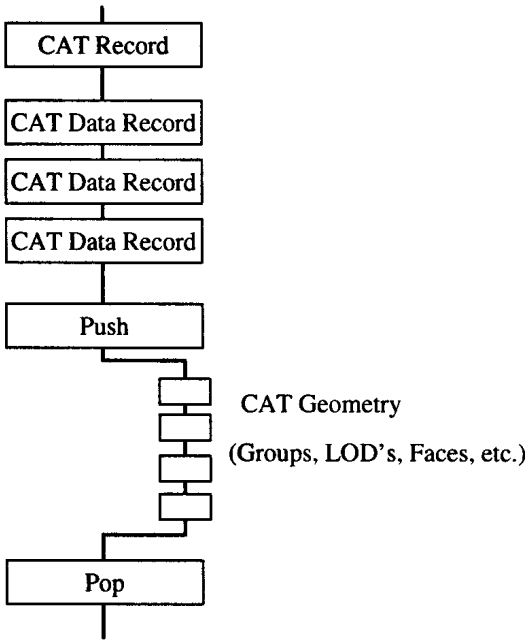
Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Switch Opcode 96	✓
Unsigned Int	2	2	Length of record	✓
Char	4	8	7 char ASCII ID; 0 terminates	✓
Int	12	4	Reserved	❶
Int	16	4	Current mask	✓
Int	20	4	Number of masks	✓
Int	24	4	Number of words per mask - the number of 32 bit words required 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.



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

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" on page 64.**)**

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 G_0 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,..., 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.

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: cp_0x, cp_0y, cp_0z, cp_1x, cp_1y, cp_1z, ... cp_Nx, cp_Ny, cp_Nz 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.

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.

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.

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	✓

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.

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 multitexture 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.	✓
			5 Has Layer 6 - if set, multitexture information for texture layer 6 is present.	✓
			6 Has Layer 7 - if set, multitexture information for texture layer 7 is present.	✓
			7-31 Spare	❸
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, Inc.	❸
			>100 = user (runtime) defined	❸
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	✓

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.

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	✓
			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	❶

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	$u_{i, N}$ - Texture coordinate U for vertex i, layer N
Float	4	$v_{i, N}$ - 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	$u0_{i, N}$ - Texture U for the 0% vertex i, layer N
Float	4	$v0_{i, N}$ - Texture V for the 0% vertex i, layer N
Float	4	$u100_{i, N}$ - Texture U for the 100% vertex i, layer N
Float	4	$v100_{i, N}$ - 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.

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	❶

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," page 101**.**

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

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.

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

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

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)

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

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

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

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)

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.

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

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.

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	❶
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	✓

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	❶
Double	8	8	Radius of the sphere	✓

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	❶
Double	8	8	Radius of the cylinder base	✓
Double	16	8	Height of the cylinder	✓

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 triangle$_N$	✓
Double	16+(N*72)	8	y coordinate of vertex 1 of triangle$_N$	✓
Double	24+(N*72)	8	z coordinate of vertex 1 of triangle$_N$	✓
Double	32+(N*72)	8	x coordinate of vertex 2 of triangle$_N$	✓
Double	40+(N*72)	8	y coordinate of vertex 2 of triangle$_N$	✓
Double	48+(N*72)	8	z coordinate of vertex 2 of triangle$_N$	✓
Double	56+(N*72)	8	x coordinate of vertex 3 of triangle$_N$	✓
Double	64+(N*72)	8	y coordinate of vertex 3 of triangle$_N$	✓
Double	72+(N*72)	8	z coordinate of vertex 3 of triangle$_N$	✓

Bounding Histogram Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Bounding Histogram Opcode 119	❷
Unsigned Int	2	2	Length - length of the record	❷
Char	4	Variable	The contents of this record is reserved for use by Multi-Gen-Paradigm.	❷

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	❶
Double	8	8	x coordinate of center	✓
Double	16	8	y coordinate of center	✓
Double	24	8	z coordinate of center	✓

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	❶
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" on page 76**.**

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.

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

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 index$_N$ - index of face N
Int	36+(N*12)	4	Reserved$_N$ - reserved space for face N
Int	40+(N*12)	4	Face data offset$_N$ - offset for face data record N in the CAT data. Note: This offset is measured relative to the Packed face data field in the CAT data face record described below.

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	LOD_N - Level of detail to which this face N belongs.
Int	Varies	4	Child index 1_N - The 1st child (within this table) of face N, or -1 for no face.
Int	Varies	4	Child index 2_N - The 2nd child (within this table) of face N, or -1 for no face.
Int	Varies	4	Child index 3_N - The 3rd child index (within this table) of face N, or -1 for no face.
Int	Varies	4	Child index 4_N - The 4th child index (within this table) of face N, or -1 for no face.
Int	Varies	4	ID $Length_N$ - length of face node ID string which follows
Char	Varies	Varies	ID_N - 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" on page 51.**)**

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.

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" on page 51
- **"**Name Table Record" on page 71
- **"**Local Vertex Pool Record" on page 30
- **"**Mesh Primitive Record" on page 32

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
```

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" on page 39 **and** "Morph Vertex List Record" on page 39 **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" on page 30. **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.

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.

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	❹
			2 = No color	✓
			3 = Packed color	✓
			4-15 = Spare	❸
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	✓

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	❹
			2 = No color	✓
			3 = Packed color	✓
			4-15 = Spare	❸
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	❶

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	❹
			2 = No color	✓
			3 = Packed color	✓
			4-15 = Spare	❸
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	✓

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	❹
			2 = No color	✓
			3 = Packed color	✓
			4-15 = Spare	❸
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	❶

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.

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	❶
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	$Length_N$ - 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 $name_N$ field.	✓
Int	Varies	2	$Reserved_N$ - reserved space for	❶

			color name N	
Int	Varies	2	Color index$_N$ - index of color in palette corresponding to color name N	✓
Int	Varies	2	Reserved$_N$ - reserved space for color name N	❶
Char	Varies	Length$_N$-8	Color name$_N$ - 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 the 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.

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	❶
Name Table Entry 0				
Int	10	4	Length$_0$ - length of entry 0	✓
Unsigned Int	14	2	Name index$_0$ - index corresponding to entry 0	✓
Char	16	Varies	Name string$_0$ - name for entry 0; 0 terminates. Variable length, maximum of 80 chars	✓
Name Table Entry 1				
Int	Varies	4	Length$_1$ - length of entry 1	✓
Unsigned Int	Varies	2	Name index$_1$ - index corresponding to entry 1	✓
Char	Varies	Varies	Name string$_1$ - 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	Length$_N$ - length of entry N	✓
Unsigned Int	Varies	2	Name index$_N$ - index corresponding to entry N	✓
Char	Varies	Varies	Name string$_N$ - 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:

Displayed ambient (red) = Material ambient (red)* geometry color (red)
Displayed ambient (green) = Material ambient (green)* geometry color (green)
Displayed ambient (blue) = Material ambient (blue)* geometry color (blue)

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:

Displayed diffuse (red) = Material diffuse (red)* geometry color (red)
Displayed diffuse (green) = Material diffuse (green)* geometry color (green)
Displayed diffuse (blue) = Material diffuse (blue)* geometry color (blue)

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))

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	❶
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	❶

* 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.**

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)	❹

Eyepoint and Trackplane Palette Record

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider the Eyepoint and Trackplane Palette Records.

OpenFlight files can contain up to ten eyepoint and trackplane positions. The first eyepoint and trackplane in the file is reserved as the "last" one set during the modeling session. The other nine are user-defined. Both the eyepoints and trackplanes are combined in the Eyepoint and Trackplane palette record which is described in this section.

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.

Eyepoint and Trackplane Palette Record (Continued)

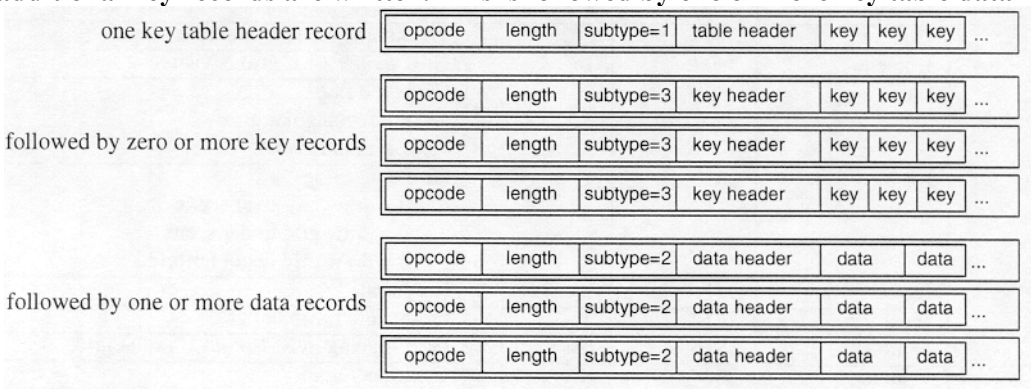
Data Type	Offset	Length	Description
The following fields are repeated for 10 trackplanes			
Trackplane 0 - 128 bytes			
Int	2728	4	Valid trackplane - 1 if this is a valid trackplane
Int	2732	4	Reserved
Double	2736	8*3	Trackplane origin coordinate (x, y, z)
Double	2760	8*3	Trackplane alignment coordinate (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" on page 108**.**

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 value$_N$ - key value N
Int	36+(N*12)	4	Reserved$_N$ - reserved space for key N, defined by record using key table for storage
Int	40+(N*12)	4	Data offset$_N$ - offset for data corresponding to key N. Note: This offset is measured relative to the Packed data field in the key table data record described below.

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" on page 103 **for parameter ID values and descriptions.**

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 value$_N$ - key value N
Int	36+(N*12)	4	Data type$_N$ - data type for key N
			0x12120001 = Node data
			0x12120002 = Arc data
			0x12120004 = Database entity name
Int	40+(N*12)	4	Data offset$_N$ - 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.

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.

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

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

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

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

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

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

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.

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 index$_N$ - index of sound N in the palette
Int	36+(N+12)	4	Reserved$_N$ - reserved space for sound N in the palette
Int	40+(N*12)	4	File name offset$_N$ - 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.

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.

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.

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	❶
Char	8	256	Light Point Type Name 0 terminates	✓
Int	264	4	Appearance	✓
Int	268	2	Surface material code	❶
Int	270	2	Feature ID	❶
Unsigned Int	272	4	Back color for bidirectional points	❶
Int	276	4	Display mode	❶
			0 = RASTER	❸
			1 = CALLIGRAPHIC	❸
			2 = EITHER	❸
Float	280	4	Intensity - scalar for front colors	❶
Float	284	4	Back intensity - scalar for back color	❶
Float	288	4	Minimum defocus - (0.0 - 1.0) for calligraphic points	❶
Float	292	4	Maximum defocus - (0.0 - 1.0) for calligraphic points	❶
Int	296	4	Fading mode	❶
			0 = Enable perspective fading calculations	❸
			1 = Disable calculations	❸
Int	300	4	Fog Punch mode	❶
			0 = Enable fog punch through calculations	❸
			1 = Disable calculations	❸
Int	304	4	Directional mode	❶
			0 = Enable directional calculations	❸
			1 = Disable calculations	❸
Int	308	4	Range mode	❶
			0 = Use depth (Z) buffer calculation	❸
			1 = Use slant range calculation	❸

Float	312	4	Min pixel size - minimum diameter of points in pixels	1
Float	316	4	Max pixel size - maximum diameter of points in pixels	1
Float	320	4	Actual size - actual diameter of points in database units	1
Float	324	4	Transparent falloff pixel size - diameter in pixels when points become transparent	1
Float	328	4	Transparent falloff exponent	1
			>= 0 - falloff multiplier exponent	3
			1.0 - linear falloff	3
Float	332	4	Transparent falloff scalar	1
			> 0 - falloff multiplier scale factor	3
Float	336	4	Transparent falloff clamp - minimum permissible falloff multiplier result	3
Float	340	4	Fog scalar	1
			>= 0 - adjusts range of points for punch threw effect.	3
Float	344	4	Fog intensity	1
Float	348	4	Size difference threshold - point size transition hint to renderer	1
Int	352	4	Directionality	1
			0 = OMNIDIRECTIONAL	3
			1 = UNIDIRECTIONAL	3
			2 = BIDIRECTIONAL	3
Float	356	4	Horizontal lobe angle - total angle in degrees	1
Float	360	4	Vertical lobe angle - total angle in degrees	1
Float	364	4	Lobe roll angle - rotation of lobe about local Y axis in degrees	1
Float	368	4	Directional falloff exponent	1
			>= 0 - falloff multiplier exponent	3
			1.0 - linear falloff	3
Float	372	4	Directional ambient intensity - of points viewed off axis	1
Float	376	4	Significance - drop out priority for RASCAL lights (0.0 - 1.0)	1
Int	380	4	Flags (bits, from left to right)	1
			0 = reserved	3
			1 = No back color	3
			TRUE = don't use back color for bidirectional points	3
			FALSE = use back color for bidirectional points	3
			2 = reserved	3
			3 = Calligraphic proximity occulting (Debunching)	3
			4 = Reflective, non-emissive point	3

			5-7 = Randomize intensity	3
			0 = never	3
			1 = low	3
			2 = medium	3
			3 = high	3
			8 = Perspective mode	3
			9 = Flashing	3
			10 = Rotating	3
			11 = Rotate Counter Clockwise	3
			Direction of rotation about local Z axis	3
			12 = reserved	3
			13-14 = Quality	3
			0 = Low	3
			1 = Medium	3
			2 = High	3
			3 = Undefined	3
			15 = Visible during day	3
			16 = Visible during dusk	3
			17 = Visible during night	3
			18-31 = Spare	3
Float	384	4	Visibility range (> 0.0)	1
Float	388	4	**Fade range ratio - percentage of total range at which light points start to fade (0.0 - 1.0)**	1
Float	392	4	**Fade in duration - time it takes (seconds) light point to fade in when turned on**	1
Float	396	4	**Fade out duration - time it takes (seconds) light point to fade out when turned off**	1
Float	400	4	**LOD range ratio - percentage of total range at which light points LODs are active (0.0 - 1.0)**	1
Float	404	4	**LOD scale - size of light point LOD polygon relative to light point diameter**	1
Int	408	2	Texture pattern index, -1 if none	1
Int	410	2	Reserved	1

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.

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 State$_N$ - state of sequence N
			0 = On
			1 = Off
			2 = Color change
Float	1344+(N*12)	4	Sequence Duration$_N$ - duration of sequence N in seconds
Unsigned Int	1348+(N*12)	4	Sequence Color$_N$ - 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.

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.

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.

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

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.

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 map- ping palette record if Texture mapping type is 5.

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.

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.

Shader Palette Record

Data Type	Offset	Length	Description	CDB OpenFlight Reader
Int	0	2	Shader Opcode 133	❶
Unsigned Int	2	2	Length - length of the record	❶
Int	4	4	Shader index	❶
Int	8	4	Shader type	❶
			0 = Cg	❸
			1 = CgFX	❸
			2 = OpenGL Shading Language	❸
char	12	1024	Shader name; 0 terminates	❶
char	1036	1024	Vertex program file name; 0 terminates (Cg Shader type specific)	❶
char	2060	1024	Fragment program file name; 0 terminates (Cg Shader type specific)	❶
Int	3084	4	Vertex program profile (Cg Shader type specific)	❶
Int	3088	4	Fragment program profile (Cg Shader type specific)	❶
char	3092	256	Vertex program entry point (Cg Shader type specific)	❶
char	3348	256	Fragment program entry point (Cg Shader type specific)	❶

3 *Texture Files*

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" on page 73.

File formats currently supported include:

•AT&T® image 8 format (8-bit color lookup)	**CDB OpenFlight Readers**
•AT&T image 8 template format	❶
•SGI intensity modulation (*.int)	✓[8]
•SGI intensity modulation with alpha (*.inta)	✓8
•SGI RGB (*.rgb)	✓
•SGI RGB with alpha (*.rgba)	✓8
•GIF	❶
•JPEG/JFIF (*.jpg)	❶
•TIFF (*.tif)	❶
•IFF/ILBM	❶
•BMP/DIB	❶
•PCX	❶
•PNG	❶
•PPM	❶
•Sun™ Raster	❶
•Direct Draw Surface (DDS)	❶
•Targa™	❶
•Alias™ Pix	❶
•SGI clip texture	❶

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.

[8] The SGI format is fully supported by the CDB Specification 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.

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.

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:

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 u_N - texel u of control point
Double	8	Texel v_N - texel v of control point
Double	8	Earth coordinate x_N - 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.

Subtexture subrecord

Data Type	Offset	Length
Char	32	$Name_N$ - name of subtexture N; 0 terminates
Int	4	$Left_N$ - Coordinate of left edge of subtexture N
Int	4	$Bottom_N$ - Coordinate of bottom edge of subtexture N
Int	4	$Right_N$ - Coordinate of right edge of subtexture N
Int	4	Top_N - Coordinate of top edge of subtexture N

4 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                                Horizontal and vertical curve control
SUPERELEVATION: 0.080000                              values
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 information for the road section
LANE_OFFSET: 1.825000
LANE_OFFSET: -1.825000
PROFILE_NAME: /usr/people/db/road/crown.flt           Lofting information, including the database file that
PROFILE_POINT: 12.000000 0.000000                     contains the lofting profile, and (X,Z) points for the
PROFILE_POINT: 9.823453 0.300000                      lofted section. Lofting information is only printed
PROFILE_POINT: 6.000000 0.500000                      for the first reference to the profile database.
PROFILE_POINT: 3.200000 0.300000
PROFILE_POINT: 0.000000 0.000000
SPEED: 70.000000                                      Passing lane flag (path attribute page)
NO_PASSING: TRUE                                      Heading, Pitch and Roll data will be stored and reported
STORE_HPR: TRUE
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
```

Data for each point, in the following order:

X, Y, Z, H, P, R

5 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.

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.

Elevation Data Point subrecord

Data Type	Offset	Length	Data Type
Float	520+(N*4)	4	Z_N - 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.

Surface Type subrecord

Data Type	Offset	Length	Data Type
Char	520+(M*4)+N	1	Surface type$_N$ - user defined surface type for data point N

6 *Linkage Editor Parameter IDs*

CDB OpenFlight Readers: CDB-compliant OpenFlight readers do not consider Linkage Editor Parameter IDs.

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

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

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

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

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

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

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

Switch Node Parameters

ID	Description
2050	Current mask index

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

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

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

7 OpenFlight Opcodes

Valid Opcodes

Opcode	Record Type	For more information, see…	CDB OpenFlight Reader
1	Header	"Header Record" on page 19	✓
2	Group	"Group Record" on page 22	✓
4	Object	"Object Record" on page 25	✓
5	Face	"Face Record" on page 26	✓
10	Push Level	"Push Level Record" on page 17	✓
11	Pop Level	"Pop Level Record" on page 17	✓
14	Degree of Freedom	"Degree of Freedom Record" on page 37	✓
19	Push Subface	"Push Subface Record" on page 17	✓
20	Pop Subface	"Pop Subface Record" on page 17	✓
21	Push Extension	"Push Extension Record" on page 17	❶
22	Pop Extension	"Pop Extension Record" on page 17	❶
23	Continuation	"Continuation Record" on page 65	✓
31	Comment	"Comment Record" on page 53	✓
32	Color Palette	"Color Palette Record" on page 70	✓
33	Long ID	"Long ID Record" on page 53	✓
49	Matrix	"Matrix Record" on page 59	✓
50	Vector	"Vector Record" on page 61	❶
52	Multitexture	"Multitexture Record" on page 54	✓
53	UV List	"UV List Record" on page 55	✓
55	Binary Separating Plane	"Binary Separating Plane Record" on page 40	✓
60	Replicate	"Replicate Record" on page 57	✓
61	Instance Reference	"Instance Reference Record" on page 19	✓
62	Instance Definition	"Instance Definition Record" on page 19	✓
63	External Reference	"External Reference Record"	✓

		on page 41	
64	Texture Palette	"Texture Palette Record" on page 73	✓
67	Vertex Palette	"Vertex Palette Record" on page 67	✓
68	Vertex with Color	"Vertex with Color Record" on page 68	✓
69	Vertex with Color and Normal	"Vertex with Color and Normal Record" on page 68	✓
70	Vertex with Color, Normal and UV	"Vertex with Color, Normal and UV Record" on page 69	✓
71	Vertex with Color and UV	"Vertex with Color and UV Record" on page 69	✓
72	Vertex List	"Vertex List Record" on page 39	✓
73	Level of Detail	"Level of Detail Record" on page 41	✓
74	Bounding Box	"Bounding Box Record" on page 62	✓
76	Rotate About Edge	"Rotate About Edge Record" on page 59	❶
78	Translate	"Translate Record" on page 59	❶
79	Scale	"Scale Record" on page 59	❶
80	Rotate About Point	"Rotate About Point Record" on page 60	❶
81	Rotate and/or Scale to Point	"Rotate and/or Scale to Point Record" on page 60	❶
82	Put	"Put Record" on page 60	❶
83	Eyepoint and Trackplane Palette	"Eyepoint and Trackplane Palette Record" on page 73	❶
84	Mesh	"Mesh Record" on page 29	✓
85	Local Vertex Pool	"Local Vertex Pool Record" on page 30	✓
86	Mesh Primitive	"Mesh Primitive Record" on page 32	✓
87	Road Segment	"Road Segment Record" on page 44	❶
88	Road Zone	"Road Zone Record" on page 58	❶
89	Morph Vertex List	"Morph Vertex List Record" on page 39	✓
90	Linkage Palette	"Linkage Palette Record" on page 77	❶
91	Sound	"Sound Record" on page 43	❶
92	Road Path	"Road Path Record" on page 46	❶
93	Sound Palette	"Sound Palette Record" on page 80	❶

94	General Matrix	"General Matrix Record" on page 60	❶
95	Text	"Text Record" on page 47	❶
96	Switch	"Switch Record" on page 49	✓
97	Line Style Palette	"Line Style Palette Record" on page 85	❶
98	Clip Region	"Clip Region Record" on page 47	❶
100	Extension	"Extension Record" on page 51	❶
101	Light Source	"Light Source Record" on page 44	❶
102	Light Source Palette	"Light Source Palette Record" on page 81	❶
103	Reserved		❶
104	Reserved		❶
105	Bounding Sphere	"Bounding Sphere Record" on page 62	✓
106	Bounding Cylinder	"Bounding Cylinder Record" on page 62	✓
107	Bounding Convex Hull	"Bounding Convex Hull Record" on page 62	✓
108	Bounding Volume Center	"Bounding Volume Center Record" on page 63	✓
109	Bounding Volume Orientation	"Bounding Volume Orientation Record" on page 63	✓
110	Reserved		❶
111	Light Point	"Light Point Record" on page 34	❶
112	Texture Mapping Palette	"Texture Mapping Palette Record" on page 86	❶
113	Material Palette	"Material Palette Record" on page 71	✓
114	Name Table	"Name Table Record" on page 71	✓
115	Continuously Adaptive Terrain (CAT)	"CAT Record" on page 50	❶
116	CAT Data	"CAT Data Record" on page 63	❶
117	Reserved		❶
118	Reserved		❶
119	Bounding Histogram	"Bounding Histogram Record" on page 62	❶
120	Reserved		❶
121	Reserved		❶
122	Push Attribute	"Push Attribute Record" on	❷

		page 18	
123	Pop Attribute	"Pop Attribute Record" on page 18	❷
124	Reserved		❶
125	Reserved		❶
126	Curve	"Curve Record" on page 51	❶
127	Road Construction	"Road Construction Record" on page 45	❶
128	Light Point Appearance Palette	"Light Point Appearance Palette Record" on page 82	✓
129	Light Point Animation Palette	"Light Point Animation Palette Record" on page 85	❶
130	Indexed Light Point	"Indexed Light Point Record" on page 34	✓
131	Light Point System	"Light Point System Record" on page 37	❶
132	Indexed String	"Indexed String Record" on page 53	✓
133	Shader Palette	"Shader Palette Record" on page 91	❶

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)

A *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.**

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:

Format Changes

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.

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)

Header Record

New attributes have been appended to the end of the existing header record (see "Header Record" on page 19). The following fields were added at the specified offsets in the header record.

Header Record changes for OpenFlight15.7
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

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
```

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" on page 26.

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

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

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.

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" on page 65.)
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

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	Coordinate$_N$ - Coordinate of vertex N (x, y, z) - present if Attribute mask includes Has Position.
Unsigned Int	Varies	4	color$_N$ - 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	normal$_N$ - Normal for vertex N (i, j, k) - present if Attribute mask includes Has Normal.
Float	Varies	4*2	uvBase$_N$ - Texture coordinates (u, v) for base texture layer of vertex N - present if Attribute mask includes Has Base UV.
Float	Varies	4*2	uv1$_N$ - Texture coordinates (u, v) for layer 1 of vertex N - present if Attribute mask includes Has UV Layer 1.
Float	Varies	4*2	uv2$_N$ - Texture coordinates (u, v) for layer 2 of vertex N - present if Attribute mask includes Has UV Layer 2.
Float	Varies	4*2	uv3$_N$ - Texture coordinates (u, v) for layer 3 of vertex N - present if Attribute mask includes Has UV Layer 3.
Float	Varies	4*2	uv4$_N$ - Texture coordinates (u, v) for layer 4 of vertex N - present if Attribute mask includes Has UV Layer 4.
Float	Varies	4*2	uv5$_N$ - Texture coordinates (u, v) for layer 5 of vertex N - present if Attribute mask includes Has UV Layer 5.
Float	Varies	4*2	uv6$_N$ - Texture coordinates (u, v) for layer 6 of vertex N - present if Attribute mask includes Has UV Layer 6.
Float	Varies	4*2	uv7$_N$ - 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.

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	$Index_N$ - 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.

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.

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.

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

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.

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	$u_{i, N}$ - Texture U for vertex i, layer N
Float	4	$v_{i, N}$ - 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	$u0_{i, N}$ - Texture U for the 0% vertex i, layer N
Float	4	$v0_{i, N}$ - Texture V for the 0% vertex i, layer N
Float	4	$u100_{i, N}$ - Texture U for the 100% vertex i, layer N
Float	4	$v100_{i, N}$ - Texture V for the 100% vertex i, layer N

Texture Attribute File

Subtexture

Subtexture definitions have been added to the end of the Texture Attribute File (see "Texture Attribute Files" on page 93). After all the geospecific control points are listed, the following subtexture information now appears:

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	Name$_N$ - name of subtexture N; 0 terminates
Int	4	Left$_N$ - Coordinate of left edge of subtexture N
Int	4	Bottom$_N$ - Coordinate of bottom edge of subtexture N
Int	4	Right$_N$ - Coordinate of right edge of subtexture N
Int	4	Top$_N$ - Coordinate of top edge of subtexture N

B *Summary of Changes Version 15.8*

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" on page 127
- "Group Record" on page 128
- "Level of Detail Record" on page 129
- "External Reference Record" on page 130
- "Indexed String Record" on page 130 (for Switch nodes)
- "Face Record" on page 131
- "Mesh Record" on page 131
- "Local Vertex Pool Record" on page 132
- "Vertex Palette Records" on page 133
- "Light Point Appearance Palette Record" on page 136
- "Light Point Animation Record" on page 139
- "Indexed Light Point Record" on page 140
- "Light Point System Record" on page 140
- "Texture Mapping Palette Record" on page 141

Also new in this version of the document is the addition of the "offset" column in the record format tables.

Document Corrections

The errors corrected in this version of the document are described in the sections that follow.

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.

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.

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" on page 47.

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.

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.

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.

CAT Record error corrected in OpenFlight 15.8 specification
Reserved field (documented)

Data Type	Offset	Length	Description
Int	60	4	Reserved

Format Changes

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.

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:

Header Record changes for OpenFlight15.8
Projection type field

Data Type	Offset	Length	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.

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.

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" on page 22.

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.

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.

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" on page 22.

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.

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" on page 41.

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.

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" on page 41.

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.

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.

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.

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

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.

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

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**:

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 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
			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.

Local Vertex Pool Record changes for OpenFlight15.8
Color field

Data Type	Offset	Description
Unsigned Int	4	**colorN - Color for vertex N - present if Attribute mask includes Has Color Index or Has RGBA Color. If Has Color Index, lower 3 bytes specify color table index, upper 1 byte is Alpha. If Has RGBA Color, 4 bytes specify (a, b, g, r) values.**

For a complete description of the local vertex pool record, see "Local Vertex Pool Record" on page 30**.**

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).

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

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

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)
			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

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" on page 66.

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.

Light Point Appearance Palette Record

The light point appearance palette record defines the visual attributes of light points.

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

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)

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**

Light Point Animation Record

The light point animation palette record defines the behavioral attributes of light points

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 State$_N$ - state of sequence N
			0 = On
			1 = Off
			2 = Color change
Float	1344+(N*12)	4	Sequence Duration$_N$ - duration of sequence N in seconds
Unsigned Int	1348+(N*12)	4	Sequence Color$_N$ - 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.

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.

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**.**

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.

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" on page 37**.**

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.

**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.

**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.

**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

C *Summary of Changes Version 16.0*

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" on page 147
- "Face Record" on page 148
- "Mesh Record" on page 148
- "Light Point Appearance Palette Record" on page 148
- "Shader Palette Record" on page 149
- "Texture Attribute File" on page 149
- "Texture Mapping Palette Record" on page 150

Document Corrections

The errors corrected in this version of the document are described in the sections that follow.

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.

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

Face Record

The possible values listed for the Draw type field have been corrected. The affected values are shown in bold font.

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.

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.

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.

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" on page 29.

Switch Record

The order of the fields were corrected. The affected fields are shown here.

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" on page 49.

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:

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" on page 86.

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:

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

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" on page 62**.**

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" on page 62**.**

Format Changes

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.

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 = Shader palette override
			8-31 = Spare

For a complete description of the external reference record, see "External Reference Record" on page 41**.**

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.

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" on page 26.

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.

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" on page 29.

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.

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 Palette Record" on page 82.

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.

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)

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:.

Texture Attribute File Format changes for OpenFlight 16.0Wrap 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

Texture Attribute File Format changes for OpenFlight 16.0Wrap 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.

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

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.

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

Index

U

V

Appendix D

D. ShapeFile and dBASE July 1998 Technical Description – Annotated

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

Annotated with CDB conventions

ESRI Shapefile Technical Description

An ESRI White Paper—July 1998

J-7855

ESRI Shapefile Technical Description

An ESRI White Paper

Contents **Page**

J-7855

ESRI Shapefile Technical Description

This document defines the shapefile (.shp) spatial data format and describes why shapefiles are important. It lists the tools available in Environmental Systems Research Institute, Inc. (ESRI), software for creating shapefiles directly or converting data into shapefiles from other formats. This document also provides all the technical information necessary for writing a computer program to create shapefiles without the use of ESRI® software for organizations that want to write their own data translators.

Why Shapefiles?

A shapefile stores nontopological geometry and attribute information for the spatial features in a data set. The geometry for a feature is stored as a shape comprising a set of vector coordinates.

Because shapefiles do not have the processing overhead of a topological data structure, they have advantages over other data sources such as faster drawing speed and edit ability. Shapefiles handle single features that overlap or that are noncontiguous. They also typically require less disk space and are easier to read and write.

Shapefiles can support point, line, and area features. Area features are represented as closed loop, double-digitized polygons. Attributes are held in a dBASE® format file. Each attribute record has a one-to-one relationship with the associated shape record.

How Shapefiles Can Be Created

Shapefiles can be created with the following four general methods:

- Export—Shapefiles can be created by exporting any data source to a shapefile using ARC/INFO®, PC ARC/INFO®, Spatial Database Engine™ (SDE™), ArcView® GIS, or *Business*MAP™ software.
- Digitize—Shapefiles can be created directly by digitizing shapes using ArcView GIS feature creation tools.
- Programming—Using Avenue™ (ArcView GIS), MapObjects™, ARC Macro Language (AML™) (ARC/INFO), or Simple Macro Language (SML™) (PC ARC/INFO) software, you can create shapefiles within your programs.
- Write directly to the shapefile specifications by creating a program.

ESRI White Paper

SDE, ARC/INFO, PC ARC/INFO, Data Automation Kit (DAK™), and ArcCAD® software provide shape-to-coverage data translators, and ARC/INFO also provides a coverage-to-shape translator. For exchange with other data formats, the shapefile specifications are published in this paper. Other data streams, such as those from global positioning system (GPS) receivers, can also be stored as shapefiles or X,Y event tables.

Shapefile Technical Description

Computer programs can be created to read or write shapefiles using the technical specification in this section.

Shape/CDB Readers: **The CDB standard globally provides a set of directory and filename conventions. The conventions do not limit filenames to the 8.3 naming convention.**

An ESRI shapefile consists of a main file, an index file, and a dBASE table. The main file is a direct access, variable-record-length file in which each record describes a shape with a list of its vertices. In the index file, each record contains the offset of the corresponding main file record from the beginning of the main file. The dBASE table contains feature attributes with one record per feature. The one-to-one relationship between geometry and attributes is based on record number. Attribute records in the dBASE file must be in the same order as records in the main file.

Naming Conventions

All file names adhere to the 8.3 naming convention. The main file, the index file, and the dBASE file have the same prefix. The prefix must start with an alphanumeric character (a–Z, 0–9), followed by zero or up to seven characters (a–Z, 0–9, _, -). The suffix for the main file is .shp. The suffix for the index file is .shx. The suffix for the dBASE table is .dbf. All letters in a file name are in lower case on operating systems with case sensitive file names.

Examples

- Main file: counties.shp
- Index file: counties.shx
- dBASE table: counties.dbf

Numeric Types

A shapefile stores integer and double-precision numbers. The remainder of this document will refer to the following types:

- Integer: Signed 32-bit integer (4 bytes)
- Double: Signed 64-bit IEEE double-precision floating point number (8 bytes)

Floating point numbers must be numeric values. Positive infinity, negative infinity, and Not-a-Number (NaN) values are not allowed in shapefiles. Nevertheless, shapefiles support the concept of "no data" values, but they are currently used only for measures. Any floating point number smaller than -10^{38} is considered by a shapefile reader to represent a "no data" value.

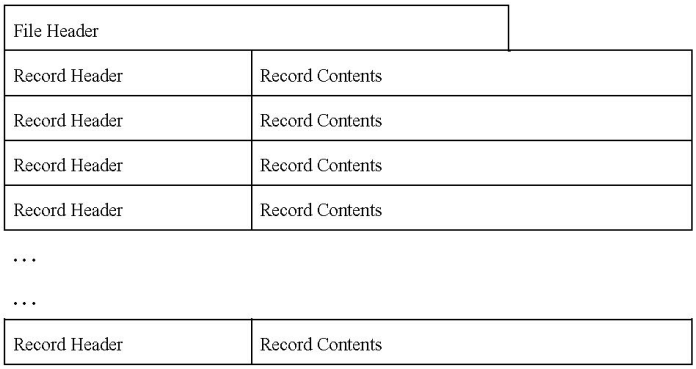
The first section below describes the general structure and organization of the shapefile. The second section describes the record contents for each type of shape supported in the shapefile.

Organization of the Main File

The main file (.shp) contains a fixed-length file header followed by variable-length records. Each variable-length record is made up of a fixed-length record header followed by variable-length record contents. **Figure 1** illustrates the main file organization.

J-7855

Figure 1
Organization of the Main File



Byte Order

All the contents in a shapefile can be divided into two categories:

- Data related
  - Main file record contents
  - Main file header's data description fields (Shape Type, Bounding Box, etc.)
- File management related
  - File and record lengths
  - Record offsets, and so on

The integers and double-precision integers that make up the data description fields in the file header (identified below) and record contents in the main file are in little endian (PC or Intel®) byte order. The integers and double-precision floating point numbers that make up the rest of the file and file management are in big endian (Sun® or Motorola®) byte order.

The Main File Header

The main file header is 100 bytes long. **Table 1** shows the fields in the file header with their byte position, value, type, and byte order. In the table, position is with respect to the start of the file.

Table 1
Description of the Main File Header

Position	Field	Value	Type	Byte Order
Byte 0	File Code	9994	Integer	Big
Byte 4	Unused	0	Integer	Big
Byte 8	Unused	0	Integer	Big
Byte 12	Unused	0	Integer	Big
Byte 16	Unused	0	Integer	Big
Byte 20	Unused	0	Integer	Big
Byte 24	File Length	File Length	Integer	Big
Byte 28	Version	1000	Integer	Little
Byte 32	Shape Type	Shape Type	Integer	Little
Byte 36	Bounding Box	Xmin	Double	Little
Byte 44	Bounding Box	Ymin	Double	Little
Byte 52	Bounding Box	Xmax	Double	Little
Byte 60	Bounding Box	Ymax	Double	Little
Byte 68*	Bounding Box	Zmin	Double	Little
Byte 76*	Bounding Box	Zmax	Double	Little
Byte 84*	Bounding Box	Mmin	Double	Little
Byte 92*	Bounding Box	Mmax	Double	Little

* Unused, with value 0.0, if not Measured or Z type

The value for file length is the total length of the file in 16-bit words (including the fifty 16-bit words that make up the header).

All the non-Null shapes in a shapefile are required to be of the same shape type. The values for shape type are as follows:

Value	Shape Type
0	Null Shape
1	Point
3	PolyLine
5	Polygon
8	MultiPoint
11	PointZ
13	PolyLineZ
15	PolygonZ
18	MultiPointZ
21	PointM
23	PolyLineM
25	PolygonM
28	MultiPointM
31	MultiPatch

Shape types not specified above (2, 4, 6, etc., and up to 33) are reserved for future use. Currently, shapefiles are restricted to contain the same type of shape as specified above. In the future, shapefiles may be allowed to contain more than one shape type. If mixed shape types are implemented, the shape type field in the header will flag the file as such.

The Bounding Box in the main file header stores the actual extent of the shapes in the file: the minimum bounding rectangle orthogonal to the X and Y (and potentially the M and Z) axes that contains all shapes. If the shapefile is empty (that is, has no records), the values for Xmin, Ymin, Xmax, and Ymax are unspecified. Mmin and Mmax can contain "no data" values (see Numeric Types on page 2) for shapefiles of measured shape types that contain no measures.

Record Headers

The header for each record stores the record number and content length for the record. Record headers have a fixed length of 8 bytes. **Table 2** shows the fields in the file header with their byte position, value, type, and byte order. In the table, position is with respect to the start of the record.

Table 2
Description of Main File Record Headers

Position	Field	Value	Type	Byte Order
Byte 0	Record Number	Record Number	Integer	Big
Byte 4	Content Length	Content Length	Integer	Big

Record numbers begin at 1.

The content length for a record is the length of the record contents section measured in 16-bit words. Each record, therefore, contributes (4 + content length) 16-bit words toward the total length of the file, as stored at Byte 24 in the file header.

Main File Record Contents

Shapefile record contents consist of a shape type followed by the geometric data for the shape. The length of the record contents depends on the number of parts and vertices in a shape. For each shape type, we first describe the shape and then its mapping to record contents on disk. In **Tables 3** through **16,** position is with respect to the start of the record contents.

Null Shapes

A shape type of 0 indicates a **null** shape, with no geometric data for the shape. Each feature type (point, line, polygon, etc.) supports nulls—it is valid to have points and null points in the same shapefile. Often null shapes are place holders; they are used during shapefile creation and are populated with geometric data soon after they are created.

Table 3
Null Shape Record Contents

Position	Field	Value	Type	Number	Byte Order
Byte 0	Shape Type	0	Integer	1	Little

Shape Types in X,Y Space

Point

A point consists of a pair of double-precision coordinates in the order X,Y.

```
Point
{
    Double    X    // X coordinate
    Double    Y    // Y coordinate
}
```

Table 4
Point Record Contents

Position	Field	Value	Type	Number	Byte Order
Byte 0	Shape Type	1	Integer	1	Little
Byte 4	X	X	Double	1	Little
Byte 12	Y	Y	Double	1	Little

MultiPoint

A MultiPoint represents a set of points, as follows:

```
MultiPoint
{
    Double[4]          Box         // Bounding Box
    Integer            NumPoints   // Number of Points
    Point[NumPoints]   Points      // The Points in the Set
}
```

The Bounding Box is stored in the order Xmin, Ymin, Xmax, Ymax.

J-7855

Table 5
MultiPoint Record Contents

Position	Field	Value	Type	Number	Byte Order
Byte 0	Shape Type	8	Integer	1	Little
Byte 4	Box	Box	Double	4	Little
Byte 36	NumPoints	NumPoints	Integer	1	Little
Byte 40	Points	Points	Point	NumPoints	Little

PolyLine

A PolyLine is an ordered set of vertices that consists of one or more parts. A part is a connected sequence of two or more points. Parts may or may not be connected to one another. Parts may or may not intersect one another.

Because this specification does not forbid consecutive points with identical coordinates, shapefile readers must handle such cases. On the other hand, the degenerate, zero length parts that might result are not allowed.

```
PolyLine
{
    Double[4]          Box         // Bounding Box
    Integer            NumParts    // Number of Parts
    Integer            NumPoints   // Total Number of Points
    Integer[NumParts]  Parts       // Index to First Point in Part
    Point[NumPoints]   Points      // Points for All Parts
}
```

The fields for a PolyLine are described in detail below:

Box — The Bounding Box for the PolyLine stored in the order Xmin, Ymin, Xmax, Ymax.

NumParts — The number of parts in the PolyLine.

NumPoints — The total number of points for all parts.

Parts — An array of length NumParts. Stores, for each PolyLine, the index of its first point in the points array. Array indexes are with respect to 0.

Points — An array of length NumPoints. The points for each part in the PolyLine are stored end to end. The points for Part 2 follow the points for Part 1, and so on. The parts array holds the array index of the starting point for each part. There is no delimiter in the points array between parts.

Table 6
PolyLine Record Contents

Position	Field	Value	Type	Number	Byte Order
Byte 0	Shape Type	3	Integer	1	Little
Byte 4	Box	Box	Double	4	Little
Byte 36	NumParts	NumParts	Integer	1	Little
Byte 40	NumPoints	NumPoints	Integer	1	Little
Byte 44	Parts	Parts	Integer	NumParts	Little
Byte X	Points	Points	Point	NumPoints	Little

Note: X = 44 + 4 * NumParts

Polygon

A polygon consists of one or more rings. A ring is a connected sequence of four or more points that form a closed, non-self-intersecting loop. A polygon may contain multiple outer rings. The order of vertices or orientation for a ring indicates which side of the ring is the interior of the polygon. The neighborhood to the right of an observer walking along the ring in vertex order is the neighborhood inside the polygon. Vertices of rings defining holes in polygons are in a counterclockwise direction. Vertices for a single, ringed polygon are, therefore, always in clockwise order. The rings of a polygon are referred to as its parts.

Because this specification does not forbid consecutive points with identical coordinates, shapefile readers must handle such cases. On the other hand, the degenerate, zero length or zero area parts that might result are not allowed.

The Polygon structure is identical to the PolyLine structure, as follows:

```
Polygon
{
    Double[4]          Box          // Bounding Box
    Integer            NumParts     // Number of Parts
    Integer            NumPoints    // Total Number of Points
    Integer[NumParts]  Parts        // Index to First Point in Part
    Point[NumPoints]   Points       // Points for All Parts
}
```

The fields for a polygon are described in detail below:

Box The Bounding Box for the polygon stored in the order Xmin, Ymin, Xmax, Ymax.

NumParts The number of rings in the polygon.

NumPoints The total number of points for all rings.

J-7855

Parts — An array of length NumParts. Stores, for each ring, the index of its first point in the points array. Array indexes are with respect to 0.

Points — An array of length NumPoints. The points for each ring in the polygon are stored end to end. The points for Ring 2 follow the points for Ring 1, and so on. The parts array holds the array index of the starting point for each ring. There is no delimiter in the points array between rings.

The instance diagram in **Figure 2** illustrates the representation of polygons. This figure shows a polygon with one hole and a total of eight vertices.

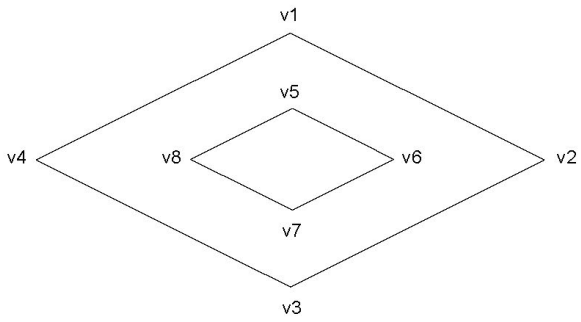
The following are important notes about Polygon shapes.

- The rings are closed (the first and last vertex of a ring MUST be the same).
- The order of rings in the points array is not significant.
- Polygons stored in a shapefile must be clean. A clean polygon is one that

1. Has no self-intersections. This means that a segment belonging to one ring may not intersect a segment belonging to another ring. The rings of a polygon can touch each other at vertices but not along segments. Colinear segments are considered intersecting.

2. Has the inside of the polygon on the "correct" side of the line that defines it. The neighborhood to the right of an observer walking along the ring in vertex order is the inside of the polygon. Vertices for a single, ringed polygon are, therefore, always in clockwise order. Rings defining holes in these polygons have a counterclockwise orientation. "Dirty" polygons occur when the rings that define holes in the polygon also go clockwise, which causes overlapping interiors.

Figure 2
An Example Polygon Instance



For this example, NumParts equals 2 and NumPoints equals 10. Note that the order of the points for the donut (hole) polygon are reversed below.

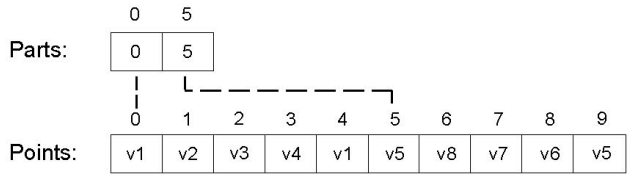


Table 7
Polygon Record Contents

Position	Field	Value	Type	Number	Byte Order
Byte 0	Shape Type	5	Integer	1	Little
Byte 4	Box	Box	Double	4	Little
Byte 36	NumParts	NumParts	Integer	1	Little
Byte 40	NumPoints	NumPoints	Integer	1	Little
Byte 44	Parts	Parts	Integer	NumParts	Little
Byte X	Points	Points	Point	NumPoints	Little

Note: X = 44 + 4 * NumParts

Measured Shape Types in X,Y Space

Shapes of this type have an additional coordinate—M. Note that "no data" value can be specified as a value for M (see Numeric Types on page 2).

PointM

A PointM consists of a pair of double-precision coordinates in the order X, Y, plus a measure M.

```
PointM
{
    Double      X       // X coordinate
    Double      Y       // Y coordinate
    Double      M       // Measure
}
```

J-7855

Table 8
PointM Record Contents

Position	Field	Value	Type	Number	Byte Order
Byte 0	Shape Type	21	Integer	1	Little
Byte 4	X	X	Double	1	Little
Byte 12	Y	Y	Double	1	Little
Byte 20	M	M	Double	1	Little

MultiPointM

A MultiPointM represents a set of PointMs, as follows

```
MultiPointM
{
    Double[4]          Box          // Bounding Box
    Integer            NumPoints    // Number of Points
    Point[NumPoints]   Points       // The Points in the Set
    Double[2]          M Range      // Bounding Measure Range
    Double[NumPoints]  M Array      // Measures
}
```

The fields for a MultiPointM are

Box — The Bounding Box for the MultiPointM stored in the order Xmin, Ymin, Xmax, Ymax

NumPoints — The number of Points

Points — An array of Points of length NumPoints

M Range — The minimum and maximum measures for the MultiPointM stored in the order Mmin, Mmax

M Array — An array of measures of length NumPoints

Table 9
MultiPointM Record Contents

Position	Field	Value	Type	Number	Byte Order
Byte 0	Shape Type	28	Integer	1	Little
Byte 4	Box	Box	Double	4	Little
Byte 36	NumPoints	NumPoints	Integer	1	Little
Byte 40	Points	Points	Point	NumPoints	Little
Byte X*	Mmin	Mmin	Double	1	Little
Byte X+8*	Mmax	Mmax	Double	1	Little
Byte X+16*	Marray	Marray	Double	NumPoints	Little

Note: X = 40 + (16 * NumPoints)
* optional

PolyLineM

A shapefile PolyLineM consists of one or more parts. A part is a connected sequence of two or more points. Parts may or may not be connected to one another. Parts may or may not intersect one another.

```
PolyLineM
{
    Double[4]          Box          // Bounding Box
    Integer            NumParts     // Number of Parts
    Integer            NumPoints    // Total Number of Points
    Integer[NumParts]  Parts        // Index to First Point in Part
    Point[NumPoints]   Points       // Points for All Parts
    Double[2]          M Range      // Bounding Measure Range
    Double[NumPoints]  M Array      // Measures for All Points
}
```

The fields for a PolyLineM are

Box — The Bounding Box for the PolyLineM stored in the order Xmin, Ymin, Xmax, Ymax.

NumParts — The number of parts in the PolyLineM.

NumPoints — The total number of points for all parts.

Parts — An array of length NumParts. Stores, for each part, the index of its first point in the points array. Array indexes are with respect to 0.

Points — An array of length NumPoints. The points for each part in the PolyLineM are stored end to end. The points for Part 2 follow the points for Part 1, and so on. The parts array holds the array index of the starting point for each part. There is no delimiter in the points array between parts.

J-7855

M Range — The minimum and maximum measures for the PolyLineM stored in the order Mmin, Mmax.

M Array — An array of length NumPoints. The measures for each part in the PolyLineM are stored end to end. The measures for Part 2 follow the measures for Part 1, and so on. The parts array holds the array index of the starting point for each part. There is no delimiter in the measure array between parts.

Table 10
PolyLineM Record Contents

Position	Field	Value	Type	Number	Byte Order
Byte 0	Shape Type	23	Integer	1	Little
Byte 4	Box	Box	Double	4	Little
Byte 36	NumParts	NumParts	Integer	1	Little
Byte 40	NumPoints	NumPoints	Integer	1	Little
Byte 44	Parts	Parts	Integer	NumParts	Little
Byte X	Points	Points	Point	NumPoints	Little
Byte Y*	Mmin	Mmin	Double	1	Little
Byte Y + 8*	Mmax	Mmax	Double	1	Little
Byte Y + 16*	Marray	Marray	Double	NumPoints	Little

Note: X = 44 + (4 * NumParts), Y = X + (16 * NumPoints)
* optional

PolygonM

A PolygonM consists of a number of rings. A ring is a closed, non-self-intersecting loop. Note that intersections are calculated in X,Y space, *not* in X,Y,M space. A PolygonM may contain multiple outer rings. The rings of a PolygonM are referred to as its parts.

The PolygonM structure is identical to the PolyLineM structure, as follows:

```
PolygonM
{
    Double[4]          Box          // Bounding Box
    Integer            NumParts     // Number of Parts
    Integer            NumPoints    // Total Number of Points
    Integer[NumParts]  Parts        // Index to First Point in Part
    Point[NumPoints]   Points       // Points for All Parts
    Double[2]          M Range      // Bounding Measure Range
    Double[NumPoints]  M Array      // Measures for All Points
}
```

The fields for a PolygonM are

Box	The Bounding Box for the PolygonM stored in the order Xmin, Ymin, Xmax, Ymax.
NumParts	The number of rings in the PolygonM.
NumPoints	The total number of points for all rings.
Parts	An array of length NumParts. Stores, for each ring, the index of its first point in the points array. Array indexes are with respect to 0.
Points	An array of length NumPoints. The points for each ring in the PolygonM are stored end to end. The points for Ring 2 follow the points for Ring 1, and so on. The parts array holds the array index of the starting point for each ring. There is no delimiter in the points array between rings.
M Range	The minimum and maximum measures for the PolygonM stored in the order Mmin, Mmax.
M Array	An array of length NumPoints. The measures for each ring in the PolygonM are stored end to end. The measures for Ring 2 follow the measures for Ring 1, and so on. The parts array holds the array index of the starting measure for each ring. There is no delimiter in the measure array between rings.

The following are important notes about PolygonM shapes.

- The rings are closed (the first and last vertex of a ring MUST be the same).
- The order of rings in the points array is not significant.

J-7855

Table 11
PolygonM Record Contents

Position	Field	Value	Type	Number	Byte Order
Byte 0	Shape Type	25	Integer	1	Little
Byte 4	Box	Box	Double	4	Little
Byte 36	NumParts	NumParts	Integer	1	Little
Byte 40	NumPoints	NumPoints	Integer	1	Little
Byte 44	Parts	Parts	Integer	NumParts	Little
Byte X	Points	Points	Point	NumPoints	Little
Byte Y*	Mmin	Mmin	Double	1	Little
Byte Y + 8*	Mmax	Mmax	Double	1	Little
Byte Y + 16*	Marray	Marray	Double	NumPoints	Little

Note: X = 44 + (4 * NumParts), Y = X + (16 * NumPoints)
* optional

Shape Types in X,Y,Z Space

Shapes of this type have an optional coordinate—M. Note that "no data" value can be specified as a value for M (see Numeric Types on page 2).

PointZ

A PointZ consists of a triplet of double-precision coordinates in the order X, Y, Z plus a measure.

```
PointZ
{
    Double    X    // X coordinate
    Double    Y    // Y coordinate
    Double    Z    // Z coordinate
    Double    M    // Measure
}
```

Table 12
PointZ Record Contents

Position	Field	Value	Type	Number	Byte Order
Byte 0	Shape Type	11	Integer	1	Little
Byte 4	X	X	Double	1	Little
Byte 12	Y	Y	Double	1	Little
Byte 20	Z	Z	Double	1	Little
Byte 28	Measure	M	Double	1	Little

MultiPointZ

A MultiPointZ represents a set of PointZs, as follows:

```
MultiPointZ
{
    Double[4]           Box         // Bounding Box
    Integer             NumPoints   // Number of Points
    Point[NumPoints]    Points      // The Points in the Set
    Double[2]           Z Range     // Bounding Z Range
    Double[NumPoints]   Z Array     // Z Values
    Double[2]           M Range     // Bounding Measure Range
    Double[NumPoints]   M Array     // Measures
}
```

The Bounding Box is stored in the order Xmin, Ymin, Xmax, Ymax.

The bounding Z Range is stored in the order Zmin, Zmax. Bounding M Range is stored in the order Mmin, Mmax.

Table 13
MultiPointZ Record Contents

Position	Field	Value	Type	Number	Byte Order
Byte 0	Shape Type	18	Integer	1	Little
Byte 4	Box	Box	Double	4	Little
Byte 36	NumPoints	NumPoints	Integer	1	Little
Byte 40	Points	Points	Point	NumPoints	Little
Byte X	Zmin	Zmin	Double	1	Little
Byte X+8	Zmax	Zmax	Double	1	Little
Byte X+16	Zarray	Zarray	Double	NumPoints	Little
Byte Y*	Mmin	Mmin	Double	1	Little
Byte Y+8*	Mmax	Mmax	Double	1	Little
Byte Y+16*	Marray	Marray	Double	NumPoints	Little

Note: X = 40 + (16 * NumPoints); Y = X + 16 + (8 * NumPoints)
* optional

J-7855

PolyLineZ

A PolyLineZ consists of one or more parts. A part is a connected sequence of two or more points. Parts may or may not be connected to one another. Parts may or may not intersect one another.

```
PolyLineZ
{
    Double[4]          Box          // Bounding Box
    Integer            NumParts     // Number of Parts
    Integer            NumPoints    // Total Number of Points
    Integer[NumParts]  Parts        // Index to First Point in Part
    Point[NumPoints]   Points       // Points for All Parts
    Double[2]          Z Range      // Bounding Z Range
    Double[NumPoints]  Z Array      // Z Values for All Points
    Double[2]          M Range      // Bounding Measure Range
    Double[NumPoints]  M Array      // Measures
}
```

The fields for a PolyLineZ are described in detail below:

Box — The Bounding Box for the PolyLineZ stored in the order Xmin, Ymin, Xmax, Ymax.

NumParts — The number of parts in the PolyLineZ.

NumPoints — The total number of points for all parts.

Parts — An array of length NumParts. Stores, for each part, the index of its first point in the points array. Array indexes are with respect to 0.

Points — An array of length NumPoints. The points for each part in the PolyLineZ are stored end to end. The points for Part 2 follow the points for Part 1, and so on. The parts array holds the array index of the starting point for each part. There is no delimiter in the points array between parts.

Z Range — The minimum and maximum Z values for the PolyLineZ stored in the order Zmin, Zmax.

Z Array — An array of length NumPoints. The Z values for each part in the PolyLineZ are stored end to end. The Z values for Part 2 follow the Z values for Part 1, and so on. The parts array holds the array index of the starting point for each part. There is no delimiter in the Z array between parts.

M Range — The minimum and maximum measures for the PolyLineZ stored in the order Mmin, Mmax.

M Array — An array of length NumPoints. The measures for each part in the PolyLineZ are stored end to end. The measures for Part 2 follow the measures for Part

1, and so on. The parts array holds the array index of the starting measure for each part. There is no delimiter in the measure array between parts.

Table 14
PolyLineZ Record Contents

Position	Field	Value	Type	Number	Byte Order
Byte 0	Shape Type	13	Integer	1	Little
Byte 4	Box	Box	Double	4	Little
Byte 36	NumParts	NumParts	Integer	1	Little
Byte 40	NumPoints	NumPoints	Integer	1	Little
Byte 44	Parts	Parts	Integer	NumParts	Little
Byte X	Points	Points	Point	NumPoints	Little
Byte Y	Zmin	Zmin	Double	1	Little
Byte Y + 8	Zmax	Zmax	Double	1	Little
Byte Y + 16	Zarray	Zarray	Double	NumPoints	Little
Byte Z*	Mmin	Mmin	Double	1	Little
Byte Z+8*	Mmax	Mmax	Double	1	Little
Byte Z+16*	Marray	Marray	Double	NumPoints	Little

Note: X = 44 + (4 * NumParts), Y = X + (16 * NumPoints), Z = Y + 16 + (8 * NumPoints)
* optional

PolygonZ

A PolygonZ consists of a number of rings. A ring is a closed, non-self-intersecting loop. A PolygonZ may contain multiple outer rings. The rings of a PolygonZ are referred to as its parts.

The PolygonZ structure is identical to the PolyLineZ structure, as follows:

```
PolygonZ
{
    Double[4]           Box         // Bounding Box
    Integer             NumParts    // Number of Parts
    Integer             NumPoints   // Total Number of Points
    Integer[NumParts]   Parts       // Index to First Point in Part
    Point[NumPoints]    Points      // Points for All Parts
    Double[2]           Z Range     // Bounding Z Range
    Double[NumPoints]   Z Array     // Z Values for All Points
    Double[2]           M Range     // Bounding Measure Range
    Double[NumPoints]   M Array     // Measures
}
```

J-7855

The fields for a PolygonZ are

Box	The Bounding Box for the PolygonZ stored in the order Xmin, Ymin, Xmax, Ymax.
NumParts	The number of rings in the PolygonZ.
NumPoints	The total number of points for all rings.
Parts	An array of length NumParts. Stores, for each ring, the index of its first point in the points array. Array indexes are with respect to 0.
Points	An array of length NumPoints. The points for each ring in the PolygonZ are stored end to end. The points for Ring 2 follow the points for Ring 1, and so on. The parts array holds the array index of the starting point for each ring. There is no delimiter in the points array between rings.
Z Range	The minimum and maximum Z values for the arc stored in the order Zmin, Zmax.
Z Array	An array of length NumPoints. The Z values for each ring in the PolygonZ are stored end to end. The Z values for Ring 2 follow the Z values for Ring 1, and so on. The parts array holds the array index of the starting Z value for each ring. There is no delimiter in the Z value array between rings.
M Range	The minimum and maximum measures for the PolygonZ stored in the order Mmin, Mmax.
M Array	An array of length NumPoints. The measures for each ring in the PolygonZ are stored end to end. The measures for Ring 2 follow the measures for Ring 1, and so on. The parts array holds the array index of the starting measure for each ring. There is no delimiter in the measure array between rings.

The following are important notes about PolygonZ shapes.

- The rings are closed (the first and last vertex of a ring MUST be the same).
- The order of rings in the points array is not significant.

Table 15
PolygonZ Record Contents

Position	Field	Value	Type	Number	Byte Order
Byte 0	Shape Type	15	Integer	1	Little
Byte 4	Box	Box	Double	4	Little
Byte 36	NumParts	NumParts	Integer	1	Little
Byte 40	NumPoints	NumPoints	Integer	1	Little
Byte 44	Parts	Parts	Integer	NumParts	Little
Byte X	Points	Points	Point	NumPoints	Little
Byte Y	Zmin	Zmin	Double	1	Little
Byte Y+8	Zmax	Zmax	Double	1	Little
Byte Y+16	Zarray	Zarray	Double	NumPoints	Little
Byte Z*	Mmin	Mmin	Double	1	Little
Byte Z+8*	Mmax	Mmax	Double	1	Little
Byte Z+16*	Marray	Marray	Double	NumPoints	Little

Note: X = 44 + (4 * NumParts), Y = X + (16 * NumPoints), Z = Y + 16 + (8 * NumPoints)
* optional

MultiPatch

A MultiPatch consists of a number of surface patches. Each surface patch describes a surface. The surface patches of a MultiPatch are referred to as its parts, and the type of part controls how the order of vertices of an MultiPatch part is interpreted. The parts of a MultiPatch can be of the following types:

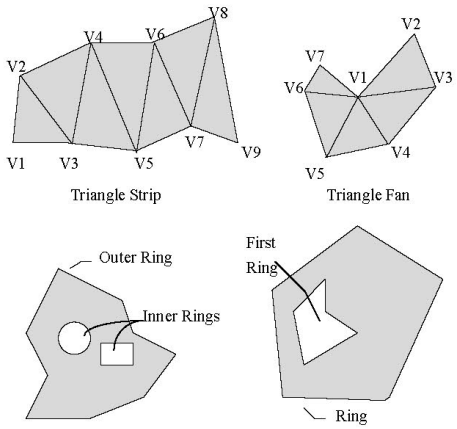
- **Triangle Strip** A linked strip of triangles, where every vertex (after the first two) completes a new triangle. A new triangle is always formed by connecting the new vertex with its two immediate predecessors.
- **Triangle Fan** A linked fan of triangles, where every vertex (after the first two) completes a new triangle. A new triangle is always formed by connecting the new vertex with its immediate predecessor and the first vertex of the part.
- **Outer Ring** The outer ring of a polygon.
- **Inner Ring** A hole of a polygon.
- **First Ring** The first ring of a polygon of an unspecified type.
- **Ring** A ring of a polygon of an unspecified type.

A single *Triangle Strip*, or *Triangle Fan*, represents a single surface patch. See **Figure 3** for examples of those part types.

J-7855

A sequence of parts that are rings can describe a polygonal surface patch with holes. The sequence typically consists of an *Outer Ring*, representing the outer boundary of the patch, followed by a number of *Inner Rings* representing holes. When the individual types of rings in a collection of rings representing a polygonal patch with holes are unknown, the sequence must start with *First Ring*, followed by a number of *Rings*. A sequence of *Rings* not preceded by an *First Ring* is treated as a sequence of *Outer Rings* without holes.

Figure 3
MultiPatch Part Examples



This figure shows examples of all types of MultiPatch parts.

The values used for encoding part type are as follows:

Value	Part Type
0	Triangle Strip
1	Triangle Fan
2	Outer Ring
3	Inner Ring
4	First Ring
5	Ring

```
MultiPatch
{
    Double[4]           Box         // Bounding Box
    Integer             NumParts    // Number of Parts
    Integer             NumPoints   // Total Number of Points
    Integer[NumParts]   Parts       // Index to First Point in Part
    Integer[NumParts]   PartTypes   // Part Type
    Point[NumPoints]    Points      // Points for All Parts
    Double[2]           Z Range     // Bounding Z Range
    Double[NumPoints]   Z Array     // Z Values for All Points
    Double[2]           M Range     // Bounding Measure Range
    Double[NumPoints]   M Array     // Measures
}
```

The fields for a MultiPatch are

Box	The Bounding Box for the MultiPatch stored in the order Xmin, Ymin, Xmax, Ymax.
NumParts	The number of parts in the MultiPatch.
NumPoints	The total number of points for all parts.
Parts	An array of length NumParts. Stores, for each part, the index of its first point in the points array. Array indexes are with respect to 0.
PartTypes	An array of length NumParts. Stores for each part its type.
Points	An array of length NumPoints. The points for each part in the MultiPatch are stored end to end. The points for Part 2 follow the points for Part 1, and so on. The parts array holds the array index of the starting point for each part. There is no delimiter in the points array between parts.
Z Range	The minimum and maximum Z values for the arc stored in the order Zmin, Zmax.
Z Array	An array of length NumPoints. The Z values for each part in the MultiPatch are stored end to end. The Z values for Part 2 follow the Z values for Part 1, and so on. The parts array holds the array index of the starting Z value for each part. There is no delimiter in the Z value array between parts.
M Range	The minimum and maximum measures for the MultiPatch stored in the order Mmin, Mmax.
M Array	An array of length NumPoints. The measures for each part in the MultiPatch are stored end to end. The measures for Part 2 follow the measures for Part 1, and so on. The parts array holds the array index of the

starting measure for each part. There is no delimiter in the measure array between parts.

The following are important notes about MultiPatch shapes.

- If a part is a ring, it must be closed (the first and last vertex of a ring MUST be the same).
- The order of parts that are rings in the points array is significant: *Inner Rings* must follow their *Outer Ring*; a sequence of *Rings* representing a single surface patch must start with a ring of the type *First Ring*.
- Parts can share common boundaries, but parts must not intersect and penetrate each other.

Table 16
MultiPatch Record Contents

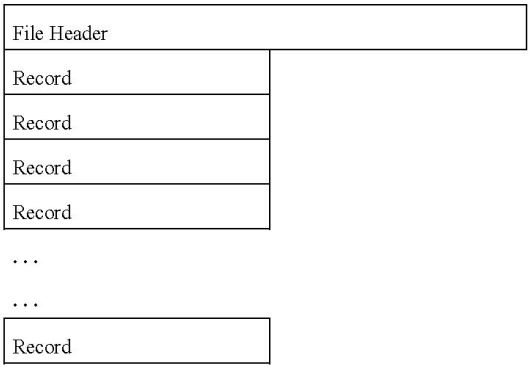
Position	Field	Value	Type	Number	Byte Order
Byte 0	Shape Type	31	Integer	1	Little
Byte 4	Box	Box	Double	4	Little
Byte 36	NumParts	NumParts	Integer	1	Little
Byte 40	NumPoints	NumPoints	Integer	1	Little
Byte 44	Parts	Parts	Integer	NumParts	Little
Byte W	PartTypes	PartTypes	Integer	NumParts	Little
Byte X	Points	Points	Point	NumPoints	Little
Byte Y	Zmin	Zmin	Double	1	Little
Byte Y+8	Zmax	Zmax	Double	1	Little
Byte Y+16	Zarray	Zarray	Double	NumPoints	Little
Byte Z*	Mmin	Mmin	Double	1	Little
Byte Z+8*	Mmax	Mmax	Double	1	Little
Byte Z+16*	Marray	Marray	Double	NumPoints	Little

Note: W = 44 + (4 * NumParts), X = W + (4 * NumParts), Y = X + (16 * NumPoints), Z = Y + 16 + (8 * NumPoints)
* optional

Organization of the Index File

The index file (.shx) contains a 100-byte header followed by 8-byte, fixed-length records. **Figure 4** illustrates the index file organization.

Figure 4
Organization of the Index File



The Index File Header

The index file header is identical in organization to the main file header described above. The file length stored in the index file header is the total length of the index file in 16-bit words (the fifty 16-bit words of the header plus 4 times the number of records).

Index Records

The *I'th* record in the index file stores the offset and content length for the *I'th* record in the main file. **Table 17** shows the fields in the file header with their byte position, value, type, and byte order. In the table, position is with respect to the start of the index file record.

Table 17
Description of Index Records

Position	Field	Value	Type	Byte Order
Byte 0	Offset	Offset	Integer	Big
Byte 4	Content Length	Content Length	Integer	Big

The offset of a record in the main file is the number of 16-bit words from the start of the main file to the first byte of the record header for the record. Thus, the offset for the first record in the main file is 50, given the 100-byte header.

The content length stored in the index record is the same as the value stored in the main file record header.

J-7855

Organization of the dBASE File

The dBASE file (.dbf) contains any desired feature attributes or attribute keys to which other tables can be joined. Its format is a standard DBF file used by many table-based applications in Windows™ and DOS. Any set of fields can be present in the table. There are three requirements, as follows:

- The file name must have the same prefix as the shape and index file. Its suffix must be .dbf. (See the example on page 2, in Naming Conventions.)
- The table must contain one record per shape feature.
- The record order must be the same as the order of shape features in the main (*.shp) file.
- The year value in the dBASE header must be the year since 1900.

For more information on the dBASE file format, visit the INPRISE Corp. Web site at www.inprise.com.

Glossary

Key terms are defined below that will help you understand the concepts discussed in this document.

ARC/INFO ARC/INFO software is designed for users who require a complete set of tools for processing and manipulating spatial data including digitizing, editing, coordinate management, network analysis, surface modeling, and grid cell based modeling. ARC/INFO operates on a large variety of workstations and minicomputers. Using open standards and client/server architecture, ARC/INFO can act as a GIS server for ArcView GIS clients.

ArcCAD ArcCAD software brings the functionality of ARC/INFO GIS software to the AutoCAD environment, providing comprehensive data management, spatial analysis, and display tools.

ARC Macro Language (AML) ARC Macro Language is a high-level, algorithmic language that provides full programming capabilities and a set of tools to tailor the user interface of your application.

ArcView GIS ArcView GIS software is a powerful, easy-to-use desktop GIS that gives you the power to visualize, explore, query, and analyze data spatially. ArcView GIS operates in Windows desktop environments as well as a large variety of workstations.

Avenue Avenue software is an object-oriented programming language and development environment created for use with ArcView GIS software. Avenue can be used to extend ArcView GIS software's basic capabilities and customize ArcView GIS for specific applications.

big endian byte order Left-to-right byte ordering of an integer word. This byte-ordering method is used on many UNIX systems including Sun, Hewlett–Packard®, IBM®, and Data General AViiON®.

Bounding Box A Bounding Box is a rectangle surrounding each shape (e.g., PolyLine) that is just large enough to contain the entire shape. It is defined as Xmin,Ymin, Xmax,Ymax.

***Business*MAP** *Business*MAP database mapping software for Windows allows you to create custom maps and represent information in two- or three-dimensional charts. *Business*MAP reads ESRI shapefiles and works with the leading contact managers, databases, and spreadsheets.

J-7855

coverage

1. A digital version of a map forming the basic unit of vector data storage in ARC/INFO software. A coverage stores geographic features as primary features (such as arcs, nodes, polygons, and label points) and secondary features (such as tics, map extent, links, and annotation). Associated feature attribute tables describe and store attributes of the geographic features.

2. A set of thematically associated data considered as a unit. A coverage usually represents a single theme such as soils, streams, roads, or land use.

Data Automation Kit (DAK)

Data Automation Kit (DAK) complements ArcView GIS and other desktop mapping software by providing high-quality digitizing and data editing, topology creation, data conversion, and map projection capabilities.

feature

A representation of a geographic feature that has both a spatial representation referred to as a "shape" and a set of attributes.

index file

An ArcView GIS shapefile index file is a file that allows direct access to records in the corresponding main file.

little endian byte order

Right-to-left byte ordering of an integer word. This byte-ordering method is used on many operating file systems including DEC OSF/1™, DEC OpenVMS™, MS–DOS®, and Windows NT™.

MapObjects

MapObjects is a collection of embeddable mapping and GIS components including an Active X Control (OCX) and programmable Active X Automation objects. Use MapObjects with a variety of standard Windows development environments to build mapping applications or add mapping components into existing applications.

MultiPoint

A single feature composed of a cluster of point locations and a single attribute record. The group of points represents the geographic feature.

NumPoints

The count of the number of x,y vertices contained in a shape.

PC ARC/INFO

PC ARC/INFO is a full-featured GIS for PC compatibles. Like ARC/INFO software, PC ARC/INFO is used by organizations around the world for automating, managing, and analyzing geographic information. Attributes describing geographic features are stored as tabular files in dBASE format.

PolyLine

An ordered set of x,y vertices representing a line or boundary.

ring

An ordered set of x,y vertices where the first vertex is the same location as the last vertex; a closed PolyLine or a polygon.

shapefile

An ArcView GIS data set used to represent a set of geographic features such as streets, hospital locations, trade areas, and ZIP Code boundaries. Shapefiles can represent point, line, or area features. Each feature in a shapefile represents a single geographic feature and its attributes.

Simple Macro Language (SML) SML is PC ARC/INFO software's Simple Macro Language—a set of commands that constitute a simple programming language for building macros with some of the features of a high-level programming language such as expression evaluation, handling of input and output, and directing program flow of control.

theme A user-defined set of geographic features. Data sources for themes in ArcView GIS include coverages, grids, images, and shapefiles. Theme properties include the data source name, attributes of interest, a data classification scheme, and drawing methodology.

topology The spatial relationships between connecting or adjacent coverage features (e.g., arcs, nodes, polygons, and points). For example, the topology of an arc includes its from- and to-nodes and its left and right polygons. Topological relationships are built from simple elements into complex elements: points (simplest elements) and arcs (sets of connected points) are used to represent more complex features such as areas (sets of connected arcs). Shapefiles do not explicitly record topology.

Coverages represent geographic features as topological line graphs. Topology can be useful for many GIS modeling operations that do not require coordinates. For example, to find an optimal path between two points requires a list of the arcs that connect to each other and the cost to traverse each arc in each direction. Coordinates are only needed for drawing the path after it is calculated.

vector A Cartesian (i.e., x,y) coordinate-based data structure commonly used to represent geographic features. Each feature is represented as one or more vertices. Attributes are associated with the feature. Other data structures include raster (which associates attributes with a grid cell) and triangulated irregular networks (TINs) for surface representation.

vertex One of a set of ordered x,y coordinates that constitutes a line.

For more than 25 years ESRI has been helping people manage and analyze geographic information. ESRI offers a framework for implementing GIS in any organization with a seamless link from personal GIS on the desktop to enterprisewide GIS client/server and data management systems. ESRI GIS solutions are flexible and can be customized to meet the needs of our users. ESRI is a full-service GIS company, ready to help you begin, grow, and build success with GIS.

Corporate

ESRI
380 New York Street
Redlands, California
92373-8100 USA
Telephone: 909-793-2853
Fax: 909-793-5953

For more information on ESRI software call ESRI at

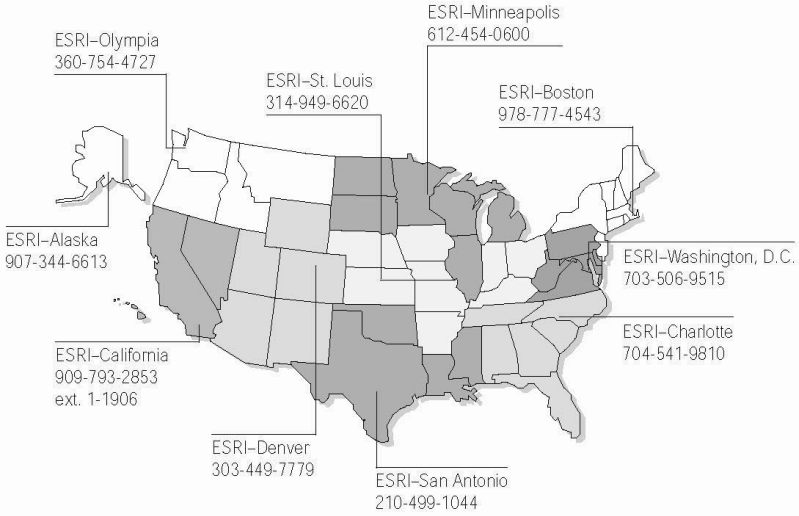
1-800-447-9778

(1-800-GIS-XPRT)

Send E-mail inquiries to
info@esri.com

Visit ESRI's Web page at
www.esri.com

Regional



International

Australia
61-89-242-1005

BeLux
32-2-460-7000

Canada
416-441-6035

France
33-1-46-23-6060

Germany
49-8166-677-0

Hong Kong
852-2-730-6883

India
91-11-620-3801

Italy
39-6-406-96-1

Nederland B.V.
31-10-217-0700

Poland
48-22-256-482

South Asia
65-735-8755

Spain
34-1-559-4347

Sweden
46-23-84090

Thailand
66-2-678-0707

United Kingdom
44-1-923-210450

Venezuela
58-2-285-1134

Outside the United States, contact your local ESRI distributor. For the number of your distributor, call ESRI at 909-793-2853, ext.1-1235

ESRI distributor or business partner address goes here

GSA
No. GS-35F-5D86H

Printed in USA

dBASE .DBF File Structure

by Borland Developer Support Staff

Technical Information Database

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

TI838D.txt dBASE .DBF File Structure
Category :Database Programming
Platform :All
Product :Delphi All

Description:
Sometimes it is necessary to delve into a dBASE table outside the control of the Borland Database Engine (BDE). For instance, if the .DBT file (that contains memo data) for a given table is irretrievably lost, the file will not be usable because the byte in the file header indicates that there should be a corresponding memo file. This necessitates toggling this byte to indicate no such accompanying memo file. Or, you may just want to write your own data access routine.

Below are the file structures for dBASE table files. Represented are the file structures as used for various versions of dBASE: dBASE III PLUS 1.1, dBASE IV 2.0, dBASE 5.0 for DOS, and dBASE 5.0 for Windows.

****
The data file header structure for dBASE III PLUS table file.
****

The table file header:
==========

Byte Contents Description
----- -------- ----------
0 1 byte Valid dBASE III PLUS table file (03h without a memo (.DBT file; 83h with a memo).

1-3 3 bytes Date of last update; in YYMMDD format.
4-7 32-bit number Number of records in the table.
8-9 16-bit number Number of bytes in the header.
10-11 16-bit number Number of bytes in the record.
12-14 3 bytes Reserved bytes.
15-27 13 bytes Reserved for dBASE III PLUS on a LAN.
28-31 4 bytes Reserved bytes.
32-n 32 bytes each Field descriptor array (the structure of this array is shown below)
n+1 1 byte 0Dh stored as the field terminator.

n above is the last byte in the field descriptor array. The size of the array depends on the number of fields in the table file.

Table Field Descriptor Bytes
==========

Byte Contents Description
----- -------- ----------
0-10 11 bytes Field name in ASCII (zero-filled).
11 1 byte Field type in ASCII (C, D, L, M, or N).
12-15 4 bytes Field data address (address is set in memory; not useful on disk).
16 1 byte Field length in binary.
17 1 byte Field decimal count in binary.
18-19 2 bytes Reserved for dBASE III PLUS on a LAN.
20 1 byte Work area ID.
21-22 2 bytes Reserved for dBASE III PLUS on a LAN.
23 1 byte SET FIELDS flag.
24-31 1 byte Reserved bytes.

Table Records
==========

The records follow the header in the table file. Data records are preceded by one byte, that is, a space (20h) if the record is not deleted, an asterisk (2Ah) if the record is deleted. Fields are packed into records without field separators or record terminators. The end of the file is mark by a single byte, with the end-of-file marker, an OEM code page character value of 26 (1Ah). You can input OEM code page data as indicated below.

Allowable Input for dBASE Data Types
==========

Data Type	Data Input
C (Character)	All OEM code page characters.
D (Date)	Numbers and a character to separate month, day, and year (stored internally as 8 digits in YYYYMMDD format).
N (Numeric)	- . 0 1 2 3 4 5 6 7 8 9
L (Logical)	? Y y N n T t F f (? when not initialized).
M (Memo)	All OEM code page characters (stored internally as 10 digits representing a .DBT block number).

Binary, Memo, and OLE Fields And .DBT Files
==========

Memo fields store data in .DBT files consisting of blocks numbered sequentially (0, 1, 2, and so on). The size of these blocks are internally set to 512 bytes. The first block in the .DBT file, block 0, is the .DBTfile header.

Memo field of each record in the .DBF file contains the number of the block (in OEM code page values) where the field's data actually begins. If a field contains no data, the .DBF file contains blanks (20h) rather than a number.

When data is changed in a field, the block numbers may also change and the number in the .DBF may be changed to reflect the new location.

This information is from the Using dBASE III PLUS manual, Appendix C.

****
The data file header structure for dBASE IV 2.0 table file.
****

File Structure:
==========

Byte	Contents	Meaning
0	1byte	Valid dBASE IV file; bits 0-2 indicate version number, bit 3 the presence of a dBASE IV memo file, bits 4-6 the presence of an SQL table, bit 7 the presence of any memo file (either dBASE III PLUS or dBASE IV).
1-3	3 bytes	Date of last update; formatted as YYMMDD.
4-7	32-bit number	Number of records in the file.
8-9	16-bit number	Number of bytes in the header.
10-11	16-bit number	Number of bytes in the record.
12-13	2 bytes	Reserved; fill with 0.
14	1 byte	Flag indicating incomplete transaction.
15	1 byte	Encryption flag.
16-27	12 bytes	Reserved for dBASE IV in a multi-user environment.
28	1 bytes	Production MDX file flag; 01H if there is an MDX, 00H if not.
29	1 byte	Language driver ID.
30-31	2 bytes	Reserved; fill with 0.
32-n*	32 bytes each	Field descriptor array (see below).
n + 1	1 byte	0DH as the field terminator.

* n is the last byte in the field descriptor array. The size of the array depends on the number of fields in the database file.

The field descriptor array:
==========

Byte	Contents	Meaning
0-10	11 bytes	Field name in ASCII (zero-filled).
11	1 byte	Field type in ASCII (C, D, F, L, M, or N).
12-15	4 bytes	Reserved.
16	1 byte	Field length in binary.
17	1 byte	Field decimal count in binary.
18-19	2 bytes	Reserved.
20	1 byte	Work area ID.
21-30	10 bytes	Reserved.
31	1 byte	Production MDX field flag; 01H if field has an index tag in the production MDX file, 00H if not.

Database records:
==========

The records follow the header in the database file. Data records are preceded by one byte; that is, a space (20H) if the record is not deleted, an asterisk (2AH) if the record is deleted. Fields are packed into records without field separators or record terminators. The end of the file is marked by a single byte, with the end-of-file marker an ASCII 26 (1AH) character.

Allowable Input for dBASE Data Types:
==========

Data	Type	Data Input
C	(Character)	All OEM code page characters.
D	(Date)	Numbers and a character to separate month, day, and year (stored internally as 8 digits in YYYYMMDD format).
F	(Floating point binary numeric)	- . 0 1 2 3 4 5 6 7 8 9
N	(Binary coded decimal numeric)	- . 0 1 2 3 4 5 6 7 8 9
L	(Logical)	? Y y N n T t F f (? when not initialized).
M	(Memo)	All OEM code page characters (stored internally as 10 digits representing a .DBT block number).

Memo Fields And .DBT Files
==========

Memo fields store data in .DBT files consisting of blocks numbered sequentially (0, 1, 2, and so on). SET BLOCKSIZE determines the size of each block. The first block in the .DBT file, block 0, is the .DBT file header.

Each memo field of each record in the .DBF file contains the number of the block (in OEM code page values) where the field's data actually begins. If a field contains no data, the .DBF file contains blanks (20h) rather than a number.

When data is changed in a field, the block numbers may also change and the number in the .DBF may be changed to reflect the new location.

This information is from the dBASE IV Language Reference manual, Appendix D.

****
The data file header structure for dBASE 5.0 for DOS table file.
****

The table file header:
==========

Byte	Contents	Description
0	1 byte	Valid dBASE for Windows table file; bits 0-2 indicate version number; bit 3 indicates presence of a dBASE IV or dBASE for Windows memo file; bits 4-6 indicate the presence of a dBASE IV SQL table; bit 7 indicates the presence of any .DBT memo file (either a dBASE III PLUS type or a dBASE IV or dBASE for Windows memo file).
1-3	3 bytes	Date of last update; in YYMMDD format.
4-7	32-bit number	Number of records in the table.
8-9	16-bit number	Number of bytes in the header.
10-11	16-bit number	Number of bytes in the record.
12-13	2 bytes	Reserved; filled with zeros.
14	1 byte	Flag indicating incomplete dBASE transaction.
15	1 byte	Encryption flag.
16-27	12 bytes	Reserved for multi-user processing.
28	1 byte	Production MDX flag; 01h stored in this byte if a production .MDX file exists for this table; 00h if no .MDX file exists.
29	1 byte	Language driver ID.
30-31	2 bytes	Reserved; filled with zeros.
32-n	32 bytes each	Field descriptor array (the structure of this array is shown below)
n+1	1 byte	0Dh stored as the field terminator.

n above is the last byte in the field descriptor array. The size of the array depends on the number of fields in the table file.

Table Field Descriptor Bytes
==========

Byte	Contents	Description
0-10	11 bytes	Field name in ASCII (zero-filled).
11	1 byte	Field type in ASCII (B, C, D, F, G, L, M, or N).
12-15	4 bytes	Reserved.
16	1 byte	Field length in binary.
17	1 byte	Field decimal count in binary.
18-19	2 bytes	Reserved.
20	1 byte	Work area ID.
21-30	10 bytes	Reserved.
31	1 byte	Production .MDX field flag; 01h if field has an index tag in the production .MDX file; 00h if the field is not indexed.

Table Records
==========

The records follow the header in the table file. Data records are preceded by one byte, that is, a space (20h) if the record is not deleted, an asterisk (2Ah) if the record is deleted. Fields are packed into records without field separators or record terminators. The end of the file is marked by a single byte, with the end-of-file marker, an OEM code page character value of 26 (1Ah). You can input OEM code page data as indicated below.

Allowable Input for dBASE Data Types
==========

Data Type	Data Input
C (Character)	All OEM code page characters.
D (Date)	Numbers and a character to separate month, day, and year (stored internally as 8 digits in YYYYMMDD format).
F (Floating point binary numeric)	- . 0 1 2 3 4 5 6 7 8 9
N (Numeric)	- . 0 1 2 3 4 5 6 7 8 9
L (Logical)	? Y y N n T t F f (? when not initialized).
M (Memo)	All OEM code page characters (stored internally as 10 digits representing a .DBT block number).

Memo Fields And .DBT Files
==========

Memo fields store data in .DBT files consisting of blocks numbered sequentially (0, 1, 2, and so on). SET BLOCKSIZE determines the size of each block. The first block in the .DBT file, block 0, is the .DBT file header.

Each memo field of each record in the .DBF file contains the number of the block (in OEM code page values) where the field's data actually begins. If a field contains no data, the .DBF file contains blanks (20h) rather than a number.

When data is changed in a field, the block numbers may also change and the number in the .DBF may be changed to reflect the new location.

Unlike dBASE III PLUS, if you delete text in a memo field, dBASE 5.0 for DOS may reuse the space from the deleted text when you input new text. dBASE III PLUS always appends new text to the end of the .DBT file. In dBASE III PLUS, the .DBT file size grows whenever new text is added, even if other text in the file is deleted.

This information is from the dBASE for DOS Language Reference manual, Appendix C.

****
The data file header structure for dBASE 5.0 for Windows table file.
****

The table file header:
==========

Byte	Contents	Description
0	1 byte	Valid dBASE for Windows table file; bits 0-2 indicate version number; bit 3 indicates presence of a dBASE IV or dBASE for Windows memo file; bits 4-6 indicate the presence of a dBASE IV SQL table; bit 7 indicates the presence of any .DBT memo file (either a dBASE III PLUS type or a dBASE IV or dBASE for Windows memo file).
1-3	3 bytes	Date of last update; in YYMMDD format.
4-7	32-bit number	Number of records in the table.
8-9	16-bit number	Number of bytes in the header.
10-11	16-bit number	Number of bytes in the record.
12-13	2 bytes	Reserved; filled with zeros.
14	1 byte	Flag indicating incomplete dBASE IV transaction.
15	1 byte	dBASE IV encryption flag.
16-27	12 bytes	Reserved for multi-user processing.
28	1 byte	Production MDX flag; 01h stored in this byte if a production .MDX file exists for this table; 00h if no .MDX file exists.
29	1 byte	Language driver ID.
30-31	2 bytes	Reserved; filled with zeros.
32-n	32 bytes each	Field descriptor array (the structure of this array is shown below)
n+1	1 byte	0Dh stored as the field terminator.

n above is the last byte in the field descriptor array. The size of the array depends on the number of fields in the table file.

Table Field Descriptor Bytes
==========

Byte	Contents	Description
0-10	11 bytes	Field name in ASCII (zero-filled).
11	1 byte	Field type in ASCII (B, C, D, F, G, L, M, or N).
12-15	4 bytes	Reserved.
16	1 byte	Field length in binary.
17	1 byte	Field decimal count in binary.
18-19	2 bytes	Reserved.
20	1 byte	Work area ID.
21-30	10 bytes	Reserved.
31	1 byte	Production .MDX field flag; 01h if field has an index tag in the production .MDX file; 00h if the field is not indexed.

Table Records
==========

The records follow the header in the table file. Data records are preceded by one byte, that is, a space (20h) if the record is not deleted, an asterisk (2Ah) if the record is deleted. Fields are packed into records without field separators or record terminators. The end of the file is marked by a single byte, with the end-of-file marker, an OEM code page character value of 26 (1Ah). You can input OEM code page data as indicated below.

Allowable Input for dBASE Data Types
==========

Data Type	Data Input
B (Binary)	All OEM code page characters (stored internally as 10 digits representing a .DBT block number).
C (Character)	All OEM code page characters.
D (Date)	Numbers and a character to separate month, day, and year (stored internally as 8 digits in YYYYMMDD format).
G (General	All OEM code page characters (stored internally as 10 digits or OLE) representing a .DBT block number).
N (Numeric)	- . 0 1 2 3 4 5 6 7 8 9
L (Logical)	? Y y N n T t F f (? when not initialized).
M (Memo)	All OEM code page characters (stored internally as 10 digits representing a .DBT block number).

Binary, Memo, and OLE Fields And .DBT Files
==========

Binary, memo, and OLE fields store data in .DBT files consisting of blocks numbered sequentially (0, 1, 2, and so on). SET BLOCKSIZE determines the size of each block. The first block in the .DBT file, block 0, is the .DBT file header.

Each binary, memo, or OLE field of each record in the .DBF file contains the number of the block (in OEM code page values) where the field's data actually begins. If a field contains no data, the .DBF file contains blanks (20h) rather than a number.

When data is changed in a field, the block numbers may also change and the number in the .DBF may be changed to reflect the new location.

Unlike dBASE III PLUS, if you delete text in a memo field (or binary and OLE fields), dBASE for Windows (unlike dBASE IV) may reuse the space from the deleted text when you input new text. dBASE III PLUS always appends new text to the end of the .DBT file. In dBASE III PLUS, the .DBT file size grows whenever new text is added, even if other text in the file is deleted.

This information is from the dBASE for Windows Language Reference manual, Appendix C.

Appendix E

E. CDB Light Names and Hierarchy

CDB Lights are listed below; the XML file are also delivered with the Specification in:

```
\CDB\Metadata\Lights.xml
```

	Light Hierarchy	v3.0 Light Code	v3.1 Light Code	Light Code	Description	Intensity (normalized)	Color (normalized RGB)	Directionality (type)	Width_Hor (degrees)	Width_Vert (degrees)	Intensity_Res (normalized)	Frequency (Hz)	Duty_Cycle (normalized)
1	**Light**	0	0	0	All purpose generic Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
2	**Platform**	1	1	1	Generic Platform Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
3	**Air**	2	2	2	Generic Aircraft Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
4	**Aircraft_Helos**	3	3	3	Generic Light for Aircraft and Helicopters	0.6	1 | 1 | 1	Omni	---	---	---	---	---
5	**Anti-collision**	4	4	4	Generic Anti collision Light - normally red flashing	0.6	1 | 0 | 0	Omni	---	---	---	---	---
6	Bottom_Light	5	5	5	Anti-collision found on bottom of the fuselage	0.6	1 | 0 | 0	Omni	---	---	---	---	---
7	NVG_Bottom_Light	6	6	6	Anti-collision found on bottom of the fuselage in NVG Mode	0.6	1 | 0 | 0	Omni	---	---	---	---	---
8	Top_Light	7	7	7	Anti-collision found on Top of the fuselage	0.6	1 | 0 | 0	Omni	---	---	---	---	---
9	NVG_Top_Light	8	8	8	Anti-collision found on Top of the fuselage in NVG Mode	0.6	1 | 0 | 0	Omni	---	---	---	---	---
10	High_Intensity		501	501	High Intensity Anti-collision Light	0.95	1 | 0 | 0	Omni	---	---	---	0.7	0.25
11	Formation_Light	9	9	9	Florescent formation strip Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
12	Flood_Light	10	10	10	White flood Lights used to illuminate the ground or part of the aircraft	0.8	1 | 1 | 1	Omni	---	---	---	---	---
13	Head_Light	11	11	11	Head Light used to allow pilots to see ahead	0.6	1 | 1 | 1	Omni	---	---	---	---	---
14	**Identification_Strobe**	12	12	12	Generic Strobe Lights used in fLight to indicate position	0.6	1 | 1 | 1	Omni	---	---	---	1	0.05
15	Red_Light	13	13	13	Red identification strobe Light	0.6	1 | 0 | 0	Omni	---	---	---	1	0.05
16	White_Light	14	14	14	White identification strobe Light	0.6	1 | 1 | 1	Omni	---	---	---	1	0.05
17	IR_Light	15	15	15	Infrared Lights used to indicate position using infrared instruments	0.6	1 | 1 | 1	Omni	---	---	---	---	---
18	Landing_Light	16	16	16	White Lights used on Landing approach	0.9	1 | 1 | 1	Dir	60	60	---	---	---
19	**Navigation**	17	17	17	Generic Lights used in fLight to indicate position	0.6	1 | 1 | 1	Omni	---	---	---	---	---
20	Red_Light	18	18	18	Red Navigation Light found on the left wing	0.6	1 | 0 | 0	Omni	---	---	---	---	---
21	Flashing_Red_Light		502	502	Flashing Red Navigation Light found on the left wing	0.6	1 | 0 | 0	Omni	---	---	---	1	0.5
22	Green_Light	19	19	19	Green Navigation Light found on the right wing	0.6	0 | 1 | 0	Omni	---	---	---	---	---
23	Flashing_Green_Light		503	503	Flashing Green Navigation Light found on the right wing	0.6	0 | 1 | 0	Omni	---	---	---	1	0.5
24	White_Light	20	20	20	White Navigation Light found on the tail wing	0.6	1 | 1 | 1	Omni	---	---	---	---	---
25	Flashing_White_Light		504	504	Flashing White Navigation Light found on the tail wing	0.6	1 | 1 | 1	Omni	---	---	---	1	0.5
26	NVG_Light	21	21	21	Navigation Light used in NVG Mode	0.6	1 | 1 | 1	Omni	---	---	---	---	---
27	Tail_Light	22	22	22	White Tail Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
28	Tail_Flood	23	23	23	Flood Light used to illuminated the tail, showing off the logo or markings	0.8	1 | 1 | 1	Omni	---	---	---	---	---
29	Taxi_Light	24	24	24	White Lights used when Aircrafts taxi on the ground	0.8	1 | 1 | 1	Dir	40	40	---	---	---
30	**Wingtip_Obstruction**	25	25	25	Generic Wintip obstruction Light	0.6	1 | 0 | 0	Omni	---	---	---	0.5	0.33
31	Red_Light	26	26	26	Red Obstruction Light found on left wing	0.6	1 | 0 | 0	Omni	---	---	---	0.5	0.33
32	Green_Light	27	27	27	Green Obstruction Light found on right wing	0.6	0 | 1 | 0	Omni	---	---	---	0.5	0.33
33	**Civil**	28	28	28	Generic Civil aircraft Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
34	Business	29	29	29		0.6	1 | 1 | 1	Omni	---	---	---	---	---
35	Regional	30	30	30		0.6	1 | 1 | 1	Omni	---	---	---	---	---
36	Transport	31	31	31		0.6	1 | 1 | 1	Omni	---	---	---	---	---
37	Widebody	32	32	32		0.6	1 | 1 | 1	Omni	---	---	---	---	---
38	**Military**	33	33	33	Generic Military aircrafts Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
39	Cargo_Light	34	34	34	Cargo Light	0.6	1 | 1 | 1	Dir	180	60	---	---	---
40	IR	35	35	35	Infrared Cargo Light	0.6	1 | 1 | 1	Dir	180	60	---	---	---
41	Refueling_Light	36	36	36	Refueling Light	0.6	1 | 1 | 1	Dir	60	60	---	---	---
42	Search_Light	37	37	37	Search Light	0.9	1 | 1 | 1	Dir	10	10	---	---	---
43	NVG_Light	38	38	38	Search Light used in NVG Mode	0.9	1 | 1 | 1	Dir	10	10	---	---	---
44	ASW_Patrol	39	39	39	GenericASW Patrol Aircraft Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
45	Bomber	40	40	40	Generic Bomber Aircraft Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
46	**Cargo_Tanker**	41	41	41	Generic Cargo Tanker Aircraft Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
47	**Pod_Light**	425	466	466	Generic Pod Lights on Cargo Tanker	0.6	1 | 1 | 1	Omni	---	---	---	---	---
48	**Starboard**	426	467	467	Generic Starboard Pod Lights on Cargo Tanker	0.6	1 | 1 | 1	Omni	---	---	---	---	---
49	Green_Light	427	468	468	Green Light Aft of Starboard pod	0.6	0 | 1 | 0	Omni	---	---	---	---	---
50	Red_Light	428	469	469	Red Light Aft of Starboard pod	0.6	1 | 0 | 0	Omni	---	---	---	---	---

	Light Hierarchy	v3.0 Light Code	v3.1 Light Code	Light Code	Description	Intensity (normalized)	Color (normalized RGB)	Directionality (type)	Width_Hor (degrees)	Width_Vert (degrees)	Intensity_Res (normalized)	Frequency (Hz)	Duty_Cycle (normalized)
51	Yellow_Light	429	470	470	Yellow Light Aft of Starboard pod	0.6	1 | 1 | 0	Omni	---	---	---	---	---
52	**Port**	430	471	471	Generic Port Pod Lights on Cargo Tanker	0.6	1 | 1 | 1	Omni	---	---	---	---	---
53	Green_Light	431	472	472	Green Light Aft of Port pod	0.6	0 | 1 | 0	Omni	---	---	---	---	---
54	Red_Light	432	473	473	Red Light Aft of Port pod	0.6	1 | 0 | 0	Omni	---	---	---	---	---
55	Yellow_Light	433	474	474	Yellow Light Aft of Port pod	0.6	1 | 1 | 0	Omni	---	---	---	---	---
56	**Aldus_Light**	434	475	475	Generic Aldus Lights on Cargo Tanker	0.6	1 | 1 | 1	Omni	---	---	---	---	---
57	**Starboard**	435	476	476	Generic Starboard Aldus Lights on Cargo Tanker	0.6	1 | 1 | 1	Omni	---	---	---	---	---
58	Amber_Light	436	477	477	Amber aldus Light at Starboard Aft door	0.6	1 | 0.6 | 0	Omni	---	---	---	---	---
59	Green_Light	437	478	478	Green aldus Light at Starboard Aft door	0.6	0 | 1 | 0	Omni	---	---	---	---	---
60	Red_Light	438	479	479	Red aldus Light at Starboard Aft door	0.6	1 | 0 | 0	Omni	---	---	---	---	---
61	Yellow_Light	439	480	480	Yellow aldus Light at Starboard Aft door	0.6	1 | 1 | 0	Omni	---	---	---	---	---
62	**Port**	440	481	481	Generic Port Aldus Lights on Cargo Tanker	0.6	1 | 1 | 1	Omni	---	---	---	---	---
63	Amber_Light	441	482	482	Amber aldus Light at Port Aft door	0.6	1 | 0.6 | 0	Omni	---	---	---	---	---
64	Green_Light	442	483	483	Green aldus Light at Port Aft door	0.6	0 | 1 | 0	Omni	---	---	---	---	---
65	Red_Light	443	484	484	Red aldus Light at Port Aft door	0.6	1 | 0 | 0	Omni	---	---	---	---	---
66	Yellow_Light	444	485	485	Yellow aldus Light at Port Aft door	0.6	1 | 1 | 0	Omni	---	---	---	---	---
67	Fighter	42	42	42	Generic Fighter Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
68	**Helicopter**	43	43	43	Specific Military Helicopter Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
69	Slung_Load_Light	44	44	44	Light used to Illuminate objects carried on a slung load	0.7	1 | 1 | 1	Omni	---	---	---	---	---
70	Attack	45	45	45	Generic Attack Helicopter Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
71	Cargo	46	46	46	Generic Cargo Helicopter Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
72	**Special_Ops**	47	47	47	Generic Special-Ops Helicopter Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
73	**MH47-E**	445	486	486	Generic Special-OpsMH-47-E Helicopter Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
74	Porch_Light	446	487	487	Lower White on bottom of Aft pylon near exhaust	0.6	1 | 1 | 1	Omni	---	---	---	---	---
75	Utility	48	48	48	Generic Utility Helicopter Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
76	Tanker	49	49	49	Generic Tanker Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
77	Unmanned	50	50	50	Generic Military Unmanned Aerial Vehicle (UAV) Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
78	**Navigation**		494	494	Generic Nav Lights on UAVs to indicate position	0.6	1 | 1 | 1	Omni	---	---	---	---	---
79	Red_Light		495	495	Red navigation Light found on left wing	0.6	1 | 0 | 0	Omni	---	---	---	---	---
80	Green_Light		496	496	Green navigation Light found on right wing	0.6	0 | 1 | 0	Omni	---	---	---	---	---
81	White_Light		497	497	White navigation Light usually on the tail	0.6	1 | 1 | 1	Omni	---	---	---	---	---
82	**Position**		498	498	Generic Position Lights on UAVs to indicate position	0.6	1 | 1 | 1	Omni	---	---	---	---	---
83	Orange_Light		499	499	Orange position Light	0.6	1 | 0.5 | 0	Omni	---	---	---	---	---
84	White_Light		500	500	White position Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
85	**Land**	51	51	51	Generic Land Vehicle Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
86	Backup_Light	52	52	52	White Lights that indication a vehicle backing up	0.3	1 | 1 | 1	Omni	---	---	---	---	---
87	Blinking_Emergency_Light	53	53	53	Yellow flashing emergency Lights (i.e. 4-way flashing indicator Light	0.4	1 | 1 | 0	Omni	---	---	---	0.5	0.5
88	Blinking_Turn_Light	54	54	54	Yellow blinking turning indicator Light	0.4	1 | 1 | 0	Omni	---	---	---	0.5	0.5
89	Brake_Light	55	55	55	Red Lights when brakes are applied	0.4	1 | 0 | 0	Omni	---	---	---	---	---
90	**Headlight**	56	56	56	Generic Headlight on a Land Vehicle that allow a driver to see ahead	0.5	1 | 1 | 1	Omni	---	---	---	---	---
91	Low_Beam_Light	57	57	57	Low beam head Lights	0.5	1 | 1 | 1	Omni	---	---	---	---	---
92	High_Beam_Light	58	58	58	High beam head Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
93	Perimeter_Amber_Light	59	59	59	Perimeter Lights	0.4	1 | 0.6 | 0	Omni	---	---	---	---	---
94	Strobing_Blue_Light	60	60	60	Blue strobe (Flashing)	0.5	0 | 0 | 1	Omni	---	---	---	1	0.05
95	Strobing_Red_Light	61	61	61	Red strobe (Flashing)	0.5	1 | 0 | 0	Omni	---	---	---	1	0.05
96	Strobing_White_Light	62	62	62	White Strobe (Flashing)	0.5	1 | 1 | 1	Omni	---	---	---	1	0.05
97	Strobing_Yellow_Light	63	63	63	Yellow Strobe (Flashing)	0.5	1 | 1 | 0	Omni	---	---	---	1	0.05
98	Tail_Light	64	64	64	Red tail Lights	0.4	1 | 0 | 0	Omni	---	---	---	---	---
99	Turn_Signal_Light	65	65	65	Yellow turning indicator Light	0.4	1 | 1 | 0	Omni	---	---	---	---	---
100	Car	66	66	66	Generic Car Lights	0.4	1 | 1 | 1	Omni	---	---	---	---	---

	Light Hierarchy	v3.0 Light Code	v3.1 Light Code	Light Code	Description	Intensity (normalized)	Color (normalized RGB)	Directionality (type)	Width_Hor (degrees)	Width_Vert (degrees)	Intensity_Res (normalized)	Frequency (Hz)	Duty_Cycle (normalized)
101	Transport_Van	67	67	67	Generic Transport Lights	0.4	1 | 1 | 1	Omni	---	---	---	---	---
102	Truck	68	68	68	Generic Truck Lights	0.4	1 | 1 | 1	Omni	---	---	---	---	---
103	Ambulance	69	69	69	Generic Ambulance Lights	0.4	1 | 1 | 1	Omni	---	---	---	---	---
104	Firetruck	70	70	70	Generic Fire truck Lights	0.4	1 | 1 | 1	Omni	---	---	---	---	---
105	**Train**	71	71	71	Generic Train Lights	0.4	1 | 1 | 1	Omni	---	---	---	---	---
106	Caboose_Rear_Light	72	72	72	Caboose red Light at rear of a train	0.4	1 | 0 | 0	Omni	---	---	---	---	---
107	Engine_Head_Light	73	73	73	Train engine white head Light	0.7	1 | 1 | 1	Omni	---	---	---	---	---
108	Tank	74	74	74	Generic Tank Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
109	**Surface**	75	75	75	Generic Surface Vehicle Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
110	**Buoy**	76	76	76	Generic Buoy Lights found on a Surface Vehicle	0.6	1 | 1 | 1	Omni	---	---	---	0.22	0.8
111	Green_Light	77	77	77	Green Buoy Light	0.6	0 | 1 | 0	Omni	---	---	---	0.22	0.8
112	Red_Light	78	78	78	Red Buoy Light	0.6	1 | 0 | 0	Omni	---	---	---	0.22	0.8
113	White_Light	79	79	79	White Buoy Light	0.6	1 | 1 | 1	Omni	---	---	---	0.22	0.8
114	Yellow_Light	80	80	80	Yellow Buoy Light	0.6	1 | 1 | 0	Omni	---	---	---	0.22	0.8
115	**Marine_entry**	81	81	81	Generic Marine Entry Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
116	Green_Light	82	82	82	Green Light	0.6	0 | 1 | 0	Omni	---	---	---	---	---
117	Red_Light	83	83	83	Red Light	0.6	1 | 0 | 0	Omni	---	---	---	---	---
118	**Ship_Boat**	84	84	84	Generic Ship/boat Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
119	**Navigation**	85	85	85	Generic Navigation Lights on a Ship Boat	0.6	1 | 1 | 1	Omni	---	---	---	---	---
120	**Directional**	86	86	86	Generic Directional navigation Lights	0.6	1 | 1 | 1	Dir	180	180	---	---	---
121	Green_Light	87	87	87	Green directional navigation Light	0.6	0 | 1 | 0	Dir	180	180	---	---	---
122	Red_Light	88	88	88	Red directional navigation Light	0.6	1 | 0 | 0	Dir	180	180	---	---	---
123	White_Light	89	89	89	White directional navigation Light	0.6	1 | 1 | 1	Dir	180	180	---	---	---
124	**Omnidirectional**	90	90	90	Generic Omnidirectional navigation Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
125	Green_Light	91	91	91	Green omnidirectional navigation Light	0.6	0 | 1 | 0	Omni	---	---	---	---	---
126	Red_Light	92	92	92	Red omnidirectional navigation Light	0.6	1 | 0 | 0	Omni	---	---	---	---	---
127	White_Light	93	93	93	White omnidirectional navigation Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
128	Search_Light	94	94	94	Search Light	0.9	1 | 1 | 1	Dir	10	10	---	---	---
129	NVG_Light	95	95	95	Search Light used in NVG mode	0.9	1 | 1 | 1	Dir	10	10	---	---	---
130	**Civil**	96	96	96	Generic Ship/boat civil Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
131	Anchor_Light	97	97	97	Lights used to illuminate the anchor	0.6	1 | 1 | 1	Dir	150	120	---	---	---
132	Flood_Light	98	98	98	Lights used to illuminate the ground or the deck	0.6	1 | 1 | 1	Dir	30	30	---	---	---
133	**Mast**	99	99	99	Generic Lights found on a mast of the civilian ship	0.6	1 | 1 | 1	Dir	225	120	---	---	---
134	Amber_Light	100	100	100	Amber Mast Light	0.6	1 | 0.6 | 0	Dir	225	120	---	---	---
135	Green_Light	101	101	101	Green Mast Light	0.6	0 | 1 | 0	Dir	225	120	---	---	---
136	Red_Light	102	102	102	Red Mast Light	0.6	1 | 0 | 0	Dir	225	120	---	---	---
137	White_Light	103	103	103	White Mast Light	0.6	1 | 1 | 1	Dir	225	120	---	---	---
138	Cargo	104	104	104	Generic Cargo Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
139	Container_Vessel	105	105	105	Generic Container Vessel Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
140	Ferry	106	106	106	Generic Ferry Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
141	Fishing_Vessel	107	107	107	Generic Fishing Vessel Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
142	Ocean_Liner	108	108	108	Generic Ocean Liner specific Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
143	Oil_Rig	109	109	109	Generic Oil Rig Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
144	Tanker	110	110	110	generic Tanker Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
145	**Military**	111	111	111	Generic Military Ship/Boat Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
146	Flare_Light	112	112	112	Light effect from a Flare	0.8	1 | 1 | 1	Omni	---	---	---	---	---
147	Flood_Light	113	113	113	Lights used to illuminate the ground or the deck	0.6	1 | 1 | 1	Dir	30	30	---	---	---
148	**Mast**	114	114	114	Generic Lights found on a mast of the military ship	0.6	1 | 1 | 1	Dir	225	120	---	---	---
149	Amber_Light	115	115	115	Amber Mast Light	0.6	1 | 0.6 | 0	Dir	225	120	---	---	---
150	Green_Light	116	116	116	Green Mast Light	0.6	0 | 1 | 0	Dir	225	120	---	---	---

	Light Hierarchy	v3.0 Light Code	v3.1 Light Code	Light Code	Description	Intensity (normalized)	Color (normalized RGB)	Directionality (type)	Width_Hor (degrees)	Width_Vert (degrees)	Intensity_Res (normalized)	Frequency (Hz)	Duty_Cycle (normalized)
151	Red_Light	117	117	117	Red Mast Light	0.6	1 | 0 | 0	Dir	225	120	---	---	---
152	White_Light	118	118	118	White Mast Light	0.6	1 | 1 | 1	Dir	225	120	---	---	---
153	**HIRF**	447	447	447	Generic High-Intensity Radiated Fields Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
154	Amber_Light	448	448	448	Amber HIRF Light	0.6	1 | 0.6 | 0	Omni	---	---	---	---	---
155	Red_Light	449	449	449	Red HIRF Light	0.6	1 | 0 | 0	Omni	---	---	---	---	---
156	**Horizon_Bar**	119	119	119	Generic Horizon Bar Lights for landing on ship	0.8	0 | 1 | 0	Omni	---	---	---	---	---
157	Green_Light	120	120	120	Green horizon bar Light	0.8	0 | 1 | 0	Omni	---	---	---	---	---
158	White_Light	121	121	121	White horizon bar Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
159	**Stern**	450	450	450	Generic Stern Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
160	Port_Light	451	451	451	Port stern Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
161	Starboard_Light	452	452	452	Starboard stern Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
162	VertRep_Light	453	453	453	Vertical Replenishment Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
163	**Aircraft_Carrier**	122	122	122	Generic aircraft carrier Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
164	Approach_Light	123	123	123	Aircraft Carrier approach Lights	0.8	1 | 1 | 1	Dir	75	75	---	---	---
165	Approach_Strobe_Light	124	124	124	Aircraft Carrier approach strobe Lights	0.9	1 | 1 | 1	Dir	75	75	---	2	0.1
166	**Deck**	125	125	125	Generic Deck Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
167	Aft_Light	126	126	126	Deck Aft area 1/4 mark	0.8	1 | 1 | 1	Omni	---	---	---	---	---
168	Fore_Light	127	127	127	Deck Fore area 3/4 mark	0.8	1 | 1 | 1	Omni	---	---	---	---	---
169	**Edge**	128	128	128	Generic Edge Light found on a Deck	0.8	0 | 0 | 1	Omni	---	---	---	---	---
170	Blue_Light	129	129	129	Blue Deck edge Light	0.8	0 | 0 | 1	Omni	---	---	---	---	---
171	Red_Light	454	454	454	Red Deck edge Light	0.8	1 | 0 | 0	Omni	---	---	---	---	---
172	White_Light	130	130	130	White Deck edge Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
173	Obstruction_Light	131	131	131	Deck Light indicating the presence of an object which is dangerous to an aircraft	0.8	1 | 0 | 0	Omni	---	---	---	0.5	0.33
174	**Mark_Area**	132	132	132	Generic Mark Area found on a deck	0.7	1 | 0.6 | 0	Omni	---	---	---	---	---
175	Amber_Light	133	133	133	Amber deck Light	0.7	1 | 0.6 | 0	Omni	---	---	---	---	---
176	Green_Light	134	134	134	Green deck Light	0.7	0 | 1 | 0	Omni	---	---	---	---	---
177	Red_Light	135	135	135	Red deck Light	0.7	1 | 0 | 0	Omni	---	---	---	---	---
178	**Ready_Light**	136	136	136	Generic Deck Ready Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
179	**Status**	137	137	137	Generic Status Light indicating the authority for flying operations to the FLight Deck Officer or Pilot	0.8	1 | 0.6 | 0	Omni	---	---	---	---	---
180	Amber_Light	138	138	138	Amber status Light	0.8	1 | 0.6 | 0	Omni	---	---	---	---	---
181	Green_Light	139	139	139	Green status Light (Go signal)	0.8	0 | 1 | 0	Omni	---	---	---	---	---
182	Red_Light	140	140	140	Red status Light (Stop signal)	0.8	1 | 0 | 0	Omni	---	---	---	---	---
183	Flood_Light	141	141	141	Lights used to illuminate the ground or the deck	0.8	1 | 1 | 1	Dir	30	30	---	---	---
184	**GPI**	142	142	142	Generic Glide path indicator Lights	0.7	1 | 0.6 | 0	Dir	180	54	---	---	---
185	Flashing_Green_Light	143	143	143	Green Flashing GPI	0.7	0 | 1 | 0	Dir	120	20	---	1.5	0.17
186	Flashing_Orange_Light	144	144	144	Orange Flashing GPI	0.7	1 | 0.6 | 0	Dir	180	54	---	3.9	0.065
187	Amber_Light	145	145	145	Amber GPI Light	0.7	1 | 0.6 | 0	Dir	30	8	---	---	---
188	Green_Light	146	146	146	Green GPI Light	0.7	0 | 1 | 0	Dir	30	2	---	---	---
189	Red_Light	147	147	147	Red GPI Light	0.7	1 | 0 | 0	Dir	30	6	---	---	---
190	**HAPI**	148	148	148	Generic Horizontal Approach Path Indicator Lights	0.8	1 | 1 | 1	Dir	80	18	---	---	---
191	Red_Light	149	149	149	Red HAPI Light	0.8	1 | 0 | 0	Dir	80	18	---	---	---
192	White_Light	150	150	150	White HAPI Light	0.8	1 | 1 | 1	Dir	80	18	---	---	---
193	Homing_Beacon_Light	151	151	151	Used to identify the vessel to an approaching aircraft	0.8	1 | 1 | 1	Omni	---	---	---	---	---
194	HPI_Light	152	152	152	Horizontal Path Indicator	0.8	1 | 1 | 1	Omni	---	---	---	---	---
195	No-Go_Light	153	153	153	Abort go Light	0.8	1 | 1 | 1	Dir	180	180	---	---	---
196	Nozzle_Rotation_Light	154	154	154	Nozzle rotation Light	0.6	1 | 1 | 1	Omni	---	---	---	---	---
197	Pri-Fly_Light	455	455	455	Primary FLight control Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
198	**SGSI**	155	155	155	Generic Stabilized Glide Slope Indicator (approach Light indicator)	0.8	1 | 0.6 | 0	Dir	40	6.5	---	---	---
199	Amber_Light	156	156	156	Amber SGSI Light	0.8	1 | 0.6 | 0	Dir	40	1.5	---	---	---
200	Blue_Light	157	157	157	Blue SGSI Light	0.8	0 | 0 | 1	Dir	40	1	---	---	---

	Light Hierarchy	v3.0 Light Code	v3.1 Light Code	Light Code	Description	Intensity (normalized)	Color (normalized RGB)	Directionality (type)	Width_Hor (degrees)	Width_Vert (degrees)	Intensity_Res (normalized)	Frequency (Hz)	Duty_Cycle (normalized)
201	Green_Light	158	158	158	Green SGSI Light	0.8	0 | 1 | 0	Dir	40	1	---	---	---
202	Red_Light	159	159	159	Red SGSI Light	0.8	1 | 0 | 0	Dir	40	6.5	---	---	---
203	Standby_Light	160	160	160	A means of indicating an aircraft to be at standby	0.8	1 | 1 | 1	Omni	---	---	---	---	---
204	Steady_Ship_Light	161	161	161	Steady ship Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
205	STOL	162	162	162	Generic Short Takeoff and landing Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
206	Dropline_Light	163	163	163	STOL Dropline Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
207	Lineup_Centerline_Light	164	164	164	STOL Lineup Centerline Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
208	Waveoff_Light	165	165	165	A means of indicating to approaching aircraft that recovery is not permitted and should be aborted immediately.	0.8	1 | 1 | 1	Omni	---	---	---	2	0.33
209	Cruiser	166	166	166	Generic Cruiser Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
210	Destroyer	167	167	167	Generic Destroyer Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
211	Frigate	168	168	168	Generic Frigate Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
212	Patrol	169	169	169	Generic Patrol ship Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
213	Battleship	170	170	170	Generic Battleship Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
214	Cargo	171	171	171	Generic Cargo Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
215	Subsurface	172	172	172	Generic Subsurface Vehicle Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
216	Submarine	173	173	173	Generic Submarine Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
217	Munition	174	174	174	Generic Munition Light	0.5	1 | 1 | 1	Omni	---	---	---	---	---
218	Tracer_Light	175	175	175	Light created by tracer fire effect in a bullet	0.5	1 | 0.6 | 0	Omni	---	---	---	---	---
219	Decoy_Flare_Light	176	176	176	Decoy flare Light	0.9	1 | 1 | 1	Omni	---	---	---	---	---
220	Distress_Flare_Light	177	177	177	Distress flare Light	0.9	1 | 0 | 0	Omni	---	---	---	---	---
221	Fireworks_Distress_Flare_Light	178	178	178	Fireworks flare Light	0.9	1 | 0 | 0	Omni	---	---	---	---	---
222	Flare_Light	179	179	179	Flare defensive counter measure Light effect (vs. IR guided missile)	0.9	1 | 1 | 1	Omni	---	---	---	---	---
223	Chaff_Light	180	180	180	Chaff defensive counter measure Light effect (vs. Radar guided missiles)	0.5	1 | 1 | 1	Omni	---	---	---	---	---
224	Lifeform	181	181	181	Generic Lifefrom Light (regroups all Lights that could be assigned to ainly human lifeforms)	0.7	1 | 1 | 1	Omni	---	---	---	---	---
225	Flashlight_Light	182	182	182	Hand held flashLight	0.5	1 | 1 | 1	Dir	45	45	---	---	---
226	Marshaller	183	183	183	Generic Marshaller Lights	0.7	1 | 1 | 1	Omni	---	---	---	---	---
227	Ground_Personel	184	184	184	GenericGround Personnel Lights	0.6	1 | 1 | 1	Omni	---	---	---	---	---
228	Survivor	185	185	185	GenericSurvivor Lights (on ground or sea)	0.7	1 | 1 | 1	Omni	---	---	---	1	0.33
229	Cultural	186	186	186	Generic Cultural Ground base Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
230	Point-Based	187	187	187	Generic Point based Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
231	Flood_Light	188	188	188	Lights used to illuminate the ground	0.8	1 | 1 | 1	Omni	---	---	---	---	---
232	Obstruction	189	189	189	Generic Obstruction Light - A Light indicating the presence of an object which is dangerous to an aircraft in flight.	0.9	1 | 0 | 0	Omni	---	---	---	---	---
233	Red			514	Generic Red Obstruction Light	0.9	1 | 0 | 0	Omni	---	---	---	0.5	0.5
234	Type_L864_Light			515	A flashing red obstruction Light with 20-40 flashes per minute (FAA type L-864)	0.9	1 | 0 | 0	Omni	---	---	---	0.5	0.5
235	Type _L885_Light			516	A flashing red obstruction Light with 60 flashes per minute (FAA type L-885)	0.9	1 | 0 | 0	Omni	---	---	---	1	0.5
236	Type_L810_Light			517	A steady-burning red obstruction Light (FAA type L-810)	0.5	1 | 0 | 0	Omni	---	---	---	---	---
237	White			518	Generic White Obstruction Light	1.0	1 | 1 | 1	Omni	---	---	---	0.66	0.1
238	Type_L856_Light			519	A high intensity flashing white obstruction Light with 40 flashes per minute (FAA type L-856)	1.0	1 | 1 | 1	Omni	---	---	---	0.66	0.1
239	Type_L857_Light			520	A high intensity flashing white obstruction Light with 60 flashes per minute (FAA type L-857)	1.0	1 | 1 | 1	Omni	---	---	---	1	0.1
240	Type_L865_Light			521	A medium intensity flashing white obstruction Light with 40 flashes per minute (FAA type L-865)	0.5	1 | 1 | 1	Omni	---	---	---	0.66	0.1
241	Type_L866_Light			522	A medium intensity flashing white obstruction Light with 60 flashes per minute (FAA type L-866)	0.5	1 | 1 | 1	Omni	---	---	---	1.0	0.1
242	Strobe_Light	190	190	190	Flashing Ground Light that helps to indicate position	0.8	1 | 1 | 1	Omni	---	---	---	1	0.05
243	Communication_Tower	191	191	191	Generic Communication Tower Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
244	FARP	192	192	192	Generic Forward Area Rearm/Refuel Point Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
245	IR_Light	193	193	193	Forward Area Rearm/Refuel Point IR Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
246	Strobe_Light	194	194	194	Forward Area Rearm/Refuel Point strobe Light	0.9	1 | 1 | 1	Omni	---	---	---	1	0.05
247	Y_Light	195	195	195	Forward Area Rearm/Refuel Point Y-shaped Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
248	Harbour_Light	196	196	196	Harbour Light	0.7	1 | 1 | 1	Omni	---	---	---	---	---
249	Pylon	197	197	197	Generic Power Pylon Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
250	Railroad_Junction	198	198	198	Generic Railroad Junction Lights	0.8	1 | 0 | 0	Omni	---	---	---	0.67	0.5

	Light Hierarchy	v3.0 Light Code	v3.1 Light Code	Light Code	Description	Intensity (normalized)	Color (normalized RGB)	Directionality (type)	Width_Hor (degrees)	Width_Vert (degrees)	Intensity_Res (normalized)	Frequency (Hz)	Duty_Cycle (normalized)
251	Flashing_Red_Light	199	199	199	Flashing Red rail road crossing stop Lights	0.8	1 | 0 | 0	Omni	---	---	---	0.67	0.5
252	Highway_Junction	200	200	200	Generic Highway Junction Lights	0.7	1 | 1 | 1	Omni	---	---	---	---	---
253	Bridge	201	201	201	Generic Bridge Lights	0.7	1 | 1 | 1	Omni	---	---	---	---	---
254	**Hazard**	202	202	202	Generic Harzard Light - A White Light indicating the presence of an hazard around the airport	0.8	1 | 1 | 1	Omni	---	---	---	---	---
255	Flashing_Light	203	203	203	White hazard flashing Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
256	Hi_Intensity_Light	204	204	204	White Hi-Intensity hazard Light	0.9	1 | 1 | 1	Omni	---	---	---	---	---
257	**Line-Based**	205	205	205	Generic Line based Lights (Linear features as Roads)	0.8	1 | 1 | 1	Omni	---	---	---	---	---
258	Fluorescent_Light	206	206	206	Fluorescent based Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
259	Incandescent_Light	207	207	207	Incandescent based Light	0.8	1 | 0.6 | 0.3	Omni	---	---	---	---	---
260	Mercury_Light	208	208	208	Mercury based Light	0.8	0.9 | 0.9 | 1	Omni	---	---	---	---	---
261	Metal_Halide_Light	209	209	209	Metal Halide based Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
262	Sodium_Light	210	210	210	Sodium based Light	0.8	1 | 1 | 0	Omni	---	---	---	---	---
263	**Multilane_Divided_Hwy**	211	211	211	Generic Multi-lane divided highway Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
264	Incandescent_Light	212	212	212	Incandescent based Light	0.8	1 | 0.6 | 0.3	Omni	---	---	---	---	---
265	Mercury_Light	213	213	213	Mercury based Light	0.8	0.9 | 0.9 | 1	Omni	---	---	---	---	---
266	Metal_Halide_Light	214	214	214	Metal Halide based Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
267	Sodium_Light	215	215	215	Sodium based Light	0.8	1 | 1 | 0	Omni	---	---	---	---	---
268	Median	216	216	216	Median divider Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
269	Edge	217	217	217	Highway edge/sidewalk Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
270	**Multilane_Hwy**	218	218	218	Generic Multi-lane highway Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
271	Incandescent_Light	219	219	219	Incandescent based Light	0.8	1 | 0.6 | 0.3	Omni	---	---	---	---	---
272	Mercury_Light	220	220	220	Mercury based Light	0.8	0.9 | 0.9 | 1	Omni	---	---	---	---	---
273	Metal_Halide_Light	221	221	221	Metal Halide based Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
274	Sodium_Light	222	222	222	Sodium based Light	0.8	1 | 1 | 0	Omni	---	---	---	---	---
275	Median	223	223	223	Median divider Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
276	Edge	224	224	224	Highway edge/sidewalk Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
277	**Highway**	225	225	225	Generic Single Lane Highway	0.8	1 | 1 | 1	Omni	---	---	---	---	---
278	Incandescent_Light	226	226	226	Incandescent based Light	0.8	1 | 0.6 | 0.3	Omni	---	---	---	---	---
279	Mercury_Light	227	227	227	Mercury based Light	0.8	0.9 | 0.9 | 1	Omni	---	---	---	---	---
280	Metal_Halide_Light	228	228	228	Metal Halide based Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
281	Sodium_Light	229	229	229	Sodium based Light	0.8	1 | 1 | 0	Omni	---	---	---	---	---
282	**Road**	230	230	230	Generic Road Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
283	Incandescent_Light	231	231	231	Incandescent based Light	0.8	1 | 0.6 | 0.3	Omni	---	---	---	---	---
284	Mercury_Light	232	232	232	Mercury based Light	0.8	0.9 | 0.9 | 1	Omni	---	---	---	---	---
285	Metal_Halide_Light	233	233	233	Metal Halide based Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
286	Sodium_Light	234	234	234	Sodium based Light	0.8	1 | 1 | 0	Omni	---	---	---	---	---
287	**Boulevard**	235	235	235	Generic Boulevard Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
288	Incandescent_Light	236	236	236	Incandescent based Light	0.8	1 | 0.6 | 0.3	Omni	---	---	---	---	---
289	Mercury_Light	237	237	237	Mercury based Light	0.8	0.9 | 0.9 | 1	Omni	---	---	---	---	---
290	Metal_Halide_Light	238	238	238	Metal Halide based Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
291	Sodium_Light	239	239	239	Sodium based Light	0.8	1 | 1 | 0	Omni	---	---	---	---	---
292	**Street**	240	240	240	Generic Small street Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
293	Incandescent_Light	241	241	241	Incandescent based Light	0.8	1 | 0.6 | 0.3	Omni	---	---	---	---	---
294	Mercury_Light	242	242	242	Mercury based Light	0.8	0.9 | 0.9 | 1	Omni	---	---	---	---	---
295	Metal_Halide_Light	243	243	243	Metal Halide based Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
296	Sodium_Light	244	244	244	Sodium based Light	0.8	1 | 1 | 0	Omni	---	---	---	---	---
297	**Lane**	245	245	245	Generic line based Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
298	Incandescent_Light	246	246	246	Incandescent based Light	0.8	1 | 0.6 | 0.3	Omni	---	---	---	---	---
299	**Area-Based**	247	247	247	Generic Area Lights which cover a larger area	0.8	1 | 1 | 1	Omni	---	---	---	---	---
300	Fluorescent_Light	248	248	248	Fluorescent based Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---

	Light Hierarchy	v3.0 Light Code	v3.1 Light Code	Light Code	Description	Intensity (normalized)	Color (normalized RGB)	Directionality (type)	Width_Hor (degrees)	Width_Vert (degrees)	Intensity_Res (normalized)	Frequency (Hz)	Duty_Cycle (normalized)
301	Incandescent_Light	249	249	249	Incandescent based Light	0.8	1 | 0.6 | 0.3	Omni	---	---	---	---	---
302	Mercury_Light	250	250	250	Mercury based Light	0.8	0.9 | 0.9 | 1	Omni	---	---	---	---	---
303	Metal_Halide_Light	251	251	251	Metal Halide based Light	0.8	1 | 1 | 1	Omni	---	---	---	---	---
304	Sodium_Light	252	252	252	Sodium based Light	0.8	1 | 1 | 0	Omni	---	---	---	---	---
305	Residential_Area	253	253	253	Generic Residential Area based Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
306	Bright	254	254	254	Generic Bright residential area Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
307	Incandescent_Light	255	255	255	Incandescent bright Light	0.8	1 | 0.6 | 0.3	Omni	---	---	---	---	---
308	Mercury_Light	256	256	256	Mercury bright Light	0.8	0.9 | 0.9 | 1	Omni	---	---	---	---	---
309	Dim	257	257	257	Generic Dim residential area Lights	0.7	1 | 1 | 1	Omni	---	---	---	---	---
310	Incandescent_Light	258	258	258	Incandescent dim Light	0.7	1 | 0.6 | 0.3	Omni	---	---	---	---	---
311	Mercury_Light	259	259	259	Mercury dim Light	0.7	0.9 | 0.9 | 1	Omni	---	---	---	---	---
312	Industrial_Area	260	260	260	Generic Industrial Area based Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
313	Bright	261	261	261	Generic Bright industrial area Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
314	Incandescent_Light	262	262	262	Incandescent bright Light	0.8	1 | 0.6 | 0.3	Omni	---	---	---	---	---
315	Mercury_Light	263	263	263	Mercury bright Light	0.8	0.9 | 0.9 | 1	Omni	---	---	---	---	---
316	Dim	264	264	264	Generic dim industrial area Lights	0.7	1 | 1 | 1	Omni	---	---	---	---	---
317	Incandescent_Light	265	265	265	Incandescent dim Light	0.7	1 | 0.6 | 0.3	Omni	---	---	---	---	---
318	Mercury_Light	266	266	266	Mercury dim Light	0.7	0.9 | 0.9 | 1	Omni	---	---	---	---	---
319	Downtown_Area	267	267	267	Generic City Downtown Area Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
320	Bright	268	268	268	Generic bright downtown area Lights	0.8	1 | 1 | 1	Omni	---	---	---	---	---
321	Incandescent_Light	269	269	269	Incandescent bright Light	0.8	1 | 0.6 | 0.3	Omni	---	---	---	---	---
322	Mercury_Light	270	270	270	Mercury bright Light	0.8	0.9 | 0.9 | 1	Omni	---	---	---	---	---
323	Dim	271	271	271	Generic dim downtown area Lights	0.7	1 | 1 | 1	Omni	---	---	---	---	---
324	Incandescent_Light	272	272	272	Incandescent dim Light	0.7	1 | 0.6 | 0.3	Omni	---	---	---	---	---
325	Mercury_Light	273	273	273	Mercury dim Light	0.7	0.9 | 0.9 | 1	Omni	---	---	---	---	---
326	Airport_Lighting	274	274	274	Generic Airport Lighting	0.9	1 | 1 | 1	Omni	---	---	---	---	---
327	Apron	275	275	275	Generic Apron Light	0.9	1 | 1 | 1	Omni	---	---	---	---	---
328	Entrance_Light	276	276	276	Apron entrance Light from runway or taxiway	0.9	1 | 1 | 1	Omni	---	---	---	---	---
329	Flood_Light	277	277	277	Flood Light to illuminated the Apron	0.9	1 | 1 | 1	Omni	---	---	---	---	---
330	Beacon	278	278	278	Generic Beacon Light	0.9	1 | 1 | 1	Omni	---	---	---	0.33	0.33
331	ID_Beacon_Light	279	279	279	Identification Beacon Light	0.9	1 | 1 | 1	Omni	---	---	---	0.33	0.33
332	UK_Pundit_Light-XX			523	Red UK Pundit Light where XX denotes two-letter Pundit code. NOTE: Red Omni flashing pattern is equivalent to the two-letter morse code for XX)	0.9	1 | 0 | 0	Omni	---	---	---	---	---
333	Double_White_Rotating_2sec_Light	427	427	427	Double peak White 2 sec interval Rotating Beacon	0.9	1 | 1 | 1	Omni	---	---	---	0.5	0.33
334	Double_White_Rotating_3sec_Light	428	428	428	Double peak White 3 sec interval Rotating Beacon	0.9	1 | 1 | 1	Omni	---	---	---	0.33	0.33
335	Double_White_Rotating_5sec_Light	429	429	429	Double peak White 5 sec interval Rotating Beacon	0.9	1 | 1 | 1	Omni	---	---	---	0.2	0.33
336	Double_White_Rotating_10sec_Light	439	439	439	Double peak White 10 sec interval Rotating Beacon	0.9	1 | 1 | 1	Omni	---	---	---	0.1	0.33
337	White_Rotating_2sec_Light	280	280	280	White 2 sec interval Rotating Beacon	0.9	1 | 1 | 1	Omni	---	---	---	0.5	0.33
338	White_Rotating_3sec_Light	281	281	281	White 3 sec interval Rotating Beacon	0.9	1 | 1 | 1	Omni	---	---	---	0.33	0.33
339	White_Rotating_5sec_Light	282	282	282	White 5 sec interval Rotating Beacon	0.9	1 | 1 | 1	Omni	---	---	---	0.2	0.33
340	White_Rotating_10sec_Light	445	445	445	White 10 sec interval Rotating Beacon	0.9	1 | 1 | 1	Omni	---	---	---	0.1	0.33
341	Green_Rotating_2sec_Light	283	283	283	Green 2 sec interval Rotating Beacon	0.9	0 | 1 | 0	Omni	---	---	---	0.5	0.33
342	Green_Rotating_3sec_Light	284	284	284	Green 3 sec interval Rotating Beacon	0.9	0 | 1 | 0	Omni	---	---	---	0.33	0.33
343	Green_Rotating_5sec_Light	285	285	285	Green 5 sec interval Rotating Beacon	0.9	0 | 1 | 0	Omni	---	---	---	0.2	0.33
344	Green_Rotating_10sec_Light	440	440	440	Green 10 sec interval Rotating Beacon	0.9	0 | 1 | 0	Omni	---	---	---	0.1	0.33
345	Yellow_Rotating_2sec_Light	430	430	430	Yellow 2 sec interval Rotating Beacon	0.9	1 | 1 | 0	Omni	---	---	---	0.5	0.33
346	Yellow_Rotating_3sec_Light	431	431	431	Yellow 3 sec interval Rotating Beacon	0.9	1 | 1 | 0	Omni	---	---	---	0.33	0.33
347	Yellow_Rotating_5sec_Light	432	432	432	Yellow 5 sec interval Rotating Beacon	0.9	1 | 1 | 0	Omni	---	---	---	0.2	0.33
348	Yellow_Rotating_10sec_Light	441	441	441	Yellow 10 sec interval Rotating Beacon	0.9	1 | 1 | 0	Omni	---	---	---	0.1	0.33
349	Double_White_Flashing_2sec_Light	433	433	433	Double peak White 2 sec interval Flashing Beacon	0.9	1 | 1 | 1	Omni	---	---	---	0.5	0.33
350	Double_White_Flashing_3sec_Light	434	434	434	Double peak White 3 sec interval Flashing Beacon	0.9	1 | 1 | 1	Omni	---	---	---	0.33	0.33

	Light Hierarchy	v3.0 Light Code	v3.1 Light Code	Light Code	Description	Intensity (normalized)	Color (normalized RGB)	Directionality (type)	Width_Hor (degrees)	Width_Vert (degrees)	Intensity_Res (normalized)	Frequency (Hz)	Duty_Cycle (normalized)
351	Double_White_Flashing_5sec_Light	435	435	435	Double peak White 5 sec interval Flashing Beacon	0.9	1 | 1 | 1	Omni	---	---	---	0.2	0.33
352	Double_White_Flashing_10sec_Light	442	442	442	Double peak White 10 sec interval Flashing Beacon	0.9	1 | 1 | 1	Omni	---	---	---	0.1	0.33
353	White_Flashing_2sec_Light	286	286	286	White 2 sec interval Flashing Beacon	0.9	1 | 1 | 1	Omni	---	---	---	0.5	0.33
354	White_Flashing_3sec_Light	287	287	287	White 3 sec interval Flashing Beacon	0.9	1 | 1 | 1	Omni	---	---	---	0.33	0.33
355	White_Flashing_5sec_Light	288	288	288	White 5 sec interval Flashing Beacon	0.9	1 | 1 | 1	Omni	---	---	---	0.2	0.33
356	White_Flashing_10sec_Light	446	446	446	White 10 sec interval Flashing Beacon	0.9	1 | 1 | 1	Omni	---	---	---	0.1	0.33
357	Green_Flashing_2sec_Light	289	289	289	Green 2 sec interval Flashing Beacon	0.9	0 | 1 | 0	Omni	---	---	---	0.5	0.33
358	Green_Flashing_3sec_Light	290	290	290	Green 3 sec interval Flashing Beacon	0.9	0 | 1 | 0	Omni	---	---	---	0.33	0.33
359	Green_Flashing_5sec_Light	291	291	291	Green 5 sec interval Flashing Beacon	0.9	0 | 1 | 0	Omni	---	---	---	0.2	0.33
360	Green_Flashing_10sec_Light	443	443	443	Green 10 sec interval Flashing Beacon	0.9	0 | 1 | 0	Omni	---	---	---	0.1	0.33
361	Yellow_Flashing_2sec_Light	436	436	436	Yellow 2 sec interval Flashing Beacon	0.9	1 | 1 | 0	Omni	---	---	---	0.5	0.33
362	Yellow_Flashing_3sec_Light	437	437	437	Yellow 3 sec interval Flashing Beacon	0.9	1 | 1 | 0	Omni	---	---	---	0.33	0.33
363	Yellow_Flashing_5sec_Light	438	438	438	Yellow 5 sec interval Flashing Beacon	0.9	1 | 1 | 0	Omni	---	---	---	0.2	0.33
364	Yellow_Flashing_10sec_Light	444	444	444	Yellow 10 sec interval Flashing Beacon	0.9	1 | 1 | 0	Omni	---	---	---	0.1	0.33
365	**Docking_Sytem**	292	292	292	Generic Docking System Light	0.9	1 | 0.6 | 0	Omni	---	---	---	---	---
366	Amber_Light	293	293	293	Amber Docking System Light	0.9	1 | 0.6 | 0	Omni	---	---	---	---	---
367	Green_Light	294	294	294	Green Docking System Light	0.9	0 | 1 | 0	Omni	---	---	---	---	---
368	Red_Light	295	295	295	Red Docking System Light	0.9	1 | 0 | 0	Omni	---	---	---	---	---
369	**Obstruction**	296	296	296	Generic Obstruction Light - A red Light indicating the presence of an object which is dangerous to an aircraft in flight.	0.85	1 | 0 | 0	Omni	---	---	---	0.5	0.33
370	Flashing_Light	297	297		Red Obstruction flashing Light (deprecated in CDB v3.2)	0.85	1 | 0 | 0	Omni	---	---	---	0.5	0.33
371	Hi_Intensity_Light	298	298		Red Hi-Intensity obstruction Light (deprecated in CDB v3.2)	0.9	1 | 0 | 0	Omni	---	---	---	0.5	0.33
372	**Runway**	299	299	299	Generic Runway Lights	0.9	1 | 1 | 1	Omni	---	---	---	---	---
373	**Approach_System**	300	300	300	Generic Airport Approach Lighting Systems	0.9	1 | 1 | 1	Dir	75	75	---	---	---
374	**Barrette**	301	301	301	Generic Barrette Light	0.9	1 | 1 | 1	Dir	75	75	---	---	---
375	Red_Light	302	302	302	Red barrette Light	0.9	1 | 0 | 0	Dir	75	75	---	---	---
376	White_Light	303	303	303	White barrette Light	0.9	1 | 1 | 1	Dir	75	75	---	---	---
377	Green_Light	488	488	488	Green barrette Light	0.9	0 | 1 | 0	Dir	75	75	---	---	---
378	Circling_Guidance_Light	304	304	304	Circling Guidance Light which helps on a circling approach	0.9	1 | 1 | 1	Dir	75	75	---	---	---
379	Landing_Marking_Light	305	305	305	Marking Lights that illuminate any markings that need to be visible on the runway in low visibility	0.9	1 | 1 | 1	Omni	---	---	---	---	---
380	Lead-in_Light	306	306	306	LDIN - lead-in Light system Lights	0.9	1 | 1 | 1	Dir	50	110	---	---	---
381	Optical_Landing_System	307	307	307	Optical landing system Lights	0.9	1 | 1 | 1	Omni	---	---	---	---	---
382	High_Intensity_Light	308	308	308	High intensity approach Light	0.9	1 | 1 | 1	Dir	75	75	---	---	---
383	Low_Intensity_Light	309	309	309	Low intensity approach Light	0.85	1 | 1 | 1	Dir	75	75	---	---	---
384	ODAL_Light	310	310	310	Omni directional approach Light	0.9	1 | 1 | 1	Omni	---	---	---	---	---
385	**PAPI**	311	311	311	Generic Precision approach path indicator. Provides visual glide slope indication using a single row of two or four Light units.	0.95	1 | 1 | 1	Dir	75	10	---	---	---
386	APAPI_Close_Light	312	312	312	Abbreviated Precision Approach Path Indicator closest to runway	0.95	1 | 1 | 1	Dir	75	10	---	---	---
387	APAPI_Far_Light	313	313	313	Abbreviated Precision Approach Path Indicator farthest to runway	0.95	1 | 1 | 1	Dir	75	10	---	---	---
388	TypeA_Light	314	314	314	PAPI A (farthest from runway)	0.95	1 | 1 | 1	Dir	75	10	---	---	---
389	TypeB_Light	315	315	315	PAPI B (3rd from runway)	0.95	1 | 1 | 1	Dir	75	10	---	---	---
390	TypeC_Light	316	316	316	PAPI C (2rd from runway)	0.95	1 | 1 | 1	Dir	75	10	---	---	---
391	TypeD_Light	317	317	317	PAPI D (Closest from runway)	0.95	1 | 1 | 1	Dir	75	10	---	---	---
392	RAIL_Light	318	318	318	Runway alignment indicator Lights	0.9	1 | 1 | 1	Dir	75	75	---	---	0.33
393	REIL_Light	319	319	319	Runway End Identifier Lights	0.95	1 | 1 | 1	Dir	75	75	---	2	0.1
394	**SFL**	320	320	320	Generic Sequence Flashing Lights	0.9	1 | 1 | 1	Dir	75	75	---	2	0.1
395	CAT-I	321	321	321	Approach Lighting System with sequenced flashing	0.9	1 | 1 | 1	Dir	75	75	---	2	0.1
396	CAT-II	322	322	322	Approach Lighting System with sequenced flashing	0.9	1 | 1 | 1	Dir	75	75	---	2	0.1
397	CALVERT-I	323	323	323	Approach Lighting System with sequenced flashing	0.9	1 | 1 | 1	Dir	75	75	---	2	0.1
398	CALVERT-II	324	324	324	Approach Lighting System with sequenced flashing	0.9	1 | 1 | 1	Dir	75	75	---	2	0.1
399	ALSF-1	325	325	325	Approach Lighting System with sequenced flashing	0.9	1 | 1 | 1	Dir	75	75	---	2	0.1
400	ALSF-II	326	326	326	Approach Lighting System with sequenced flashing	0.9	1 | 1 | 1	Dir	75	75	---	2	0.1

	Light Hierarchy	v3.0 Light Code	v3.1 Light Code	Light Code	Description	Intensity (normalized)	Color (normalized RGB)	Directionality (type)	Width_Hor (degrees)	Width_Vert (degrees)	Intensity_Res (normalized)	Frequency (Hz)	Duty_Cycle (normalized)
401	SSALF	327	327	327	Approach Lighting System with sequenced flashing	0.9	1 | 1 | 1	Dir	75	75	---	2	0.1
402	SSALR	328	328	328	Approach Lighting System with sequenced flashing	0.9	1 | 1 | 1	Dir	75	75	---	2	0.1
403	MALSF	329	329	329	Approach Lighting System with sequenced flashing	0.9	1 | 1 | 1	Dir	75	75	---	2	0.1
404	MALSR	330	330	330	Approach Lighting System with sequenced flashing	0.9	1 | 1 | 1	Dir	75	75	---	2	---
405	**VASI**	331	331	331	Generic Visual Approach Slope Indicator System (VASI)	0.9	1 | 1 | 1	Dir	75	10	---	---	---
406	**2Bar**	332	332	332	Generic 2 Bar Component VASI	0.9	1 | 1 | 1	Dir	75	10	---	---	---
407	First_Light	333	333	333	2-Bar VASIS (1st bar closest to threshold)	0.9	1 | 1 | 1	Dir	75	10	---	---	---
408	Second_Light	334	334	334	2-Bar VASIS (2nd bar farthest from threshold)	0.9	1 | 1 | 1	Dir	75	10	---	---	---
409	**3Bar**	335	335	335	Generic 3 Bar component VASI	0.9	1 | 1 | 1	Dir	75	10	---	---	---
410	First_Light	336	336	336	3-Bar VASIS (1st bar closest to threshold)	0.9	1 | 1 | 1	Dir	75	10	---	---	---
411	Second_Light	337	337	337	3-Bar VASIS (2nd bar, In between 1st and 3rd)	0.9	1 | 1 | 1	Dir	75	10	---	---	---
412	Third_Light	338	338	338	3-Bar VASIS (3rd bar farthest from threshold)	0.9	1 | 1 | 1	Dir	75	10	---	---	---
413	LCVASI_Light	339	339	339	Low-cost VASI Light	0.9	1 | 1 | 1	Dir	75	10	---	---	---
414	TypeP_Light	340	340	340	PVASI pulsating Light	0.9	1 | 1 | 1	Dir	75	10	---	---	---
415	**TypeT**	341	341	341	Generic T Shaped VASI (T-VASIS)	0.9	1 | 1 | 1	Dir	75	10	---	---	---
416	Fly-down_Light	342	342	342	Fly Down Lights	0.9	1 | 1 | 1	Dir	75	7	---	---	---
417	Wing_Bar_Light	343	343	343	T-VASIS wing bar Light	0.9	1 | 1 | 1	Dir	75	10	---	---	---
418	**2.50_Degree**	344	344	344	Generic 2.50 degree T-VASI	0.9	1 | 1 | 1	Dir	75	2.5	---	---	---
419	Fly-Up1_Light	345	345	345	T-VASIS Fly-up 1 (closest to Wing Bar) for 2.5 degree Glide slope	0.9	1 | 1 | 1	Dir	75	2.5	---	---	---
420	Fly-Up2_Light	346	346	346	T-VASIS Fly-up 2 (closest to Wing Bar) for 2.5 degree Glide slope	0.9	1 | 1 | 1	Dir	75	2.4166	---	---	---
421	Fly-Up3_Light	347	347	347	T-VASIS Fly-up 3 (farthest to Wing Bar) for 2.5 degree Glide slope	0.9	1 | 1 | 1	Dir	75	2.3334	---	---	---
422	**2.75_Degree**	348	348	348	Generic 2.75 degree T-VASI	0.9	1 | 1 | 1	Dir	75	2.75	---	---	---
423	Fly-Up1_Light	349	349	349	T-VASIS Fly-up 1 (closest to Wing Bar) for 2.7 degree Glide slope	0.9	1 | 1 | 1	Dir	75	2.75	---	---	---
424	Fly-Up2_Light	350	350	350	T-VASIS Fly-up 2 (closest to Wing Bar) for 2.7 degree Glide slope	0.9	1 | 1 | 1	Dir	75	2.6666	---	---	---
425	Fly-Up3_Light	351	351	351	T-VASIS Fly-up 3 (farthest to Wing Bar) for 2.7 degree Glide slope	0.9	1 | 1 | 1	Dir	75	2.5834	---	---	---
426	**3.00_Degree**	352	352	352	Generic 3.00 degree T-VASI	0.9	1 | 1 | 1	Dir	75	3	---	---	---
427	Fly-Up1_Light	353	353	353	T-VASIS Fly-up 1 (closest to Wing Bar) for 3.0 degree Glide slope	0.9	1 | 1 | 1	Dir	75	3	---	---	---
428	Fly-Up2_Light	354	354	354	T-VASIS Fly-up 2 (closest to Wing Bar) for 3.0 degree Glide slope	0.9	1 | 1 | 1	Dir	75	2.9166	---	---	---
429	Fly-Up3_Light	355	355	355	T-VASIS Fly-up 3 (farthest to Wing Bar) for 3.0 degree Glide slope	0.9	1 | 1 | 1	Dir	75	2.8334	---	---	---
430	**3.25_Degree**	356	356	356	Generic 3.25 degree T-VASI	0.9	1 | 1 | 1	Dir	75	3.25	---	---	---
431	Fly-Up1_Light	357	357	357	T-VASIS Fly-up 1 (closest to Wing Bar) for 3.25 degree Glide slope	0.9	1 | 1 | 1	Dir	75	3.25	---	---	---
432	Fly-Up2_Light	358	358	358	T-VASIS Fly-up 2 (closest to Wing Bar) for 3.25 degree Glide slope	0.9	1 | 1 | 1	Dir	75	3.1666	---	---	---
433	Fly-Up3_Light	359	359	359	T-VASIS Fly-up 3 (farthest to Wing Bar) for 3.25 degree Glide slope	0.9	1 | 1 | 1	Dir	75	3.0834	---	---	---
434	**3.50_Degree**	360	360	360	Generic 3.5 degree T-VASI	0.9	1 | 1 | 1	Dir	75	3.5	---	---	---
435	Fly-Up1_Light	361	361	361	T-VASIS Fly-up 1 (closest to Wing Bar) for 3.5 degree Glide slope	0.9	1 | 1 | 1	Dir	75	3.5	---	---	---
436	Fly-Up2_Light	362	362	362	T-VASIS Fly-up 2 (closest to Wing Bar) for 3.5 degree Glide slope	0.9	1 | 1 | 1	Dir	75	3.4166	---	---	---
437	Fly-Up3_Light	363	363	363	T-VASIS Fly-up 3 (farthest to Wing Bar) for 3.5 degree Glide slope	0.9	1 | 1 | 1	Dir	75	3.3334	---	---	---
438	**3.75_Degree**	364	364	364	Generic 3.75 degree T-VASI	0.9	1 | 1 | 1	Dir	75	3.75	---	---	---
439	Fly-Up1_Light	365	365	365	T-VASIS Fly-up 1 (closest to Wing Bar) for 3.75 degree Glide slope	0.9	1 | 1 | 1	Dir	75	3.75	---	---	---
440	Fly-Up2_Light	366	366	366	T-VASIS Fly-up 2 (closest to Wing Bar) for 3.75 degree Glide slope	0.9	1 | 1 | 1	Dir	75	3.6666	---	---	---
441	Fly-Up3_Light	367	367	367	T-VASIS Fly-up 3 (farthest to Wing Bar) for 3.75 degree Glide slope	0.9	1 | 1 | 1	Dir	75	3.5834	---	---	---
442	**4.00_Degree**	368	368	368	Generic 4.00 degree T-VASI	0.9	1 | 1 | 1	Dir	75	4	---	---	---
443	Fly-Up1_Light	369	369	369	T-VASIS Fly-up 1 (closest to Wing Bar) for 4.0 degree Glide slope	0.9	1 | 1 | 1	Dir	75	4	---	---	---
444	Fly-Up2_Light	370	370	370	T-VASIS Fly-up 2 (closest to Wing Bar) for 4.0 degree Glide slope	0.9	1 | 1 | 1	Dir	75	3.9166	---	---	---
445	Fly-Up3_Light	371	371	371	T-VASIS Fly-up 3 (farthest to Wing Bar) for 4.0 degree Glide slope	0.9	1 | 1 | 1	Dir	75	3.8334	---	---	---
446	**Centerline**	372	372	372	Generic runway centerline Light	0.9	1 | 1 | 1	Bi-Dir	75	75	---	---	---
447	Red_Light	373	373	373	Unidirectional Red runway centerline Light	0.9	1 | 0 | 0	Dir	75	75	---	---	---
448	White_Light	374	374	374	Unidirectional White runway centerline Light	0.9	1 | 1 | 1	Dir	75	75	---	---	---
449	White_White_Light	375	375	375	Bidirectional White runway centerline Light	0.9	1 | 1 | 1	Bi-Dir	75	75	---	---	---
450	White_Red_Light	376	376	376	Bidirectional White/Red runway centerline Light	0.9	1 | 1 | 1	Bi-Dir	75	75	---	---	---

	Light Hierarchy	v3.0 Light Code	v3.1 Light Code	Light Code	Description	Intensity (normalized)	Color (normalized RGB)	Directionality (type)	Width_Hor (degrees)	Width_Vert (degrees)	Intensity_Res (normalized)	Frequency (Hz)	Duty_Cycle (normalized)
451	Red_Red_Light		511	511	Bidirectional Red runway centerline Light	0.9	1 | 0 | 0	Bi-Dir	75	75	---	---	---
452	**Edge**	377	377	377	Generic Runway Edge Lights	0.9	1 | 1 | 1	Bi-Dir	180	180	---	---	---
453	White_Light	378	378	378	Unidirectional White Edge Light	0.9	1 | 1 | 1	Dir	180	180	---	---	---
454	Amber_Light	379	379	379	Unidirectional Amber Edge Light	0.9	1 | 0.6 | 0	Dir	180	180	---	---	---
455	Red_Light	380	380	380	Unidirectional Red Edge Light	0.9	1 | 0 | 0	Dir	180	180	---	---	---
456	Blue_Light	381	381	381	Unidirectional Blue Edge Light	0.9	0 | 0 | 1	Dir	180	180	---	---	---
457	White_White_Light	382	382	382	Bidirectional White Edge Light	0.9	1 | 1 | 1	Bi-Dir	180	180	---	---	---
458	White_Amber_Light	383	383	383	White-Amber Edge Light	0.9	1 | 1 | 1	Bi-Dir	180	180	---	---	---
459	White_Red_Light	384	384	384	White-Red Edge Light	0.9	1 | 1 | 1	Bi-Dir	180	180	---	---	---
460	White_Blue_Light	385	385	385	White-Blue Edge Light	0.9	1 | 1 | 1	Bi-Dir	180	180	---	---	---
461	Amber_Amber_Light	386	386	386	Bidirectional Amber Edge Light	0.9	1 | 0.6 | 0	Bi-Dir	180	180	---	---	---
462	Amber_Red_Light	387	387	387	Amber-Red Edge Light	0.9	1 | 0.6 | 0	Bi-Dir	180	180	---	---	---
463	Amber_Blue_Light	388	388	388	Amber-Blue Edge Light	0.9	1 | 0.6 | 0	Bi-Dir	180	180	---	---	---
464	Blue_Red_Light	389	389	389	Blue-Red Edge Light	0.9	0 | 0 | 1	Bi-Dir	180	180	---	---	---
465	Red_Red_Light	390	390	390	Bidirectional Red Edge Light	0.9	1 | 0 | 0	Bi-Dir	180	180	---	---	---
466	Blue_Blue_Light	391	391	391	Bidirectional Blue Edge Light	0.9	0 | 0 | 1	Bi-Dir	180	180	---	---	---
467	End_Wing_Light	392	392	392	Runway End Wing Lights	0.9	1 | 0 | 0	Dir	180	180	---	---	---
468	End_Light	393	393	393	Runway End Lights	0.9	1 | 0 | 0	Dir	180	180	---	---	---
469	Flood_Light	394	394	394	Runway flood Lights	0.9	1 | 1 | 1	Omni	---	---	---	---	---
470	**Overrun**	395	395	395	Generic Overun Light - A Light which indicated runway over run area	0.9	1 | 0.6 | 0	Dir	150	90	---	---	---
471	Amber_Light	396	396	396	Amber overrun Light	0.9	1 | 0.6 | 0	Dir	150	90	---	---	---
472	Blue_Light	397	397	397	Blue overrun Light	0.9	0 | 0 | 1	Dir	150	90	---	---	---
473	Red_Light	398	398	398	Red overrun Light	0.9	1 | 0 | 0	Dir	150	90	---	---	---
474	Threshold_Wing_Light	399	399	399	Threshold wing Lights	0.9	0 | 1 | 0	Dir	180	180	---	---	---
475	Threshold_Light	400	400	400	Runway threshold Lights: used to identify the landing threshold of the runway	0.9	0 | 1 | 0	Dir	180	180	---	---	---
476	Touchdown_Zone_Light	401	401	401	Touchdown Zone Lights: used to identify the appropriate landing area on the runway after the threshold	0.9	1 | 1 | 1	Dir	180	180	---	---	---
477	LAHSO_Light	402	402	402	Land and hold Short Operations Light: runway intersecting stop Lights	0.9	1 | 0.6 | 0	Omni	---	---	---	---	---
478	**Taxiway**	403	403	403	Generic Airport Taxiway Lights	0.9	0 | 0 | 1	Omni	---	---	---	---	---
479	Apron_Entrance_Light	404	404	404	Apron Entrance Light which indication area where taxi enters apron area	0.9	0 | 0 | 1	Omni	---	---	---	---	---
480	CAT-III_Hold_Bar_Light	405	405	405	Category III Hold bar Light	0.9	0 | 1 | 0	Dir	180	180	---	---	---
481	**Centerline**	406	406	406	Generic Centerline Taxiway Lights	0.9	0 | 1 | 0	Dir	90	110	---	---	---
482	Aligned_Light	407	407	407	ALighted Light for a straight sequence of a taxiway	0.9	0 | 1 | 0	Dir	90	110	---	---	---
483	Curved_Light	408	408	408	Curved Lights for a curved sequence of a taxiway	0.9	0 | 1 | 0	Dir	50	110	---	---	---
484	**Edge**	409	409	409	Generic Taxiway edge Lights	0.9	0 | 0 | 1	Omni	---	---	---	---	---
485	Blue_Light	425	425	425	Blue Taxi edge Light	0.9	0 | 0 | 1	Omni	---	---	---	---	---
486	White_Light	426	426	426	White Taxi edge Light	0.9	1 | 1 | 1	Omni	---	---	---	---	---
487	**High-speed**	410	410	410	Generic Taxiway high speed area Lights	0.9	1 | 0.6 | 0	Dir	50	110	---	---	---
488	Amber_Light	411	411	411	Amber high-speed Lights	0.9	1 | 0.6 | 0	Dir	50	110	---	---	---
489	Green_Light	412	412	412	Green high-speed Lights	0.9	0 | 1 | 0	Dir	50	110	---	---	---
490	**Lead-on**	413	413	413	Generic Lead-On Light	0.9	0 | 1 | 0	Omni	---	---	---	---	---
491	Green_Light		489	489	Green Lead-On Light	0.9	0 | 1 | 0	Omni	---	---	---	---	---
492	Yellow_Light		490	490	Yellow Lead-On Light	0.9	1 | 1 | 0	Omni	---	---	---	---	---
493	**Lead-off**		491	491	Generic Lead-Off Light	0.9	0 | 1 | 0	Omni	---	---	---	---	---
494	Green_Light		492	492	Green Lead-Off Light	0.9	0 | 1 | 0	Omni	---	---	---	---	---
495	Yellow_Light		493	493	Yellow Lead-Off Light	0.9	1 | 1 | 0	Omni	---	---	---	---	---
496	No-entry_Light	414	414	414	No entry zone Lights	0.9	1 | 0 | 0	Omni	---	---	---	---	---
497	Runway_Guard	415	415	415	Runway guard Lights	0.9	1 | 1 | 1	Omni	---	---	---	---	---
498	Stop_Bar_Light	416	416	416	Stop Bar Lights	0.9	1 | 0 | 0	Dir	180	180	---	---	---
499	**Clearance**	417	417	417	Generic Clearance bar Light. They are located at "hold short" positions on taxiways in order to increase the visibility of	0.9	1 | 1 | 0	Dir	---	---	---	---	---
500	Unidirectional_Light			512	Unidirectional Taxiway Clearance Light (used when the hold is intended for one direction only)	0.9	0 | 1 | 0	Dir	7	7	---	---	---

	Light Hierarchy	v3.0 Light Code	v3.1 Light Code	Light Code	Description	Intensity (normalized)	Color (normalized RGB)	Directionality (type)	Width_Hor (degrees)	Width_Vert (degrees)	Intensity_Res (normalized)	Frequency (Hz)	Duty_Cycle (normalized)
501	Bidirectional_Light			513	Bidirectional Taxiway Clearance Light (used when the hold is intended for two directions)	0.9	1 | 1 | 0	Dir	7	7	---	---	---
502	Guard	418	418	418	Generic RGL (Runway Guard Light) is used to enhance the visibility of taxiway holding positions on an airport	0.9	1 | 1 | 1	Omni	---	---	---	---	---
503	Type1_Light	419			(deprecated in CDB v3.1)	0.9	1 | 1 | 1	Omni	---	---	---	---	---
504	Type2_Light	420			(deprecated in CDB v3.1)	0.9	1 | 1 | 1	Omni	---	---	---	---	---
505	Type3_Light	421			(deprecated in CDB v3.1)	0.9	1 | 1 | 1	Omni	---	---	---	---	---
506	Type4_Light	422			(deprecated in CDB v3.1)	0.9	1 | 1 | 1	Omni	---	---	---	---	---
507	Wind_Indicator_Light	423	423	423	Wind indicator Light	0.9	1 | 1 | 1	Omni	---	---	---	---	---
508	Windsock_Light	424	424	424	Windsock Light used to illuminate the windsock in poor visibility	0.9	1 | 1 | 1	Omni	---	---	---	---	---
509	Heliport	457	457	457	Generic Heliport Lights	0.9	0 | 0 | 1	Omni	---	---	---	---	---
510	Approach_System	458	458	458	Generic Heliport Approach System Lights	0.9	0 | 1 | 0	Dir	90	110	---	---	---
511	Landing_Marking	460	460	460	Generic Landing Marking Light on Heliport Approach System	0.9	1 | 1 | 1	Dir	75	10	---	---	---
512	Amber_Light	465	465	465	Heliport Approach Landing Marking Amber Light	0.9	1 | 1 | 1	Dir	75	10	---	---	---
513	Green_Light	463	463	463	Heliport Approach Landing Marking Green Light	0.9	1 | 1 | 1	Dir	75	10	---	---	---
514	Red_Light	464	464	464	Heliport Approach Landing Marking Red Light	0.9	1 | 1 | 1	Dir	75	10	---	---	---
515	Edge	459	459	459	Generic Heliport Edge Lights	0.9	0 | 0 | 1	Omni	---	---	---	---	---
516	White_White_Light	462	462	462	White White Heliport Edge Light	0.9	0 | 0 | 1	Omni	---	---	---	---	---
517	White_Light	461	461	461	White Heliport Edge Light	0.9	1 | 1 | 1	Omni	---	---	---	---	---

Appendix F

F. CDB Model Components

Model components represent concrete parts found on 3D models whether the model is used as a moving model or a cultural feature. This appendix constitutes a data dictionary of names that can be used for zone names, as defined in chapter 6. Note that names are independent from simulation models and client devices since they represent real and tangible objects or components found on 3D models. The XML file containing the CDB Model Components is part of the CDB Specification Distribution Package and can be found in the following file:

```
\CDB\Metadata\Model_Components.xml
```

Note: As of CDB Specification 3.2, the list of CDB model components is no longer presented here to avoid the risk of miscorrelation between the appendix and the metadata. The list is now exclusively found in the Metadata folder.

Appendix G

G. Gamma Tutorial

G.1 Introduction[9]

There is nominally no gamma correction done to the stored samples of CDB imagery files. As a result, a gamma of 1/2.2 should be applied to imagery data when viewing it through a (sRGB-calibrated) monitor with gamma of 2.2. The CDB specification recommends the sRGB IEC 61966-2 standard when performing the calibration of displays (at DBGF or a simulator). The sRGB standard provides the necessary guidelines for the handling of gamma, and of color (in a device-independent fashion) under specified viewing conditions.

It would be convenient for graphics programmers if all of the components of an imaging system were linear. The voltage coming from an electronic camera would be directly proportional to the intensity (power) of light in the scene; the light emitted by a CRT would be directly proportional to its input voltage, and so on. However, real-world devices do not behave in this way.

Real imaging systems will have several components, and more than one of these can be nonlinear. If all of the components have transfer characteristics that are power functions, then the transfer function of the entire system is also a power function. The exponent (gamma) of the whole system's transfer function is just the product of all of the individual exponents (gammas) of the separate stages in the system. Also, stages that are linear pose no problem, since a power function with an exponent of 1.0 is really a linear function. So a linear transfer function is just a special case of a power function, with a gamma of 1.0. Thus, as long as our imaging system contains only stages with linear and power-law transfer functions, we can meaningfully talk about the gamma of the entire system. This is indeed the case with most real imaging systems.

If the overall gamma of an imaging system is 1.0, its output is linearly proportional to its input. This means that the ratio between the intensities of any two areas in the reproduced image will be the same as it was in the original scene. It might seem that this should always be the goal of an imaging system: to accurately reproduce the tones of the original scene. Alas, that is not the case.

When the reproduced image is to be viewed in “bright surround” conditions, where other white objects nearby in the room have about the same brightness as white in the image, then an overall gamma of 1.0 does indeed give real-looking reproduction of a natural scene. Photographic prints

[9] Graphics Specification (see http://www.w3.org/TR/PNG-GammaAppendix)

viewed under room light and computer displays in bright room light are typical “bright surround” viewing conditions.

However, sometimes images are intended to be viewed in “dark surround” conditions, where the room is substantially black except for the image. This is typical of the way movies and slides (transparencies) are viewed by projection. Under these circumstances, an accurate reproduction of the original scene results in an image that human viewers judge as “flat” and lacking in contrast. It turns out that the projected image needs to have a gamma of about 1.5 relative to the original scene for viewers to judge it “natural”. Thus, slide film is designed to have a gamma of about 1.5, not 1.0.

There is also an intermediate condition called “dim surround”, where the rest of the room is still visible to the viewer, but is noticeably darker than the reproduced image itself. This is typical of television viewing, at least in the evening, as well as subdued-light computer work areas. In dim surround conditions, the reproduced image needs to have a gamma of about 1.25 relative to the original scene in order to look natural.

The requirement for boosted contrast (gamma) in dark surround conditions is due to the way the human visual system works, and applies equally well to computer monitors. Thus, a modeler trying to achieve the maximum realism for the images it displays really needs to know what the room lighting conditions are, and adjust the gamma of the displayed image accordingly.

If asking the user about room lighting conditions is inappropriate or too difficult, it is reasonable to assume that the overall gamma (viewing_gamma as defined below) is somewhere between 1.0 and 1.25. That's all that most systems that implement gamma correction do.

According to PNG (Portable Network Graphics) Specification Version 1.0, W3C Recommendation 01-October-1996 Appendix, Gamma Tutorial,

(http://www.w3.org/TR/PNG-GammaAppendix):

“All display systems, almost all photographic film, and many electronic cameras have nonlinear signal-to-light-intensity or intensity-to-signal characteristics. Fortunately, all of these nonlinear devices have a transfer function that is approximated fairly well by a single type of mathematical function: a power function. This power function has the general equation

output = input ^ gamma

where ^ denotes exponentiation, and “gamma” (often printed using the Greek letter gamma, thus the name) is simply the exponent of the power function.

By convention, “input” and “output” are both scaled to the range [0..1], with 0 representing black and 1 representing maximum white. Normalized in this way, the power function is completely described by a single number, the exponent “gamma”.

So, given a particular device, we can measure its output as a function of its input, fit a power function to this measured transfer function, extract the exponent, and call it gamma[10]. We often say “this device has a gamma of 2.5” as a shorthand for “this device has a power-law response with an exponent of 2.5”. We can also talk about the gamma of a mathematical transform, or of a lookup table in a frame buffer, so long as the input and output of the thing are related by the power-law expression above.

Real imaging systems will have several components, and more than one of these can be nonlinear. If all of the components have transfer characteristics that are power functions, then the transfer function of the entire system is also a power function. The exponent (gamma) of the whole system's transfer function is just the product of all of the individual exponents (gammas) of the separate stages in the system.

Also, stages that are linear pose no problem, since a power function with an exponent of 1.0 is really a linear function. So a linear transfer function is just a special case of a power function, with a gamma of 1.0.

[10] Gamma refers to the measured transfer function that represents the non-linear response of the device. CRT devices have traditionally dominated the computer graphics/television industry and have an exponential response; as a result, the “gamma” of a device is often represented as a single value representing the “best-fit” exponent of power-law response of the device. A more accurate and general approach to gamma involves a look-up table which captures the (arbitrary) response of the device.

Thus, as long as our imaging system contains only stages with linear and power-law transfer functions, we can meaningfully talk about the gamma of the entire system. This is indeed the case with most real imaging systems."[11]

In an ideal world, sample values would be stored in floating point, there would be lots of precision, and it wouldn't really matter much. But in reality, we're always trying to store images in as few bits as we can.

If we decide to use samples that are linearly proportional to intensity, and do the gamma correction in the frame buffer LUT, it turns out that we need to use at least 12-16 bits for each of red, green, and blue to have enough precision in intensity. With any less than that, we will sometimes see "contour bands" or "Mach bands" in the darker areas of the image, where two adjacent sample values are still far enough apart in intensity for the difference to be visible.

However, through an interesting coincidence, the human eye's subjective perception of brightness is related to the physical stimulation of light intensity in a manner that is very much like the power function used for gamma correction. If we apply gamma correction to measured (or calculated) light intensity before quantizing to an integer for storage in a frame buffer, we can get away with using many fewer bits to store the image. In fact, 8 bits per color is almost always sufficient to avoid contouring artifacts. This is because, since gamma correction is so closely related to human perception, we are assigning our 256 available sample codes to intensity values in a manner that approximates how visible those intensity changes are to the eye. Compared to a linear-sample image, we allocate fewer sample values to brighter parts of the tonal range and more sample values to the darker portions of the tonal range.

Thus, for the same apparent image quality, images using gamma-encoded sample values need only about two-thirds as many bits of storage as images using linear samples.

If we consider a pipeline that involves capturing (or calculating) an image, storing it in an image file, reading the file, and displaying the image on some sort of display screen, there are at least 5

[11] Copyright © 1994-2004 W3C® (MIT, ERCIM, Keio)

places in the pipeline that could have nonlinear transfer functions. Let's give each a specific name for their characteristic gamma[12]:

(1) Camera_gamma (γ_c): The characteristic of the image sensor.

(2) Encoding_gamma (γ_e): The gamma of any transformation performed by the software writing the image file.

(3) Decoding_gamma (γ_d): The gamma of any transformation performed by any software reading the image file.

(4) LUT_gamma (γ_{lut}): The gamma of the frame buffer LUT, if present.

(5) CRT_gamma (γ_{crt}): The gamma of the device, i.e. the nonlinear signal-to-light-intensity or intensity-to-signal characteristics. For CRT-based devices, this is generally 2.5.

In addition, let's add a few other names:

(1) File_gamma (γ_f): The gamma of the image in the file, relative to the original scene, i.e. $\gamma_f = \gamma_c \gamma_e$

(2) DS_gamma (γ_{DS}): The gamma of the "display system" downstream of the frame buffer. In this context, the term display system encompasses everything after the frame buffer, that is $\gamma_{DS} = \gamma_{lut} \gamma_{CRT}$

[12] These definitions have been kindly provided by the World Wide Web Consortium and are included in the PNG file format specification available at http://www.w3.org/pub/WWW/TR/REC-png-multi.html.

(3) Viewing_gamma (γ_v): The overall gamma that we want to obtain to produce pleasing images generally 1.0 to 1.25.

When the file_gamma is not 1.0, we know that some form of gamma correction has been done on the sample values in the file, and we call them "gamma corrected" samples. However, since there can be so many different values of gamma in the image display chain, and some of them are not known at the time the image is written, the samples are not really being "corrected" for a specific display condition. We are really using a power function in the process of encoding an intensity range into a small integer field, and so it is more correct to say "gamma encoded" samples instead of "gamma corrected" samples. The CDB specification does not rely on such gamma encoding in order to achieve smaller integer number representations. Instead, the CDB specification relies on standard compression algorithms to achieve an efficient representation of color imagery.[13]

When displaying an image file on the simulator, the image-decoding software is responsible for making the overall gamma of the system equal to the desired viewing_gamma, by selecting the decoding_gamma appropriately. If the viewing condition is different from the specification, then the decoding process must compensate. This can be done by modifying the gamma values in equation G-1 below by the appropriate factor. If one does modify the gamma values in equation G-1 below, extreme care must be taken to avoid quantization errors when working with 24 bit images. The display_gamma should be measured (and known) for the display rendering the image (either at the DB generation workstation or the simulator). The correct viewing_gamma depends on lighting conditions, and that will generally have to come from the user. In dimly lit office environments, the generally preferred value for viewing gamma is in the vicinity of 1.125[14]. In many digital video systems, camera_gamma is about 0.5. CRT_gamma is typically 2.2, while encoding_gamma, decoding_gamma, and LUT_gamma are all 1.0. As a result, viewing_gamma ends up being about 1.125. Coincidently, this happens to be the optimal viewing gamma for an ambient luminance level of 64 lux or 5 ft-lbt.

$$\gamma_c \gamma_d \gamma_{DS} = \gamma_v \qquad \text{(eq. G-1)}$$

[13] The JPEG-2000 standard is based on the *sRGB* default color space per the IEC 61966-2-1 Standard which calls for a gamma 2.2 under the specified viewing conditions

[14] Historically, viewing gammas of 1.5 have been used for viewing projected slides in a dark room and viewing gammas of 1.25 have been used for viewing monitors in a very dim room. This very dim room value of 1.25 has been used extensively in television systems and assumes a ambient luminance level of approximately 15 lux (or 1.4 ft-lb). The current proposal assumes an encoding ambient luminance level of 64 lux (or 5. ft-lb) which is more representative of a dim room in viewing computer generated imagery or a FAA level-D approved flight simulator visual system. Such a system assumes a viewing gamma of 1.125 and is thus consistent with the ITU-R BT.709 standard.

$$\gamma_c \gamma_d \gamma_{lut} \gamma_{crt} = \gamma_v$$

$$0.511 \times 1.0 \times 1.0 \times 2.2 = 1.125 = \gamma_v$$

In a complex system such as a flight simulator, the system architect must be aware of the gamma at every stage of the system, starting from the source of the imagery (e.g. camera or satellite) right through to the simulator's display device. His objective is to ensure that product of all gammas match the viewing gamma of the simulator.

Given the above assumptions, and our objective of ensuring that the product of all gammas in the viewing chain equals the viewing gamma, the modeler will end up (subjectively) adjusting images to an equivalent file gamma of 1.25.

The bottom portion of the illustration show the path taken by the CDB imagery as it is ingested first by the real-time publisher, then by the IG, the IG color look-up tables and finally through to the visual display system. In this example, we will assume the following:

1. The imagery file in the CDB is unmodified (i.e. those produced by the Adobe Photoshop at the DBGF). Note that as a result of viewing gamma of $\gamma_v = 1.25$, the file gamma ended up at $\gamma_f = 1.25$ at the DBGF. As a result, the CDB also has a file gamma of $\gamma_f = 1.25$
2. The IG performs all of its internal operations in a linear color space (i.e. the IG_gamma is $\gamma_{IG} = 1.00$)
3. The simulator visual system produces an average scene brightness of approximately 6 ft-lamberts: under these viewing conditions, the viewing gamma is $\gamma_v = 1.125$.
4. The measured gamma of the visual display system is $\gamma_{crt} = 2.025$
5. The content of the IG's color look-up tables is adjusted to compensate for the gamma of the visual display system, i.e. it is loaded with $\gamma_{lut} = 1/2.025$

Given the above assumptions, and our objective of ensuring that the product of all gammas in the chain equals the viewing gamma of 1.125, the required visual run-time publisher gamma must account for the difference in viewing gamma at the DBGF and at the simulator. As a result, the publisher gamma must be (1.125/1.25).

G.2 Harmonization of Gamma at DBGF with Gamma of Simulator Visual System

Both the modelers and the visual system architects should be keenly aware of the handling of gamma at the Database Generation Facility and at the simulator. Figure G-1: Typical Handling of Gamma at DBGF and Simulator, illustrates the typical handling of gamma in both of these cases.

The top portion of the illustration shows the path taken by source data as a modeler is viewing it at this workstation via the application software. In this example, we will assume the following:

1. The DBGF imagery application is Adobe Photoshop. The default color space profile used by Adobe Photoshop (i.e. the *.icm file) is the sRGB Color Space Profile which is defined by the sRGB standard to be a gamma of 2.2, therefore the Photoshop uses a $\gamma_{lut} = 1/2.2$
2. The DBGF workstation is running Windows (therefore the O/S does not gammatize the imagery before sending it to the display, $\gamma_d = 1.25$)
3. The measure gamma of the DBGF workstation monitor is $\gamma_{CRT} = 2.2$
4. The DBGF workstation is located in a dimly lit room, so the viewing gamma is in effect $\gamma_v = 1.25$

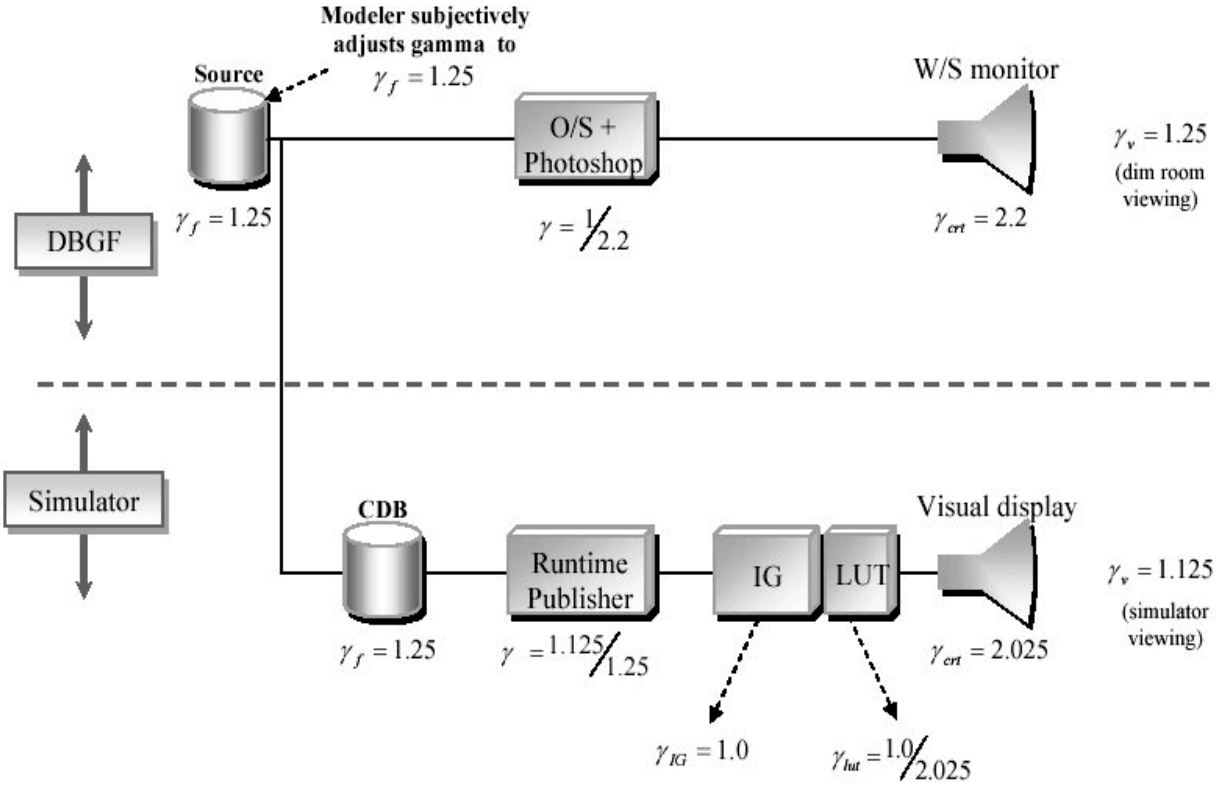


Figure G-1: Typical Handling of Gamma at DBGF and Simulator

G.3 Handling of Color

The default CDB specification color space follows the same convention as the Windows sRGB Color Space Profile. *sRGB* is the default color space in Windows, based on the IEC 61966-2-1 Standard. A *sRGB* compliant device does not have to provide a profile or other support for color management to work well.

Nonetheless, whether calibrated or not to the IEC Standard, all variants of RGB are typically close enough that undemanding viewers can get by with simply displaying the data without color correction. By storing calibrated RGB, the CDB standard retains compatibility with existing database tools and software programs that expect RGB data, yet provides enough information for conversion to XYZ in applications that need precise colors. Thus, the CDB specification gets the best of both worlds.

Full compliance to the CDB specification requires adherence to the color space described in this section; however, in virtually all cases, direct use of un-calibrated RGB is sufficient. The builders of Synthetic Environment Databases and the users of Visual Systems should be aware of

these color space conventions; significant deviation from the underlying IEC assumptions may yield significant color differences.

The CDB specification encoded RGB color tri-stimulus values assume the following:

(1) Display luminance level: 80 cd/m2

(2) Display white point x = 0.3127, y = 0.3291 (D65)

(3) Display model Offset (R, G and B): 0.055

(4) Display Gun/Phosphor Gamma (R, G, and B): 2.2

Table G-1: CIE Chromaticity for CDB Reference Primaries & CIE Standard Illuminant

	Red	Green	Blue	D65 (white)
X	0.6400	0.3000	0.1500	0.3127
Y	0.3300	0.6000	0.0600	0.3291
Z	0.0300	0.1000	0.7900	0.3583

According to PNG (Portable Network Graphics) Specification Version 1.0, W3C Recommendation 01-October-1996 Appendix, Color Tutorial,

(http://www.w3.org/TR/PNG-GammaAppendix):

"The color of an object depends not only on the precise spectrum of light emitted or reflected from it, but also on the observer, their species, what else they can see at the same time, even what they have recently looked at. Furthermore, two very different spectra can produce exactly the same color sensation. Color is not an objective property of real-world objects; it is a subjective, biological sensation. However, by making some simplifying assumptions (such as: we are talking about *human* vision) it is possible to produce a mathematical model of color and thereby obtain good color accuracy.

G.3.1 Device-dependent Color

Display the same RGB data on three different monitors, side by side, and you will get a noticeably different color balance on each display. This is because each monitor emits a slightly different shade and intensity of red, green, and blue light. RGB is an example of a device-dependent color model; the color you get depends on the device. This also means that a particular color represented as say RGB 87, 146, 116 on one monitor might have to be specified as RGB 98, 123, 104 on another to produce the *same* color.

G.3.2 Device-independent color

A full physical description of a color would require specifying the exact spectral power distribution of the light source. Fortunately, the human eye and brain are not so sensitive as to require exact reproduction of a spectrum. Mathematical, device-independent color models exist that describe fairly well how a particular color will be seen by humans. The most important device-independent color model, to which all others can be related, was developed by the International Commission on Illumination in 1931 (CIE-1931, in French) and is called "CIE XYZ" or simply "XYZ".

In XYZ, X is the sum of a weighted power distribution over the whole visible spectrum. So are Y and Z, each with different weights. Thus any arbitrary spectral power distribution is condensed down to just three floating-point numbers. The weights were derived from color matching experiments done on human subjects in the 1920s. CIE XYZ has been an International Standard since 1931, and it has a number of useful properties:

(1) Two colors with the same XYZ values will look the same to humans

(2) Two colors with different XYZ values will not look the same

(3) The Y value represents all the brightness information (luminance)

(4) The XYZ color of any object can be objectively measured

Color models based on XYZ have been used for many years by people who need accurate control of color i.e., lighting engineers for film and TV, paint and dyestuffs manufacturers, and so on. They are thus proven in industrial use. Accurate, device-independent color started to spread from high-end, specialized areas into the mainstream during the late 1980s and early 1990s, and CDB takes notice of that trend.

G.4 Calibrated, Device-Dependent Color

Traditionally, image file formats have used uncalibrated, device-dependent color. If the precise details of the original display device are known, it becomes possible to convert the device-dependent colors of a particular image to device-independent ones. Making simplifying assumptions, such as working with CRTs (which are much easier than printers), all we need to know are the XYZ values of each primary color and the CRT exponent.

So why does not the CDB specification store images in XYZ instead of RGB? Well, two reasons. First, storing images in XYZ would require more bits of precision, which would make the files bigger. Second, all programs would have to convert the image data before viewing it. But more importantly, whether calibrated or not, all variants of RGB are close enough that undemanding viewers can get by with simply displaying the data without color correction. By storing calibrated RGB, the CDB specification retains compatibility with existing database tools and software programs that expect RGB data, yet provides enough information for conversion to XYZ in applications that need precise colors. Thus, we get the best of both worlds.

G.5 What are chromaticity and luminance?

Chromaticity is an objective measurement of the color of an object, leaving aside the brightness information. Chromaticity uses two parameters x and y, which are readily calculated from XYZ:

$$x = X / (X + Y + Z)$$
$$y = Y / (X + Y + Z) \qquad \text{(eq. G-3)}$$

XYZ colors having the same chromaticity values will appear to have the same hue but can vary in absolute brightness. Notice that x,y are dimensionless ratios, so they have the same values no matter what units we've used for X,Y,Z.

The Y value of an XYZ color is directly proportional to its absolute brightness and is called the luminance of the color. We can describe a color either by XYZ coordinates or by chromaticity x,y plus luminance Y. The XYZ form has the advantage that it is linearly related to RGB intensities.

G.6 How are computer monitor colors described?

The “white point” of a display device is the chromaticity x,y of the monitor's nominal white, that is, the color produced when R = G = B = maximum.

It's customary to specify CRT monitor colors by giving the chromaticities of the individual phosphors R, G, and B, plus the white point. The white point allows one to infer the relative brightness of the three phosphors, which isn't determined by their chromaticities alone.

NOTE: The absolute brightness of the monitor is not specified. For computer graphics work, we generally don't care very much about absolute brightness levels. Instead of dealing with absolute XYZ values (in which X,Y,Z are expressed in physical units of radiated power, such as candelas per square meter), it is convenient to work in "relative XYZ" units, where the monitor's nominal white is taken to have a luminance (Y) of 1.0. Given this assumption, it's simple to compute XYZ coordinates for the monitor's white, red, green, and blue from their chromaticity values.

G.7 How do I convert from source_RGB to XYZ

Make a few simplifying assumptions first, like the monitor really is jet black with no input and the guns don't interfere with one another. Then, given that you know the

CIE XYZ values for each of red, green, and blue for a particular monitor, you put them into a matrix M:

$$M = \begin{bmatrix} Xr & Xg & Xb \\ Yr & Yg & Yb \\ Zr & Zg & Zb \end{bmatrix} \qquad \text{(eq. G-4)}$$

RGB intensity samples normalized to the range zero to one can be converted to XYZ by matrix multiplication.

NOTE: If you the RGB samples are gamma-encoded, the gamma encoding must be un-done.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = M \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad \text{(eq. G-5)}$$

In other words, X = Xr*R + Xg*G + Xb*B, and similarly for Y and Z. You can go the other way too:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = M^{-1} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 3.2410 & -1.5374 & -0.4986 \\ -0.9692 & 1.8760 & 0.0416 \\ 0.0556 & -0.2040 & 1.0570 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad \text{(eq. G-6)}$$

Where M^{-1}= The inverse of the matrix M used to go from XYZ-1931 color space to the CDB specification RGB color space.

In the RGB encoding process, negative sRGB tri-stimulus values, and sRGB tri-stimulus values greater than 1,00 are not retained. When encoding software cannot support this extended range, the luminance dynamic range and color gamut of RGB is limited to the tri-stimulus values between 0,0 and 1,0 by simple clipping.

According to PNG (Portable Network Graphics) Specification Version 1.0, W3C Recommendation 01-October-1996 Appendix, Color Tutorial,

(http://www.w3.org/TR/PNG-GammaAppendix):

> "The gamut of a device is the subset of visible colors that the device can display. (Note that this has nothing to do with gamma.) The gamut of an RGB device can be visualized as a polyhedron in XYZ space; the vertices correspond to the device's black, blue, red, green, magenta, cyan, yellow, and white.
>
> Different devices have different gamut (e.g. database generation workstation, simulator display systems). In other words one device may be able to display certain colors (usually highly saturated ones) that another device cannot. The gamut of a particular RGB device can be determined from its R, G, and B chromaticities and white point.

Converting image data from one device to another generally results in gamut mismatches colors that cannot be represented exactly on the destination device. The process of making the colors fit, which can range from a simple clip to elaborate nonlinear scaling transformations, is termed gamut mapping. The aim is to produce a reasonable visual representation of the original image.

Appendix H

H. Navaids Attribution

This section provides a list and description of the instance-level attribution fields held in Navigation *Dataset Instance Attribute* files. The attribute name is limited to a maximum of 10 characters.

The Logical data type in column 2 of the following tables refers to the dBASE III Logical data type. A true value is defined as one of the letters T, t, Y, and y; while the false value is defined as F, f, N, and n.

H.1 Airport

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AlterNam	String	50 chars	-		Alternate name other than the official name that can be used occasionally.
AsCoStNumb	Uint64	-	-		Associated Comms record storage number
BeacoAvail	Logical	Boolean	-		Indicates if a rotating beacon is present.
City	String	50 chars	-		Airport city name.
CivMilTyp	CivilMilitaryType	0-6	-		Airport usage type (civil, military, etc.)
ClearStatu	ClearanceStatus	0-3	-		Clearance status.
Country	CountryEntry	0-336	-	2116	Country where the airport is located.
DayliTim	Float32	+/-24	Hrs		Difference to Zulu time based on the daylight saving time.
DayTimFram	String	100 chars	-		Timeframe when daylight saving time is observed by a country.
FlipPage	String	75 chars	-		Related pages for that airport in the companion FLIP.
FuelType	String	memo	-		Fuel type available.
HydElePres	Logical	Boolean	-		Indication of the presence of a hydrographic element near the airport.
IataCode	String	6 chars	-		Airport IATA designator.

Attribute Name	Data Type	Range	Unit	Key	Description
IcaoCode	String	4 chars	-	2103	Airport ICAO area code.
Ident	String	6 chars	-	2102	Airport ICAO ident.
IfrCapab	Logical	Boolean	-		Indicates if the airport has published IFR approaches.
IslanGrou	String	50 chars	-		Airport associated with islands or group of islands.
Jasu	String	100 chars	-		Type of Jet Aircraft Starting Units (JASU) available.
LonRunLeng	Uint32	-	Ft		Length of the longest runway of the airport.
LonRunSurf	PavementType	0-3	-		Surface type of the longest runway.
MagTruIndi	MagneticTrueIndication	0-6	-		Indicates if the details and procedures are given relative to Magnetic or True North.
MagneVaria	Float32	+/-180	Deg		Magnetic variation.
MgrsPosit	String	20 chars	-		MGRS position given using the UTM or the UPS grid.
Name	String	100 chars	-		Official name.
NavIcaCod	String	4 chars	-		Recommended navaid ICAO code.
NavaiIden	String	6 chars	-		Recommended navaid ident.
Notam	NotamSystem	0-4	-		Notam service.
OilType	String	75 chars	-		Type of oil available.
OperaAgenc	String	255 chars	-		Primary operating agency.
OperaHour	OperatingHours	0-4	-		Operating hours of the airport.

Attribute Name	Data Type	Range	Unit	Key	Description
Point1	GeoCoordinate	x,y,z	-		Position (latitude, longitude, altitude) of the NavObject.
Remark	String	memo	-		Essential remarks for terminal procedures.
ServiRemar	String	Memo	-		Service remarks for airport.
SpeedLimit	Uint32	-	Kts		Speed limit in knots.
SpeLimAlti	Sint32	-	Ft		Altitude below where speed limits may be imposed
StateName	StateEntry	0-51	-		State or province where the airport is located.
SupFluTyp	String	50 chars	-		Type of available fluids/system/oxygen/nitrogen.
TerraImpac	Logical	Boolean	-		Indicates a terrain impact on the airport.
Timezone	Float32	+/-24	Hrs		Difference to Zulu time.
TransAltit	Sint32	-	Ft		Upper altitude limit for which the vertical position of an A/C is controlled by reference to altitudes (MSL).
TransLeve	Sint32	-	Ft		Lowest flight level available to use above the transition altitude.

H.2 AirRefueling

Attribute Name	Data type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AiReOpIden	String	20 chars	-	2102	Air refueling operation identifier
AltitDescr	RefuelingAltitudeDescription	0-4	-		Indicates how Altitude 1 and 2 should be used (Refuel1)
AltitDesc1	RefuelingAltitudeDescription	0-4	-		Indicates how Altitude 1 and 2 should be used (Refuel2)
AltitDesc2	RefuelingAltitudeDescription	0-4	-		Indicates how Altitude 1 and 2 should be used (Refuel3)
ApRaBeCoSe	Uint32	-	-		APN 69/134/135 radar beacon code setting
ApRaBeCoS1	Uint32	-	-		APX 78 radar beacon code setting
BackuFrequ	Uint64	-	Hz		Backup UHF frequency
ComTelNumb	String	100	-		Commercial telephone number(s) of the scheduling unit
Country	CountryEntry	0-336	-	2116	Country where the refueling track or anchor is located
Direction	RefuelingDirection	0-8	-	2122	Predominant direction of the refueling track or anchor at the point of entry
DsnTelNumb	String	100	-		Defense switched network telephone number
IcaoCode	String	4	-		ICAO code at point of entry
PrimaFrequ	Uint64	-	Hz		Primary UHF frequency

ReceiChann	Uint32	-	-		Air-to-Air Y-band tacan channel used during refueling operations
Point	GeoCoordinate	x,y,z	-		Reference Position (latitude, longitude, altitude)
RefueAltit	Sint32	-	Ft		Altitude 1 to be used with altitude description 1
RefueAlti1	Sint32	-	Ft		Altitude 2 to be used with altitude description 1
RefueAlti2	Sint32	-	Ft		Altitude 1 to be used with altitude description 2
RefueAlti3	Sint32	-	Ft		Altitude 2 to be used with altitude description 2
RefueAlti4	Sint32	-	Ft		Altitude 1 to be used with altitude description 3
RefueAlti5	Sint32	-	Ft		Altitude 2 to be used with altitude description 3
Remark	String	memo	-		Remarks are limited to essential information
SchedUni	String	130	-		General information on scheduling unit (name, area, etc.)
TankeChann	Uint32	-	-		Air-to-Air Y-band tacan channel used during refueling operations
Type	RefuelingOperationType	0-3	-		Type of refueling operation

H.3 AirRefuelingControl

Attribute Name	Data type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AiReOpIden	String	20 chars	-	2108	Air refueling operation identifier
AiReStNumb	Uint64	-	-		Associated air refueling record storage number
AiTrCoCent	String	50 chars	-		ATC controlling airspace where refueling track/anchor is located
AiTrCoCeRe	String	memo	-		Remarks pertaining to the controlling agency, frequency, frequency direction, or general information
AtcCenMult	Uint32	-	-	2115	Differentiates between different entries for the same ATC center
Country	CountryEntry	0-336	-		Country where the air traffic control center is located
Direction	RefuelingDirection	0-8	-	2122	Predominant direction of the refueling track or anchor at the point of entry
Frequency1	Uint64	-	Hz		Center frequency 1
Frequency2	Uint64	-	Hz		Center frequency 2
Frequency3	Uint64	-	Hz		Center frequency 3
Frequency4	Uint64	-	Hz		Center frequency 4
Frequency5	Uint64	-	Hz		Center frequency 5

FreDirRest	FrequencyDirectionRestriction	0-3	-		Direction in which the specified frequency applies
FreDirRes1	FrequencyDirectionRestriction	0-3	-		Direction in which the specified frequency applies
FreDirRes2	FrequencyDirectionRestriction	0-3	-		Direction in which the specified frequency applies
FreDirRes3	FrequencyDirectionRestriction	0-3	-		Direction in which the specified frequency applies
FreDirRes4	FrequencyDirectionRestriction	0-3	-		Direction in which the specified frequency applies
IcaoCode	String	4 chars	-		ICAO code
Point1	GeoCoordinate	x,y,z	-		Reference Position (longitude, latitude, altitude)
RefPoiTyp	RefuelingPointType	0-7	-		Type of refueling point

H.4 AirRefuelingFootnote

Attribute Name	Data type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AiReOpIden	String	20 chars	-	2108	Air refueling operation identifier
AiReStNumb	Uint64	-	-		Associated air refueling record storage number
Country	CountryEntry	0-336	-		Country where the refueling operation is located
Direction	RefuelingDirection	0-8	-		Predominant direction of the refueling track or anchor at the point of entry
Footnote	String	memo	-		Footnote
IcaoCode	String	4 chars	-		ICAO code
Point1	GeoCoordinate	x,y,z	-		Reference Position (longitude, latitude, altitude)

H.5 AirRefuelingPoint

Attribute Name	Data type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AiReOpIden	String	20 chars	-	2108	Air refueling operation identifier
AiReStNumb	Uint64	-	-		Associated air refueling record storage number
Bearing	Uint32	0-359	Deg		Bearing TO navaid (brg FROM navaid if DME)
CoWiNaFla	Logical	Boolean	-		Indicates if point is collocated with a navaid
Country	CountryEntry	0-336	-		Country where the refueling point is located
Direction	RefuelingDirection	0-8	-		Predominant direction of the refueling track or anchor at the point of entry
Distance	Uint32	-	Nm		Distance to navaid
IcaoCode	String	4 chars	-		ICAO code
Ident	String	6 chars	-	2102	Refueling point identifier
NavaiCount	CountryEntry	0-336	-		Navaid country
NavaiIden	String	6 chars	-		Navaid identifier
NavKeyCod	Uint32	-	-		Navaid key code
NavaidType	NavaidType	0-15	-		Navaid type
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of refueling point

SequeNumbe	Uint32	-	-	2115	Refueling point sequence number
Type	RefuelingPointType	0-7	-		Type of refueling point

H.6 AirRefuelingSegment

Attribute Name	Data type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AiReOpIden	String	20 chars	-	2108	Air refueling operation identifier
AiReStNumb	Uint64	-	-		Associated air refueling record storage number
Point2	GeoCoordinate	x,y,z	-		Arc origin position (longitude, latitude, altitude)
ArcSegDeri	ArcSegmentDerivation	0-3	-		Indicates how the arc segment is defined
Bearing1	Float32	+/-180	Deg		Bearing 1 from center coordinates or navaid
Bearing2	Float32	+/-180	Deg		Bearing 2 from center coordinates or navaid
Country	CountryEntry	0-336	-		Country where the refueling segment is located
Direction	RefuelingDirection	0-8	-		Predominant direction of the refueling track or anchor at the point of entry
IcaoCode	String	4 chars	-		ICAO code
NavaiCount	CountryEntry	0-336	-		Navaid country
NavaiIden	String	6 chars	-		Navaid identifier
NavKeyCod	Uint32		-		Navaid key code
NavaidType	NavaidType	0-15	-		Navaid type
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of refueling point

Radius1	Float32	-	Nm		Radius 1
Radius2	Float32	-	Nm		Radius 2
Point3	GeoCoordinate	x,y,z	-		Segment end position (longitude, latitude, altitude)
SegmeNumbe	Uint32	-	-	2115	Defines relative position of airspace segment
Shape	BoundaryShape	0-8	-		Type of airspace segment being plotted

H.7 Airspace Boundary

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirwaLeve	AirwayLevel	0-3	-		Airspace structure in which boundary is effective (high/low)
Class	String	2 chars	-		Airspace boundary class
ClaExcFla	Logical	Boolean	-		Flag indicating exceptions to the airspace class
ClaExcRema	String	memo	-		Provides the details of the exception in the airspace
ComCalSig	String	40 chars	-	2111	Call sign of the communications facilities
ContrAutho	String	60 chars	-		Office responsible for air traffic within airspace
Country	CountryEntry	0-336	-	2116	Country where the boundary is located
Frequency	Uint64	-	Hz		Frequency for communicating with identified facility
Frequenc1	Uint64	-	Hz		Frequency 2 used for communicating with identifed facility
IcaoCode	String	4 chars	-		ICAO code of the airspace boundary
Ident	String	6 chars	-	2102	ICAO ident of airspace boundary
LowEffAlti	Sint32	-	Ft		Lower vertical limit of the given airspace
LoEfAlRefe	AltitudeReference	0-4	-		Lower effective altitude reference
LowRvsAlti	Sint32	-	Ft		Lower vertical limit of the given RVSM airspace

Name	String	50 chars	-		Official name of the airspace boundary
Point1	GeoCoordinate	x,y,z	-		Reference Position (longitude, latitude, altitude)
ReqNavPerf	Float32	-	Nm		Required performance accuracy necessary for operation within airspace
Type	AirspaceBoundaryType	0-14	-		Airspace boundary type
UppEffAlti	Sint32	-	Ft		Upper vertical limit of the given airspace
UpEfAlRefe	AltitudeReference	0-4	-		Upper effective altitude reference
UppRvsAlti	Sint32	-	Ft		Upper vertical limit of the given RVSM airspace

H.8 AirwayRestriction

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
BloAltTo	Logical	Boolean	-		Consider restriction altitude 1 to 2 as a restricted range
BloAltTo1	Logical	Boolean	-		Consider restriction altitude 2 to 3 as a restricted range
BloAltTo2	Logical	Boolean	-		Consider restriction altitude 3 to 4 as a restricted range
BloAltTo3	Logical	Boolean	-		Consider restriction altitude 4 to 5 as a restricted range
BloAltTo4	Logical	Boolean	-		Consider restriction altitude 5 to 6 as a restricted range
BloAltTo5	Logical	Boolean	-		Consider restriction altitude 6 to 7 as a restricted range
BlockAltit	Logical	Boolean	-		Consider restriction altitude 7 as a restricted altitude
Country	CountryEntry	0-336	-	2116	Country where the start fix point is located
CruisTabl	CruiseTable	0-4	-		Cruise table indicator
EndDate	String	12 chars	-		End date
EnFiIcCod	String	4 chars	-		ICAO code of end fix point
EndFixIden	String	6 chars	-		End fix point identifier
ExcluIndic	ExclusionIndicator	0-4	-		Altitudes to be excluded
OpeEndDay	DayOfWeek	0-7	-		Time of operation end day
OpeEndDay1	DayOfWeek	0-7	-		Time of operation end day

Attribute Name	Data Type	Range	Unit	Key	Description
OpeEndDay2	DayOfWeek	0-7	-		Time of operation end day
OpeEndDay3	DayOfWeek	0-7	-		Time of operation end day
OpeStaDay	DayOfWeek	0-7	-		Time of operation start day
OpeStaDay1	DayOfWeek	0-7	-		Time of operation start day
OpeStaDay2	DayOfWeek	0-7	-		Time of operation start day
OpeStaDay3	DayOfWeek	0-7	-		Time of operation start day
OpeEndTime	String	20 chars	-		Time of operation end time
OpeEndTim1	String	20 chars	-		Time of operation end time
OpeEndTim2	String	20 chars	-		Time of operation end time
OpeEndTim3	String	20 chars	-		Time of operation end time
OpeStaTime	String	20 chars	-		Time of operation start time
OpeStaTim1	String	20 chars	-		Time of operation start time
OpeStaTim2	String	20 chars	-		Time of operation start time
OpeStaTim3	String	20 chars	-		Time of operation start time
RestrAltit	Sint32	-	Ft		Restriction altitude
RestrAlti1	Sint32	-	Ft		Restriction altitude
RestrAlti2	Sint32	-	Ft		Restriction altitude
RestrAlti3	Sint32	-	Ft		Restriction altitude

Attribute Name	Data Type	Range	Unit	Key	Description
RestrAlti4	Sint32	-	Ft		Restriction altitude
RestrAlti5	Sint32	-	Ft		Restriction altitude
RestrAlti6	Sint32	-	Ft		Restriction altitude
RestrIden	Uint32	6	-		Restriction identifier
RestrNot	String	memo	-		Restriction note
RestrTyp	RestrictionType	0-4	-		Restriction type
RouteIdent	String	6 chars	-	2102	Route identifier
StartDate	String	12 chars	-		Start date
StFiIcCod	String	4 chars	-		ICAO code of start fix point
StaFixIden	String	6 chars	-		Start fix point identifier
TimeCode	TimeCode	0-4	-		Time code
TimeIndica	TimeIndicator	0-3	-		Time indicator

H.9 Approach

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirStoNumb	Uint64	-	-		Airport storage number
Altitude1	Sint32	-	Ft		First altitude limit
AltitTyp	AltitudeType	0-4	-		Altitude 1 type
Altitude2	Sint32	-	Ft		Second altitude limit
AltitTyp1	AltitudeType	0-4	-		Altitude 2 type
AltitDescr	AltitudeDescription	0-13	-		Altitude description
ArcRadius	Float32	-	Nm		Arc radius
CenterFix	String	10 chars	-		Point which defines the center of the arc flight path
CeFiIcCod	String	4 chars	-		ICAO code of the center fix
Country	CountryEntry	0-336	-	2116	Country associated with the terminal procedure
Course	Float32	+/-180	Deg		Outbound course from waypoint in fix ident
FixDetails	FixDetails	0-9	-		Fix details
FixFunctio	FixFunction	0-7	-		Fix function
FixIcaCod	String	4 chars	-		ICAO code of the fix point

Attribute Name	Data Type	Range	Unit	Key	Description
FixIdent	String	10 chars	-		Fix identifier
FlyOveTyp	FlyOverType	0-4	-		Fly over type
MagCouIndi	MagneticTrueIndication	0-6	-		Indicates if the course provided is magnetic course
NavaiCount	CountryEntry	0-336	-		Country where recommended navaid 1 is located
Point2	GeoCoordinate	x,y,z	-		Navaid 1 DME position (longitude, latitude, altitude)
NavKeyCod	Uint32	-	-		Distinguish between navaid of same type with same ident in same country
NavMagVari	Float32	+/-180	Deg		Recommended navaid 1 magnetic variation
Point3	GeoCoordinate	x,y,z	-		Navaid 1 position (longitude, latitude, altitude)
NavaiTyp	SegmentNavaidType	0-13	-		Recommended navaid 1 type
NavaiCoun1	CountryEntry	0-336	-		Country where recommended navaid 2 is located
Point4	GeoCoordinate	x,y,z	-		Navaid 2 DME position (longitude, latitude, altitude)
NavKeyCod1	Uint32	-	-		Distinguish between navaid of same type with same ident in same country
NavMagVar1	Float32	+/-180	Deg		Recommended navaid 2 magnetic variation
Point5	GeoCoordinate	x,y,z	-		Navaid 2 position (longitude, latitude, altitude)
NavaiTyp1	SegmentNavaidType	0-13	-		Recommended navaid 2 type
PathTermin	PathTermination	0-23	-		Path and Termination
ReNaIcCod	String	4 chars	-		ICAO code of the recommended navaid 1

Attribute Name	Data Type	Range	Unit	Key	Description
RecNavIden	String	10 chars	-		Recommended navaid identifier 1
RecNavIde1	String	10 chars	-		Recommended navaid identifier 2
ReqNavPerf	Float32	-	Nm		Required navigation performance
RouteDista	Float32	-	Nm		Distance in nautical miles from waypoint in fix ident
RouteType	RouteType	0-4	-		Termination Procedure Type
SpeAirCate	AircraftCategory	0-4	-		Aircraft category that speed limit 1 applies to
SpeedAltit	Sint32	-	Ft		Altitude where speed limit 1 applies
SpeedLimit	Uint32	-	Kts		Speed limit 1
SpeAirCat1	AircraftCategory	0-4	-		Aircraft category that speed limit 2 applies to
SpeedAlti1	Sint32	-	Ft		Altitude where speed limit 2 applies
SpeedLimi1	Uint32	-	Kts		Speed limit 2
SuTeDaStNu	Uint64	-	-		Storage number of associated Supplemental Terminal Data record
ThrCroHeig	Uint32	-	Ft		Threshold crossing height
TransAltit	Sint32	-	Ft		Transition altitude
TurnDirect	TurnDirection	0-3	-		Turn direction
TurDirVali	Logical	Boolean	-		Turn direction valid
WaypoCount	CountryEntry	0-336	-		Waypoint country

Attribute Name	Data Type	Range	Unit	Key	Description
WaypoDescr	WaypointDescription	0-15	-		Waypoint description
WaypoDista	Float32	-	Nm		Nautical miles between fix point and recommended navaid 1 (RHO)
WaypoDist1	Float32	-	Nm		Nautical miles between fix point and recommended navaid 2
WayMagBear	Float32	+/-180	Deg		Magnetic bearing between fix point and recommended navaid 1 (THETA)
WayMagBea1	Float32	+/-180	Deg		Magnetic bearing between fix point and recommended navaid 2
WayMagVari	Float32	+/-180	Deg		Waypoint magnetic variation
Point1	GeoCoordinate	x,y,z	-		Waypoint position (longitude, latitude, altitude)
AirIcaCod	String	4 chars	-		ICAO code of the associated airport
AirpoIden	String	6 chars	-	2102	Identifier of the associated airport
AppRouTyp	ApproachRouteType	0-39	-		Approach route type
GpsFmsIndi	GpsFmsIndicator	0-6	-		Authorized system used for procedure
Ident	String	6 chars	-	2108	SID/STAR/Approach identifier
MultiCod	String	2 chars	-		Multiple records having same center fix
MultiIndic	String	10 chars	-		Multiple records having same transition fix
RouteQuali	RouteQualifier1	0-9	-		Approach route qualifier 1
RouteQual1	RouteQualifier2	0-6	-		Approach route qualifier 2
SequeNumbe	Uint32	-	-		Sequence number

Attribute Name	Data Type	Range	Unit	Key	Description
TransIden	String	10 chars	-		Transition identifier
VertiAngl	Float32	+/-180	Deg		Descent angle for the procedure

H.10 Arresting Gear

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirIcaIden	String	6 chars	-	2108	ICAO identifier of the associated airport
AirpoIden	String	10 chars	-	2102	DAFIF identifier of the associated airport
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of airport
AirStoNumb	Uint64	-	-		Storage number of the associated airport
Country	CountryEntry	0-336	-	2116	Country in which the airport is located
DisFroRefe	Uint32	-	Ft	2114	Distance from the reference given in location reference
LocatRefer	LocationReference	0-3	-	2122	Reference for location of arresting gear
RunwaIden	String	6 chars	-	2111	Runway identifier
Type	String	80 chars	-	2107	Arresting gear type

H.11 Comms

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
24HouAvail	Logical	Boolean	-		24 hour availability of comms frequency flag
AirIcaCod	String	4 chars	-		ICAO code of the associated airport
AirpoIden	String	6 chars	-	2102	Identifier of the associated airport
AirStoNumb	Uint64	-	-		Storage number of the associated airport
AltitDescr	AltitudeDescription	0-13	-		Altitude description
AntenPatte	String	30 chars	-		Antenna Pattern Description
AreaCode	String	12 chars	-		Area code for telephone numbers
CallSign	String	50 chars	-		Name of facility being called
CellNetwor	String	30 chars	-		Cellular network information
CommsAltit	Sint32	-	Ft		Communications altitude limit 1
CommsAlti1	Sint32	-	Ft		Communications altitude limit 2
CommsDetai	CommsDetails	0-7	-		Communications details
CommsDista	Uint32	-	Nm		Communications distance
CommsEncry	CommsEncryption	0-1	-		Communications encryption status/mode

Attribute Name	Data Type	Range	Unit	Key	Description
ComFliTyp	CommsFlightType	0-4	-		FIR/UIR address to supplement identifier
CommsType	CommsType	0-58	-	2107	Communications type
Country	CountryEntry	0-336	-	2116	Country where the communications information is applicable
DistaDescr	DistanceDescription	0-2	-		Comms distance description
Encrypted	Logical	Boolean	-		Encrypted
FirUirIden	String	6 chars	-	2108	FIR/UIR identifier
FirUirIndi	FirUirType	0-3	-		FIR/UIR indicator
Frequency	Uint64	-	Hz	2104	Communications frequency
FrequTyp	FrequencyType	0-7	-		Communications frequency type
GuardTrans	GuardTransmit	0-3	-		Communications transmit/receive flag
MagneVaria	Float32	+/-180	Deg		Magnetic variation
Modulation	Modulation	0-2	-		Signal modulation
MonitFrequ	MonitoredFrequency	0-6	-		Monitored emergency frequencies
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of communications antenna
RadarCapab	Logical	Boolean	-		Radar capability flag
ReceiSensi	Float64	-	Watt		Receiver sensitivity
Remark	String	memo	-		Remarks associated with Comms station

Attribute Name	Data Type	Range	Unit	Key	Description
ReAiStNumb	Uint64	-	-		Storage number of associated remote airport facility
ReFaIcCod	String	4 chars	-		ICAO code of associated remote facility
RemFacIden	String	6 chars	-		Identifier of associated remote facility
RemFacTyp	FacilityRecordType	0-4	-		Associated remote facility type
RemoteName	String	50 chars	-	2120	Name of associated remote facility
ReNaStNumb	Uint64	-	-		Storage number of associated remote navaid
RetraAvail	Logical	Boolean	-		Retransmission available
RetraFrequ	Uint64	-	Hz		Retransmission frequency
Sector	String	100 chars	-		Area in which frequency is effective
SeAiStNumb	Uint64	-	-		Storage number of sector airport facility
SecEndBear	Uint32	0-359	Deg		Sector end bearing
SeFaIcCod	String	4 chars	-		ICAO code of sector facility
SecFacIden	String	6 chars	-		Identifier of sector facility
SecFacTyp	FacilityRecordType	0-4	-		Sector facility type
SeNaStNumb	Uint64	-	-		Storage number of sector navaid
SecStaBear	Uint32	0-359	Deg		Sector start bearing
ServiIndic	ServiceIndicator	0-10	-		Communications service indicator
SignaEmiss	SignalEmission	0-7	-		Signal emission

Attribute Name	Data Type	Range	Unit	Key	Description
SpeOpeHour	String	100 chars	-		Hours of operation different from airport/heliport
TelepNumbe	String	20 chars	-		Telephone number
TransPowe	Float64	-	Watt		Transmission power
VoiceMessa	String	30 chars	-		Voice message

H.12 Controlled Airspace

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AiBoStNumb	Uint64	-	-		Associated AirspaceBoundary record storage number
AirspCente	String	6 chars	-	2102	Ident for airspace 'center'
AirspClass	String	2 chars	-		Airspace classification (one character)
AirspTyp	AirspaceType	0-18	-	2107	Controlled airspace type
AirTypChar	String	2 chars	-	2122	Controlled airspace type character read directly from data file
ArcBearing	Float32	+/-180	Deg		Arc bearing
ArcDistanc	Float32	-	Nm		Arc distance

Attribute Name	Data Type	Range	Unit	Key	Description
ArcDistan1	Float32	-	Nm		Arc distance (radius of arc from center point)
Point3	GeoCoordinate	x,y,z	-		Arc origin position (longitude, latitude, altitude)
ArcSegDeri	ArcSegmentDerivation	0-3	-		Indicates how the arc segment is defined
Bearing1	Float32	+/-180	Deg		True bearing from arc origin or navaid
Bearing2	Float32	+/-180	Deg		True bearing from arc origin or navaid
BoundEn	Logical	Boolean	-		End of boundary description - return to origin point
BoundShap	BoundaryShape	0-8	-		Boundary shape type
Country	CountryEntry	0-336	-	2116	Country where airspace is located
Country1	CountryEntry	0-336	-		Country through which the boundary passes
Country2	CountryEntry	0-336	-		Country through which the boundary passes
Country3	CountryEntry	0-336	-		Country through which the boundary passes
Country4	CountryEntry	0-336	-		Country through which the boundary passes
Country5	CountryEntry	0-336	-		Country through which the boundary passes
IcaoCode	String	4 chars	-		ICAO code for the airspace
Level	AirwayLevel	0-3	-		Type of airway (high, low, or either)
LowerLimit	Sint32	-	Ft		Lower limit
LoLiAlRefe	AltitudeReference	0-4	-		Altitude reference
MultiCod	String	2 chars	-	2118	Differentiate between airspaces with same designator

Attribute Name	Data Type	Range	Unit	Key	Description
Name	String	50 chars	-		Controlled airspace name
NavaiCount	CountryEntry	0-336	-		Country in which navaid is located
NavaiIden	String	6 chars	-		Navaid identifier
NavKeyCod	Uint32	-	-		Distinguish between same type navaid with same ident and country
NavaidType	NavaidType	0-15	-		Navaid type
Notam	Logical	Boolean	-		Active times by NOTAM
Point2	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude)
Point1	GeoCoordinate	x,y,z	-		Reference Position (longitude, latitude, altitude)
ReqNavPerf	Float32	-	Nm		Required navigation performance
SequeNumbe	Uint32	-	-		Sequence number
TimeCode	PrimaryTimeCode	0-4	-		Time codes for primary records
UpperLimit	Sint32	-	Ft		Upper limit
UpLiAlRefe	AltitudeReference	0-4	-		Reference for upper limit altitude

H.13 Enroute Airway

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.

Attribute Name	Data Type	Range	Unit	Key	Description
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirwaLeve	AirwayLevel	0-3	-		Airway level
AirwaRestr	Logical	Boolean	-		Airway restriction exists
AtcComFla	Logical	Boolean	-		ATC compulsory waypoint flag
BoundCod	BoundaryCode	0-10	-		Boundary code
Country	String	95 chars	-		List of countries through which the ATS route segment passes
CrLeNoStFl	Logical	Boolean	-		IFR cruising levels are not in agreement with appropriate diagrams (FLIP)
CruisTabl	CruiseTable	0-4	-		Cruise table indicator
Direction	Direction	0-2	-		Predominant direction of ATS route
DirecRestr	DirectionRestriction	0-3	-		Direction restriction
EnAiRoTyp	EnrouteAirwayRouteType	0-7	-		Enroute airway route type
FixCountry	CountryEntry	0-336	-	2116	Country where the fix point is located
FixDetails	FixDetails	0-9	-		Fix details
FixFunctio	FixFunction	0-7	-		Fix function
FixIcaCod	String	4 chars	-		ICAO code of fix point
FixIdent	String	6 chars	-		Fix identifier
FixNavTyp	NavaidType	0-15	-		Fix type
FixRecTyp	FixRecordType	0-8	-		Fix point record type

Attribute Name	Data Type	Range	Unit	Key	Description
FixStoNumb	Uint64	-	-		Fix point storage number
FixTurRadi	Float32	-	Nm		Fix turn radius 1
FixTurRad1	Float32	-	Nm		Fix turn radius 2
FlyOveTyp	FlyOverType	0-4	-		Fly over type
FrequClas	FrequencyClass	0-2	-		Frequency class of ATS route (UHF/VHF or LF/MF)
IcaoCode	String	4 chars	-		ICAO Code
InbouCours	Float32	+/-180	Deg		Inbound course to waypoint in fix ident
InbCouRefe	MagneticTrueIndication	0-6	-		Inbound course reference
MaximAltit	Sint32	-	Ft		Maximum altitude for segment
MaxFliAlti	Sint32	-	Ft		Maximum altitude for airway
MinimAltit	Sint32	-	Ft		Altitude limit in direction flight coded for segment
MinimAlti1	Sint32	-	Ft		Segment altitude limit for opposite of coded direction of flight
MinFliAlti	Sint32	-	Ft		Minimum altitude limit for airway
OutboCours	Float32	+/-180	Deg		Outbound course from waypoint in fix ident
OutCouRefe	MagneticTrueIndication	0-6	-		Outbound course reference
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of waypoint
ReNaIcCod	String	4 chars	-		ICAO code of recommended navaid
RecNavIden	String	6	-		Recommended navaid identifier

Attribute Name	Data Type	Range	Unit	Key	Description
Remark	String	memo	-		Essential information related to ATS route
ReqNavPerf	Float32	-	Nm		Required navigation performance
RouteDista	Float32	-	Nm		Distance in nautical miles from waypoint in fix ident
RouteIdent	String	8 chars	-	2102	Route identifier
RouSegTyp	AtsRouteSegmentType	0-2	-		ATS route segment type
RouteStatu	RouteStatus	0-5	-		ATS route status
RvsmFlag	Logical	Boolean	-		Reduced vertical separation minima
SequeNumbe	Uint32	-	-		Sequence number
StateName	StateEntry	0-51	-		State through which ATS route passes
TransRadiu	Float32	-	-		Transition radius
WaypoDescr	WaypointDescription	0-15	-		Waypoint description
WaypoDista	Float32	-	Nm		Nautical miles between fix point and recommended navaid
WayMagBear	Float32	+/-180	Deg		Magnetic bearing between fix point and recommended navaid

H.14 FirUir

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AdjFirIden	String	6 chars	-	2108	Adjacent FIR ident
AdjUirIden	String	6 chars	-	2120	Adjacent UIR ident
AiBoStNumb	Uint64	-	-		Associated airspace boundary record storage number
AltitUni	AltitudeUnit	0-3	-		Unit used in specific FIR/UIR to fulfill requirement of ICAO flight plan
ArcBearing	Float32	+/-180	Deg		Arc bearing
ArcDistanc	Float32	-	Nm		Arc distance
ArcDistan1	Float32	-	Nm		Arc distance (radius of arc from center point)
Point2	GeoCoordinate	x,y,z	-		Arc origin position (longitude, latitude, altitude)
ArcSegDeri	ArcSegmentDerivation	0-3	-		Indicates how the arc segment is defined
Bearing1	Float32	+/-180	Deg		True bearing from arc origin or navaid
Bearing2	Float32	+/-180	Deg		True bearing from arc origin or navaid
BoundEn	Logical	Boolean	-		End of boundary description - return to origin point
BoundShap	BoundaryShape	0-8	-		Boundary shape type
Country1	CountryEntry	0-336	-		Country through which the boundary passes

Attribute Name	Data Type	Range	Unit	Key	Description
Country2	CountryEntry	0-336	-		Country through which the boundary passes
Country3	CountryEntry	0-336	-		Country through which the boundary passes
Country4	CountryEntry	0-336	-		Country through which the boundary passes
Country5	CountryEntry	0-336	-		Country through which the boundary passes
CruisTabl	CruiseTable	0-4	-		Cruise table applicable
EntRepRequ	Logical	Boolean	-		Entry report required for FIR/UIR
FirUppLimi	Sint32		Ft		FIR Upper Limit
FlightType	CommsFlightType	0-4	-	2122	Type of airway (high, low, or either)
IcaoCode	String	4 chars	-		FIR/UIR ICAO code
Ident	String	6 chars	-	2102	FIR/UIR Ident
Name	String	50 chars	-		Fir/Uir name
NavaiCount	CountryEntry	0-336	-		Country in which navaid is located
NavaiIden	String	6 chars	-		Navaid identifier
NavKeyCod	Uint32	-	-		Distinguish between same type navaid with same ident and country
NavaidType	NavaidType	0-15	-		Navaid type
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude)
SequeNumbe	Uint32	-	-		Sequence number
SpeedUnit	SpeedUnit	0-3	-		Unit used in specific FIR/UIR to fulfill requirement of ICAO flight plan

Attribute Name	Data Type	Range	Unit	Key	Description
Type	FirUirType	0-3	-	2107	FIR/UIR type
UirLowLimi	Sint32	-	Ft		UIR Lower limit
UirUppLimi	Sint32	-	Ft		Upper limit

H.15 Gate

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
Airline	String	50 chars	-		Airline assigned to gate
AirIcaCod	String	4 chars	-		ICAO code of the associated airport
AirpoIden	String	6 chars	-	2102	Identifier of the associated airport
AirStoNumb	Uint64	-	-		Storage number of the associated airport
Country	CountryEntry	0-336	-	2116	Country where the gate is located
Ident	String	6 chars	-	2108	Gate identifier
Name	String	50 chars	-		Name commonly applied to the gate
Orientatio	Float32	+/-180	Deg		Orientation of gate (bearing)
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of gate

H.16 GLS

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirpoIden	String	6 chars	-	2102	Ident of the associated airport
AirStoNumb	Uint64	-	-		Storage number of the associated airport
ApproSlop	Float32	+/-180	Deg		Glideslope angle of the GLS approach
Bearing	Float32	+/-180	Deg		Localizer bearing of GLS approach
Category	LandingAidCategory	0-9	-		Category/Class of the GLS
Channel	String	10 chars	-		Channel decoded to identify frequency of differential GLS ground station and approach info sent by diff. GLS ground station
Country	CountryEntry	0-336	-	2116	Country where the GLS is located
IcaoCode	String	4 chars	-	2103	ICAO code
Ident	String	6 chars	-	2108	GLS reference path identifier
LocatIden	String	10 chars	-		Airport or heliport ICAO location identifier code where transmitter is installed
MagneVaria	Float32	+/-180	Deg		Magnetic variation
Point1	GeoCoordinate	x,y,z	-		Station position (longitude, latitude, altitude)
RunwaIden	String	6 chars	-		Ident of the associated runway

RunStoNumb	Uint64	-	-		Storage number of the associated runway
SerVolRadi	Uint32	-	Nm		Radius of service volume around transmitter
StatiTyp	GlsStationType	0-2	-		Type of differential ground station (eg: LAAS/GLS or SCAT-1)
TdmaSlot	String	30 chars	-		Time division multiple access (TDMA) slot in which ground station transmits related approach

H.17 Helipad

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AircrTyp	String	10 chars	-		Aircraft type known to have used helipad in last 5 years.
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of the helipad approach end.
Bearing	Float32	+/-180	Deg		Magnetic bearing.
Country	CountryEntry	0-336	-	2116	Helipad country.
Point2	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of the displaced threshold (latitude, longitude, elevation).
HelipClose	Logical	Boolean	-		Indicates if the helipad is closed or unusable.
HelIcaCod	String	4 chars	-	2103	Associated Heliport ICAO code.
HelipIden	String	6 chars	-	2108	Associated Heliport identifier.
HelStoNumb	Uint64	-	-		Associated Heliport storage number.
Ident	String	6 chars	-	2102	Helipad identifier.
Length	Uint32	-	Ft		Helipad length.
LightSyste	LightingSystem	0-64	-		Lighting system 1.
LightSyst1	LightingSystem	0-64	-		Lighting system 2.
LightSyst2	LightingSystem	0-64	-		Lighting system 3.

PadShape	PadShape	0-2	-		Shape of helipad (circular or rectangular).
SequeNumbe	Uint32	-	-	2115	Sequence number to differentiate helipads at same heliport.
Slope	Float32	-	%		Helipad gradient
Point3	GeoCoordinate	x,y,z	-		Position (longitude, latitude, elevation) of the helipad stop end.
StopwLengt	Uint32	-	Ft		Length of the area beyond the takeoff helipad.
StoSurTyp	RunwaySurfaceType	0-21	-		Stopway surface type.
SurfaTyp	RunwaySurfaceType	0-21	-		Helipad surface type.
TakeoDista	Uint32	-	Ft		Takeoff distance available.
TrueBearin	Float32	+/-180	Deg		Helipad true bearing.
TruNorRefe	Logical	Boolean	-		True North reference flag.
Width	Uint32	10	Ft		Helipad width.

H.18 Heliport

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AlterNam	String	50 chars	-		Alternate name other than the official name that can be used occasionally.
AsCoStNumb	Uint64	-	-		Associated Comms record storage number
BeacoAvail	Logical	Boolean	-		Indicates if a rotating beacon is present.
City	String	50 chars	-		Heliport city name.
CivMilTyp	CivilMilitaryType	0-6	-		Heliport usage type (civil, military, etc.).
ClearStatu	ClearanceStatus	0-3	-		Clearance status.
Country	CountryEntry	0-336	-	2116	Country where the heliport is located.
DayliTim	Float32	+/-24	Hrs		Difference to Zulu time based on the daylight saving time.
DayTimFram	String	100 chars	-		Timeframe when daylight saving time is observed by a country.
FlipPage	String	75 chars	-		Related pages for that heliport in the companion FLIP.
FuelType	String	memo	-		Fuel type available.
HydElePres	Logical	Boolean	-		Indication of the presence of an hydrographic element near the heliport.

Attribute Name	Data Type	Range	Unit	Key	Description
IataCode	String	6 chars	-	2106	Heliport IATA designator.
IcaoCode	String	4 chars	-	2103	Heliport ICAO area code.
Ident	String	6 chars	-	2102	Heliport ICAO ident.
IfrCapabil	Logical	Boolean	-		Indicates if the heliport has published IFR approaches.
IslanGrou	String	50 chars	-		Heliport associated with islands or group of islands.
Jasu	String	100 chars	-		Type of Jet Aircraft Starting Units (JASU) available.
MagneVaria	Float32	+/-180	Deg		Magnetic variation.
MagTruIndi	MagneticTrueIndication	0-6	-		Indicates if the details and procedures are given relative to Magnetic or True North.
MgrsPositi	String	20 chars	-		MGRS position given using the UTM or the UPS grid.
Name	String	100 chars	-		Official name.
NavIcaCod	String	4 chars	-		Recommended navaid ICAO code.
Navailden	String	6 chars	-		Recommended navaid ident.
Notam	NotamSystem	0-4	-		Notam service.
OilType	String	75 chars	-		Type of oil available.
OperaHour	OperatingHours	0-4	-		Operating hours of the heliport.
PadDimensi	Uint32	-	Ft		Pad dimension.

Attribute Name	Data Type	Range	Unit	Key	Description
PadDimens1	Uint32	-	Ft		Pad dimension.
PadIdent	String	6 chars	-	2108	Helipad identifier.
PadShape	PadShape	0-2	-		Pad shape.
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of the NavObject.
Remark	String	memo	-		Essential remarks for terminal procedures.
ServiRemar	String	memo	-		Service remarks for airport.
SpeedLimit	Uint32	-	Kts		Speed limit in knots.
SpeLimAlti	Sint32	-	Ft		Altitude below where speed limits may be imposed
StateName	StateEntry	0-51	-		State or province where the heliport is located.
SupFluTyp	String	50 chars	-		Type of available fluids/system/oxygen/nitrogen.
TerraImpac	Logical	Boolean	-		Indicates a terrain impact on the heliport.
Timezone	Float32	+/-24	Hrs		Difference to Zulu time.
TransAltit	Sint32	-	Ft		Upper altitude limit for which the vertical position of an A/C is controlled by reference to altitudes (MSL).
TransLeve	Sint32	-	Ft		Lowest flight level available to use above the transition altitude.

H.19 HoldingPattern

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirIcaCod	String	4 chars	-		ICAO code of the associated airport
AirpoIden	String	6 chars	-		Identifier of the associated airport
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of airport
AirStoNumb	Uint64	-	-		Storage number of the associated airport
ArcRadius	Float32	-	Nm		Turning radius, inbound to outbound leg, for RNP Holding
Country	CountryEntry	0-336	-	2116	Country where the holding pattern applies
DupliIden	String	6 chars	-	2108	Duplicate identifier
FixCountry	CountryEntry	0-336	-		Country where the fix point is located
FixIcaCod	String	4 chars	-		Fix ICAO Code
FixIdent	String	6 chars	-	2102	Fix identifier
FixRecTyp	FixRecordType	0-8	-	2107	Record type of fix point
FixStoNumb	Uint64	-	-		Fix point storage number
HoldiCours	Float32	+/-180	Deg		Inbound holding course
HoPaTuDire	PathTurnDirection	0-2	-		Holding pattern turn direction

HoldiSpee	Uint32	-	Kts		Holding pattern maximum speed in knots
LegLength	Float32	-	Nm		Leg length in nautical miles
LegTime	Float32	-	Min		Leg time in minutes
MagneCours	MagneticTrueIndication	0-6	-		Indicates if magnetic course
MaximAltit	Sint32	-	Ft		Maximum altitude
MinimAltit	Sint32	-	Ft		Minimum altitude
Name	String	50 chars	-		Name commonly applied to the holding pattern
NavaiCount	CountryEntry	0-336	-		Country of navaid collocated with waypoint
NavaiIden	String	6 chars	-		Identifier of navaid collocated with waypoint
NavKeyCod	Uint32	-	-		Key code of navaid collocated with waypoint
NavaidType	NavaidType	0-15	-		Type of navaid collocated with waypoint
ReqNavPerf	Float32	-	Nm		Required navigation performance
TrackDescr	TrackDescription	0-3	-		Defines track geometry for single terminal segment record
Type	HoldingPatternType	0-7	-		Type of holding pattern

H.20 Ils

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirIcaCod	String	4 chars	-	2103	ICAO code of the associated airport.
AirpoIden	String	6 chars	-	2102	Ident of the associated airport.
AirStoNumb	Uint64	-	-		Storage number of the associated airport.
AppRouIden	String	6 chars	-		Ident of the associated approach route 1.
ApRoStNumb	Uint64	-	-		Storage number of the associated approach route 1.
AppRouIde1	String	6 chars	-		Ident of the associated approach route 2.
ApRoStNum1	Uint64	-	-		Storage number of the associated approach route 2.
AppRouIde2	String	6 chars	-		Ident of the associated approach route 3.
ApRoStNum2	Uint64	-	-		Storage number of the associated approach route 3.
AppRouIde3	String	6 chars	-		Ident of the associated approach route 4.
ApRoStNum3	Uint64	-	-		Storage number of the associated approach route 4.
AppRouIde4	String	6 chars	-		Ident of the associated approach route 5.
ApRoStNum4	Uint64	-	-		Storage number of the associated approach route 5.
BacCouAvai	IlsBackCourse	0-3	-		Back course availability information.

Attribute Name	Data Type	Range	Unit	Key	Description
Bearing	Float32	+/-180	Deg		Localizer magnetic bearing.
BeariRefer	MagneticTrueIndication	0-6	-		Bearing reference.
Category	LandingAidCategory	0-9	-		Category/class of the ILS.
Country	CountryEntry	0-336	-	2116	Country where the ILS is located.
Declinatio	Float32	+/-180	Deg		Station declination.
DecliRefer	MagneticTrueIndication	0-6	-		Declination angle reference.
FalGliFla	Logical	Boolean	-		False glidepath flag
FalLocFla	Logical	Boolean	-		False localizer flag
GlideAngl	Float32	+/-180	Deg		Glideslope angle.
GlideBeamw	Float32	+/-180	Deg		Glideslope beamwidth.
GlideFrequ	Uint64		Hz		ILS glideslope frequency.
GliMagVari	Float32	+/-180	Deg		ILS glideslope magnetic variation.
Point3	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of the glideslope emitter.
GliXOffse	Sint32	-	Ft		Glideslope X offset.
GliYOffse	Sint32	-	Ft		Glideslope Y offset.
Ident	String	6 chars	-		Localizer ICAO ident.
LocalBeamw	Float32	+/-180	Deg		Localizer beamwidth.
LocalFrequ	Uint64	-	Hz	2104	ILS localizer frequency.

Attribute Name	Data Type	Range	Unit	Key	Description
LocMagVari	Float32	+/-180	Deg		ILS localizer magnetic variation.
Point2	GeoCoordinate	x,y,z	-		Localizer position (longitude, latitude, altitude).
LocXOffse	Sint32	-	Ft		Localizer X offset.
LocYOffse	Sint32	-	Ft		Localizer Y offset.
Name	String	50 chars	-		Official name of the localizer.
NavStoNumb	Uint64	-	-		Storage number of the associated navaid.
Point1	GeoCoordinate	x,y,z	-		Reference Position (longitude, latitude, altitude)
Runwalden	String	6 chars	-	2111	Ident of the associated runway.
RunStoNumb	Uint64	-	-		Storage number of the associated runway.
SynchTyp	SynchronisationType	0-2	-		Synchronization type.
ThrCroHeig	Uint32	-	Ft		Height above the landing threshold on a normal glidepath.
TrueBearin	Float32	+/-180	Deg		Localizer true bearing.

H.21 Marker

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirpoIden	String	6 chars	-	2108	Ident of the associated airport/heliport
AirStoNumb	Uint64	-	-		Storage number of the associated airport/heliport
AssocNavai	AssociatedNavaid	0-2	-		Associated navaid information
Channel	String	6 chars	-		Navaid channel.
Country	CountryEntry	0-336	-	2116	Country where the marker is located
Frequency	Uint64		Hz		Frequency
HighLow	MarkerPower	0-2	-		Marker power
IcaoCode	String	4 chars	-	2103	Marker ICAO area code
Ident	String	6 chars	-	2102	Marker ident
IlsBearing	Float32	+/-180	Deg		Bearing of the ILS localizer
IlsBeaRefe	MagneticTrueIndication	0-6	-		Reference for the ILS bearing
LocalIden	String	6 chars	-		Associated localizer ident
LocStoNumb	Uint64	-	-		Associated localizer storage number
Location	Float32	-	Nm		Location from the approach end of the runway

LocatCollo	Logical	Boolean	-		Locator collocation flag
LocatIden	String	6 chars	-		Associated locator ident
LocStoNum1	Uint64	-	-		Associated locator storage number
MagneVaria	Float32	+/-180	Deg		Magnetic variation
MinAxiBear	Float32	+/-180	Deg		True bearing of the marker minor axis
MorseCode	String	3 chars	-		Corresponding letters of the Morse code
Name	String	50 chars	-		Marker official name
NavaiCount	CountryEntry	0-336	-		Navaid country.
NavaiFrequ	Uint64	-	Hz		Frequency
NavKeyCod	Uint32	-	-		Navaid key code.
NavaidType	NavaidType	0-15	-		Navaid type.
Point1	GeoCoordinate	x,y,z	-		Marker position (longitude, latitude, altitude)
RunwaIden	String	6 chars	-	2111	Ident of the associated runway
RunStoNumb	Uint64	-	-		Storage number of the associated runway
Type	MarkerType	0-10	-	2107	Marker type

H.22 MilitaryTrainingRoute

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
Country	CountryEntry	0-336	-	2116	Country where the military training route originates
EffecTime	String	100 chars	-		Hours, days and/or dates that military training route is in effect
IcaoCode	String	4 chars	-		ICAO code of air traffic controlling authority where route originates
Ident	String	10 chars	-	2102	Designation of the military training route
OriMilUni	String	100 chars	-		Military unit designated as the originating activity
Point1	GeoCoordinate	x,y,z	-		Reference Position (longitude, latitude, altitude)
Remark	String	Memo	-		Remarks are limited to terrain following ops, special operating proc., flight service stations (100nm radius) & SR remarks
SchMilUni	String	100 chars	-		Military unit responsible for scheduling training route flights
Type	MilitaryRouteType	0-3	-		Type of military training route

H.23 MilitaryTrainingRouteAirspace

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
ActPoIden	String	4 chars	-		Ident of the action point within the military training route
MiTrRoIden	String	10 chars	-	2102	Military training route identifier
MiTrRoStNu	Uint64	-	-		Associated military training route storage number
MTROSNumbe	Uint64	-	-		Associated military training route overlay storage number
NeAcPoIden	String	4 chars	-		Ident of the next action point within the military training route
Point1	GeoCoordinate	x,y,z	-		Reference Position (longitude, latitude, altitude)
Sector	String	10 chars	-		Designation for the section of the special use airspace
SegmeNumbe	Uint32	-	-	2115	Defines relative position of segment in military training route airspace
SequeNumbe	Uint32	-	-	2120	Defines order of special use airspace (SUAS) or military operations area (MOA) identifiers
SpUsAiIden	String	18 chars	-		Special use airspace or military operations area identifier
SpUsAiStNu	Uint64	-	-		Associated special use airspace storage number

H.24 MilitaryTrainingRouteDescription

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
ActPoiIden	String	4 chars	-	2108	Ident of the action point within the military training route
AddRouInfo	String	100 chars	-		Info vital to execution of military training route at a specific point to the next point
Bearing	Float32	+/-180	Deg		Bearing from DME or bearing to non-DME navaid
CoWiNaFla	Logical	Boolean	-		Point collocated with navaid flag
Country	CountryEntry	0-336	-		Country where the point is located
CrossAltit	Sint32	-	Ft		Crossing altitude 1
CroAltRefe	AltitudeReference	0-4	-		Crossing altitude 1 reference
CrossAlti1	Sint32	-	Ft		Crossing altitude 2
CroAltRef1	AltitudeReference	0-4	-		Crossing altitude 2 reference
CroAltDesc	RouteAltitudeDescription	0-5	-		Indicates how the crossing altitude(s) should be applied
Distance	Float32	-	Nm		Range from non-DME navaid or slant range from DME
EnrouAltit	Sint32	-	Ft		Enroute altitude 1
EnrAltRefe	AltitudeReference	0-4	-		Enroute altitude 1 reference
EnrouAlti1	Sint32	-	Ft		Enroute altitude 2

EnrAltRef1	AltitudeReference	0-4	-		Enroute altitude 2 reference
EnrAltDesc	RouteAltitudeDescription	0-5	-		Indicates how the enroute altitude(s) should be applied
IcaoCode	String	4 chars	-		ICAO code
MiTrRoIden	String	10 chars	-	2102	Military training route identifier
MiTrRoStNu	Uint64	-	-		Associated military training route storage number
NavaiCount	CountryEntry	0-336	-		Navaid country
NavaiIden	String	6 chars	-		Navaid identifier
NavKeyCod	Uint32	-	-		Navaid key code
NavaidType	NavaidType	0-15	-		Navaid type
NeAcPoIden	String	4 chars	-		Ident of next action point within a military training route
PointFunct	PointFunction	0-6	-		Function of the point
Point1	GeoCoordinate	x,y,z	-		Position of point (longitude, latitude, altitude)
RouWidLef	Float32	-	Nm		Route width to left of centerline to the next point
RouWidRigh	Float32	-	Nm		Route width to right of centerline to the next point
TurnDirect	PathTurnDirection	0-2	-		Specific direction in which a turn is to be made
TurnRadius	Float32	-	Nm		Turn radius around a point

H.25 MilitaryTrainingRouteOverlay

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AcPoBiSeAn	Float32	+/-180	Deg		Bi-section path angle for the next point based on next segment path (acute angle to that path)
ActPoiFunc	PointFunction	0-6	-		Function of the action point
ActPoiIden	String	4 chars	-		Ident of the action point within the military training route
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of the action point
AcPoRoWiLe	Float32	-	Nm		Route width to left of action point
AcPoRoWiRi	Float32	-	Nm		Route width to right of action point
AcPoTuDire	PathTurnDirection	0-2	-		Specific direction in which a turn is to be made
AcPoTuRadi	Float32	-	Nm		Turn radius around action point
MiTrRoIden	String	10 chars	-	2102	Military training route identifier
MiTrRoStNu	Uint64	-	-		Associated military training route storage number
NAPBSAngl	Float32	+/-180	Deg		Bi-section path angle for the next point based on next segment path (acute angle to that path)
NeAcPoFunc	PointFunction	0-6	-		Function of the next action point
NeAcPoIden	String	4 chars	-		Ident of the next action point within the military training route
Point2	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of the next action point
NAPRWLef	Float32	-	Nm		Route width to left of the next action point

NAPRWRigh	Float32	-	Nm		Route width to right of the next action point
NeAcPoTuDi	PathTurnDirection	0-2	-		Specific direction in which a turn is to be made
NeAcPoTuRa	Float32	-	Nm		Turn radius around the next action point
SegmeNumbe	Uint32	-	-	2115	Defines relative position of segment in military training route overlay

H.26 Mls

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirIcaCod	String	4 chars	-	2103	Icao code of the associated airport
AirpoIden	String	6 chars	-	2102	Ident of the associated airport
AirStoNumb	Uint64	-	-		Storage number of the associated airport
AzimuBeari	Float32	+/-180	Deg		Magnetic bearing of the MLS azimuth
AziLefAngl	Float32	+/-180	Deg		Azimuth proportional left angle
AziLefCove	Sint32	+/-180	Deg		Azimuth left coverage
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of the MLS azimuth transmitter
AziRigAngl	Float32	+/-180	Deg		Azimuth proportional right angle
AziRigCove	Sint32	+/-180	Deg		Azimuth right coverage
AziTruBear	Float32	+/-180	Deg		Azimuth true bearing in degrees
AziXOffse	Float32	-	Ft		Azimuth X offset
AziYOffse	Float32	-	Ft		Azimuth Y offset
BacAziBear	Float32	+/-180	Deg		Magnetic bearing of the MLS back azimuth
BaAzLeAngl	Float32	+/-180	Deg		Back azimuth proportional left angle

Attribute Name	Data Type	Range	Unit	Key	Description
BaAzLeCove	Sint32	+/-180	Deg		Back azimuth left coverage
Point2	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of the MLS back azimuth transmitter
BaAzRiAngl	Float32	+/-180	Deg		Back azimuth proportional right angle
BaAzRiCove	Sint32	+/-180	Deg		Back azimuth right coverage
BaAzTrBear	Float32	+/-180	Deg		Back azimuth true bearing in degrees
BaAzXOffse	Float32	-	Ft		Back azimuth X offset
BaAzYOffse	Float32	-	Ft		Back azimuth Y offset
Category	LandingAidCategory	0-9	-		Category/class of the MLS
Channel	String	6 chars	-	2110	Assigned channel
Collocatio	MlsCollocation	0-3	-		MLS collocation information
Country	CountryEntry	0-336	-	2116	Country where the MLS is located
Point3	GeoCoordinate	x,y,z	-		MLS Datum point position (longitude, latitude, altitude)
DaPoXOffse	Float32	-	Ft		Datum point X offset
DaPoYOffse	Float32	-	Ft		Datum point Y offset
EleAngSpa	Float32	+/-180	Deg		Elevation angle span
EleMinAngl	Float32	+/-180	Deg		Elevation minimum angle
EleNomAngl	Float32	+/-180	Deg		Elevation nominal angle

Attribute Name	Data Type	Range	Unit	Key	Description
Point4	GeoCoordinate	x,y,z	-		Elevation position (longitude, latitude, altitude)
EleXOffse	Float32	-	Ft		Elevation X offset
EleYOffse	Float32	-	Ft		Elevation Y offset
Frequency	Uint64	-	Hz	2104	Frequency
HigRatAppr	Logical	Boolean	-		MLS high rate approach available
Ident	String	6 chars	-		MLS ICAO ident
MagneVaria	Float32	+/-180	Deg		Magnetic variation
Name	String	50 chars	-		Official name of the MLS
RunwaIden	String	6 chars	-	2111	Ident of the associated runway
RunStoNumb	Uint64	-	-		Storage number of the associated runway
SynchTyp	SynchronizationType	0-2	-		Synchronization Type
ThrCroHeig	Uint32	-	Ft		Height above the landing threshold on a normal glidepath

H.27 Msa

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirIcaCod	String	4 chars	-		Airport/Heliport ICAO Code
AirpoIden	String	6 chars	-	2108	Airport/Heliport Ident
AirStoNumb	Uint64	-	-		Airport/Heliport Storage Number
Country	CountryEntry	0-336	-	2116	Country where the MSA applies
IcaoCode	String	4 chars	-		MSA ICAO Code
MagTruIndi	MagneticTrueIndication	0-6	-		Magnetic/True Indication
MsaCenter	String	6 chars	-	2102	MSA Center
MsCeFiStNu	Uint64	-	-		MSA Center Fix Storage Number
MsaCenTyp	FixRecordType	0-8	-	2107	MSA Center Type
MultiCod	String	2 chars	-	2118	Multiple Code
NavKeyCod	Uint32	2 chars	-		Navaid key code if MSA center is a navaid
NavaidType	NavaidType	0-15	-		Navaid type if MSA center is a navaid
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of the MSA center fix
RouteIdent	String	50 chars	-	2111	Identifier of the terminal procedure associated with MSA

Attribute Name	Data Type	Range	Unit	Key	Description
RouteType	RouteType	0-4	-		Type of terminal procedure associated with MSA
SectoAltit	Uint32	-	Ft		Sector Altitude
SecEndBear	Uint32	0-359	Deg		Sector End Bearing
SecEndRadi	Uint32	-	Nm		Sector Radius
SecStaBear	Uint32	0-359	Deg		Sector Start Bearing
SecStaRadi	Uint32	-	Nm		Sector Start Radius
SectoAlti1	Uint32		Ft		Sector Altitude
SecEndBea1	Uint32	0-359	Deg		Sector End Bearing
SectoRadiu	Uint32	-	Nm		Sector Radius
SecStaBea1	Uint32	0-359	Deg		Sector Start Bearing
SectoAlti2	Uint32	-	Ft		Sector Altitude
SecEndBea2	Uint32	0-359	Deg		Sector End Bearing
SectoRadi1	Uint32	-	Nm		Sector Radius
SecStaBea2	Uint32	0-359	Deg		Sector Start Bearing
SectoAlti3	Uint32	-	Ft		Sector Altitude
SecEndBea3	Uint32	0-359	Deg		Sector End Bearing
SectoRadi2	Uint32	-	Nm		Sector Radius
SecStaBea3	Uint32	0-359	Deg		Sector Start Bearing

Attribute Name	Data Type	Range	Unit	Key	Description
SectoAlti4	Uint32	-	Ft		Sector Altitude
SecEndBea4	Uint32	0-359	Deg		Sector End Bearing
SectoRadi3	Uint32	-	Nm		Sector Radius
SecStaBea4	Uint32	0-359	Deg		Sector Start Bearing
SectoAlti5	Uint32	-	Ft		Sector Altitude
SecEndBea5	Uint32	0-359	Deg		Sector End Bearing
SectoRadi4	Uint32	-	Nm		Sector Radius
SecStaBea5	Uint32	0-359	Deg		Sector Start Bearing
SectoAlti6	Uint32	-	Ft		Sector Altitude
SecEndBea6	Uint32	0-359	Deg		Sector End Bearing
SectoRadi5	Uint32	-	Nm		Sector Radius
SecStaBea6	Uint32	0-359	Deg		Sector Start Bearing

H.28 Navaid

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirIcaCod	String	4 chars	-		ICAO code of the associated airport/heliport
AirpoIden	String	6 chars	-	2108	Ident of the associated airport/heliport
AirStoNumb	Uint64	-	-		Storage number of the associated airport/heliport
AsCoStNumb	Uint64	-	-		Associated Comms record storage number
AssMarTyp	AssociatedMarkerType	0-4	-		Associated marker type
BfoOperati	Logical	Boolean	-		BFO operation flag
BiasedIls	Logical	Boolean	-		Biased ILS flag
Channel	String	6 chars	-	2110	Assigned channel
Collocatio	NavaidCollocation	0-8	-		Navaid collocation information
CompoTyp	ComponentType	0-10	-		Component type (e.g.: DME, locator, etc.)
Country	CountryEntry	0-336	-	2116	Country where the navaid is located
Declinatio	Float32	+/-180	Deg		Station declination
DecliRefer	MagneticTrueIndication	0-6	-		Magnetic, True, or other (grid direction)
DmeIdent	String	6 chars	-		DME identifier

Attribute Name	Data Type	Range	Unit	Key	Description
DmeOffset	Float32		Nm		DME offset
Point2	GeoCoordinate	x,y,z	-		DME position (longitude, latitude, altitude)
EmissTyp	EmissionType	0-3	-		Emission type (A0, A1 or A2)
Frequency	Uint64	-	Hz	2104	Navaid frequency
FreProAlti	Uint32	-	Ft		Frequency protection altitude
FreProDist	Uint32	-	Nm		Frequency protection distance
IcaoCode	String	4 chars	-	2103	Navaid ICAO area code
Ident	String	6 chars	-	2102	Navaid ICAO Ident
KeyCode	Uint32	-	-	2118	Distinguish between same type navaid with same ident and country
LocalBeari	Float32	+/-180	Deg		Localizer bearing
LocBeaRefe	MagneticTrueIndication	0-6	-		Magnetic, True, or other (grid direction)
LocalWidt	Float32	+/-180	Deg		Localizer width
MagneVaria	Float32	+/-180	Deg		Magnetic variation
Modulation	SignalModulation	0-2	-		Modulation (400Hz or 1020Hz)
Name	String	45 chars	-		Navaid official name
NexNavDist	Uint32	-	Nm		Distance to the next navaid
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of the NavObject

Attribute Name	Data Type	Range	Unit	Key	Description
Power	Uint32	-	Watt		Navaid power capacity
PreciDm	Logical	Boolean	-		Precision vs non-precision DME
RadClaCod	RadioClassCode	0-7	-		Navaid radio class code
Range	Uint32	-	Nm		Navaid power capacity
RangeRelia	RangeReliability	0-10	-		Navaid range reliability
RepetRat	Uint32	-	1/min		NDB repetition rate [number of occurrences per minute]
RunwaDista	Float32	-	Nm		Distance to associated runway
RunwaIden	String	6 chars	-		Associated runway identifier
State	StateEntry	0-51	-		State or province name where the navaid is located
Status	NavaidStatus	0-3	-		Navaid status
SynchTyp	SynchronisationType	0-2	-		Navaid synchronization type
ThrCroHeig	Uint32	-	Ft		Threshold crossing height
Type	NavaidType	0-15	-	2107	Navaid type
VhfNavaid	Logical	Boolean	-	2122	Flag indicating if navaid is a VHF navaid.
VoIdFiPat	String	30 chars	-		Voice identifier file name and path
VoIdePres	Logical	Boolean	-		Voice identifier present flag
VoiOnFrequ	Logical	Boolean	-		Voice on frequency presence flag
VoOnFrFil	String	30 chars	-		Voice on frequency file link

Attribute Name	Data Type	Range	Unit	Key	Description
WeathBroad	WeatherBroadcast	0-2	-		Weather broadcast information
WeaBroFil	String	30 chars	-		Weather broadcast file link

H.29 Off Route Terrain Clearance Altitude

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AlterIden	String	8 chars		2108	Alternate OffRouteTerrainClearanceAlt identifier
Altitude	Uint32	-	Ft		Altitude: 1000ft clearance in non-mountainous & 2000ft in mountainous areas of US and 3000ft clearance for NIMA products.
Ident	String	8 chars	-	2102	OffRouteTerrainClearanceAlt identifier
Point2	GeoCoordinate	x,y,z	-		North east corner (longitude, latitude, altitude) of the cell in which altitude applies
Point1	GeoCoordinate	x,y,z	-		North west corner (longitude, latitude, altitude) of the cell in which altitude applies
Point3	GeoCoordinate	x,y,z	-		South east corner (longitude, latitude, altitude) of the cell in which altitude applies
Point4	GeoCoordinate	x,y,z	-		South west corner (longitude, latitude, altitude) of the cell in which altitude applies

H.30 ParachuteJumpArea

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AltitRefer	AltitudeReference	0-4	-		Altitude reference (eg: AMSL, AGL, etc.)
Country	CountryEntry	0-336	-	2116	Country where the parachute jump area is located
EffecAltit	Sint32	-	Ft		Altitude for which the area is effective
EffecTim	String	50 chars	-		Indicates hours, dates, or condition of operation
IcaoCode	String	4 chars	-		ICAO region code
Ident	String	8 chars	-	2102	DAFIF parachute jump area identifier
Name	String	50 chars	-		Official name assigned to the jump area
OperaHour	String	20 chars	-		Actual hours of operation
OperaTim	String	95 chars	-		Operating times of the area
Point1	GeoCoordinate	x,y,z	-		Reference Position (longitude, latitude, altitude)
StateName	StateEntry	0-51	-		State or province where the jump area is located

H.31 ParachuteJumpAreaBoundary

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
ArcSegDeri	ArcSegmentDerivation	0-3	-		Indicates how the arc segment is defined
Bearing1	Float32	+/-180	Deg		Bearing from navigational aid to designated area
Bearing2	Float32	+/-180	Deg		Bearing from navigational aid to designated area
BoundShap	BoundaryShape	0-8	-		Type of area point being plotted by positions, radii, etc.
Point2	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of circle or arc center
Country	CountryEntry	0-336	-	2116	Country in which boundary segment is located
Distance1	Float32	-	Nm		Distance from navigational aid to the designated area
Distance2	Float32	-	Nm		Distance from navigational aid to the designated area
IcaoCode	String	4 chars	-		ICAO code
Ident	String	8	-	2102	DAFIF parachute jump area identifier
NavaiCount	CountryEntry	0-336	-		Country where the navaid is located
NavaiIden	String	6 chars	-		Navaid identifier
NavKeyCod	Uint32	-	-		Navaid key code
Point3	GeoCoordinate	x,y,z	-		Navaid position (longitude, latitude, altitude)

NavStoNumb	Uint64	-	-		Associated navaid storage number
NavaidType	NavaidType	0-15	-		Navaid type
PaJuArStNu	Uint64	-	-		Storage number of associated ParachuteJumpArea record
Radius1	Float32	-	Nm		Radius of arc or circle from the center position
Radius2	Float32	-	Nm		Radius of arc or circle from the center position
Point1	GeoCoordinate	x,y,z	-		Reference Position (longitude, latitude, altitude)
Point4	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of the segment end position
Point5	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of the segment start position
SequeNumbe	Uint32	-	-	2115	Sequence number
Type	ParachuteJumpAreaType	0-7	-		Parachute jump area boundary type

H.32 PathPoint

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirpoIden	String	6 chars	-	2102	Associated airport/heliport identifier
AirStoNumb	Uint64	-	-		Associated airport/heliport storage number
AppPerDesi	ApproachPerformance	0-0	-		Indicates the category type of the approach (APD)
AppRouIden	String	6 chars	-		Identifier of the approach route to be flown
AppSegTyp	ApproachSegmentType	0-1	-		Type of the final approach segment (operations type)
Country	CountryEntry	0-336	-	2116	Country in which the airport/heliport is located
FlPaAlElHe	Sint32	-	Ft		Surveyed height in reference to WGS-84 ellipsoid
FlPaAlOrHe	Sint32	-	Ft		Surveyed height in reference to Mean Sea Level (MSL)
Point2	GeoCoordinate	x,y,z	-		Flight path alignment point (FPAP) position (longitude, latitude, altitude)
GlideAngl	Float32	+/-180	Deg		Intended descent angle for final approach flight path
IcaoCode	String	4 chars	-		ICAO code for the airport/heliport
LaThElHeig	Sint32	-	Ft		Surveyed height in reference to WGS-84 ellipsoid

LaThOrHeig	Sint32	-	Ft		Surveyed height in reference to Mean Sea Level (MSL)
Point1	GeoCoordinate	x,y,z	-		Landing threshold point (LTP) position (longitude, latitude, altitude)
LengtOffse	Uint32	-	Ft		Distance from stop end of runway (SER) to the FPAP
RePaDaSele	PathDataSelector	0-0	-		Reference path data selector enables automatic tuning of a procedure by Ground Based Augmentation Systems (GBAS) avionics
RefPatIden	String	6 chars	-		Ident to confirm selection of correct approach procedure
RouteIndic	String	25 chars	-		Differentiates between multiple final approach segments to the same runway or helipad (single alpha character)
RunwaIden	String	6 chars	-		Associated runway/helipad identifier
ServiProvi	ServiceProvider	0-0	-		Associates approach procedure to a particular Satellite Based Approach System (SBAS) service provider
ThrCouWidt	Float32	-	Ft		Width of lateral course at Landing Threshold Point
ThrCroHeig	Uint32	6	Ft		Height above landing threshold on a normal glidepath

H.33 PreferredRoute

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AircrGrou	AircraftGroup	0-21	-		Types of aircrafts permitted to use the route
AirwaLeve	AirwayLevel	0-3	-		Airway level (high, low, or both)
AlRoAiGrou	AircraftGroup	0-21	-		Types of aircrafts permitted to use the alternate route
AltitDescr	AltitudeDescription	0-13	-		Description of how segment altitude limits should be applied
DirecRestr	DirectionRestriction	0-3	-		Direction restriction (forward, backward, either)
EffecTime	String	50 chars	-		Period during which preferred route is effective
EffecTime1	String	50 chars	-		Period during which preferred route is effective
EffecTime2	String	50 chars	-		Period during which preferred route is effective
FixCountry	CountryEntry	0-336	-	2116	Country where the fix point is located
FixIcaCod	String	4 chars	-		ICAO code of fix point
FixIdent	String	30 chars	-		Fix identifier (may be name if ident not available)
FiNaKeCod	Uint32	-	-		Key code of fix point for navaid fix
FixPoiTyp	FixPointType	0-19	-		Fix point type for navaid and ATS fixes
FiPoReTyp	FixPointRecordType	0-13	-		Fix record type

Attribute Name	Data Type	Range	Unit	Key	Description
FixStoNumb	Uint64	-	-		Fix storage number
Ident	String	8 chars	-	2102	Route identifier
InFiIcCod	String	4 chars	-		ICAO code of the initial fix point
IniFixIden	String	6 chars	-		Identifier of departure airport or initial fix of the route
IniFixNam	String	50 chars	-		Name of the initial fix point
InFiReTyp	FixRecordType	0-8	-		Initial fix record type
InFiStNumb	Uint64	-	-		Storage number of the associated initial fix point
MaxRouAlti	Sint32	-	Ft		Maximum altitude limit for route
MaSpLiFla	Logical	Boolean	-		Speed limit represents maximum speed allowed (FALSE - min speed)
MinRouAlti	Sint32	-	Ft		Minimum altitude limit for route
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of fix point
RefRouIden	String	6 chars	-		Reference route identifier (route to be flown)
RnaReqFla	Logical	Boolean	-		RNAV equipment required flag
RouteUse	RouteUse	0-2	-		Route use (point-to-point or area-to-area)
RoutiTyp	RoutingType	0-7	-		Type of reference route
SegAltLimi	Sint32	-	Ft		Segment altitude limit 1
SegAltLim1	Sint32	-	Ft		Segment altitude limit 2
SequeNumbe	Uint32	-	-		Sequence number

Attribute Name	Data Type	Range	Unit	Key	Description
SpeedLimit	Uint32	-	Kts		Speed limit for the route
TeFiIcCod	String	4 chars	-		ICAO code of the terminal fix point
TerFixIden	String	6 chars	-		Identifier of arrival airport or terminal fix of the route
TerFixNam	String	50 chars	-		Name of the terminal fix point
TeFiReTyp	FixRecordType	0-8	-		Terminal fix record type
TeFiStNumb	Uint64	-	-		Storage number of the associated terminal fix point
TimeCode	PrimaryTimeCode	0-4	-		Describes continuity of time of applicability
Type	PreferredRouteType	0-9	-		Preferred route type

H.34 Preset Site

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirTruBear	Float32	0-360	Deg		True bearing of aircraft at the preset site
Airpolden	String	6 chars	-	2108	Identifier of the associated airport
AirStoNumb	Uint64	-	-		Storage number of the associated airport
Ident	String	32 chars	-	2102	PresetSite identifier
Point1	GeoCoordinate	x,y,z	-		Preset site position (longitude, latitude, altitude)
Runwalden	String	6 chars	-		Ident of the associated runway
RunStoNumb	Uint64	-	-		Storage number of the associated runway
SegmeNumbe	Uint32	-	-		The segment number of the preset site, if it belongs to a segment group
Type	PresetSiteType	0-8	-		Type of preset site

H.35 RestrictiveAirspace

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirspDesig	String	6 chars	-	2102	Restrictive airspace designation
AirResTyp	AirspaceRestrictionType	0-9	-		Restrictive airspace type
ArcBearing	Float32	+/-180	Deg		Arc bearing
ArcDistanc	Float32	-	Nm		Arc distance
ArcDistan1	Float32	-	Nm		Arc distance (radius of arc from center point)
Point3	GeoCoordinate	x,y,z	-		Arc origin position (longitude, latitude, altitude)
ArcSegDeri	ArcSegmentDerivation	0-3	-		Indicates how the arc segment is defined
Bearing1	Float32	+/-180	Deg		True bearing from arc origin or navaid
Bearing2	Float32	+/-180	Deg		True bearing from arc origin or navaid
BoundEn	Logical	Boolean	-		End of boundary description - return to origin point
BoundShap	BoundaryShape	0-8	-		Boundary shape type
Country	CountryEntry	0-336	-	2116	Country where airspace is located
IcaoCode	String	4 chars	-		ICAO code for the airspace
Level	AirwayLevel	0-3	-		Type of airway (high, low, or either)

LowerLimit	Sint32	-	Ft		Lower limit
LoLiAlRefe	AltitudeReference	0-4	-		Altitude reference
MultiCod	String	2 chars	-		Differentiate between airspaces with same designator
Name	String	50 chars	-		Restrictive airspace name
NavaiCount	CountryEntry	0-336	-		Country in which navaid is located
NavaiIden	String	6 chars	-		Navaid identifier
NavKeyCod	Uint32	-	-		Distinguish between same type navaid with same ident and country
NavaidType	NavaidType	0-15	-		Navaid type
Notam	Logical	Boolean	-		Active times by NOTAM
Point2	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude)
Point1	GeoCoordinate	x,y,z	-		Reference Position (longitude, latitude, altitude)
Sector	String	100 chars	-	2117	Designation for the section of the airspace
SequeNumbe	Uint32	-	-		Sequence number
SpUsAiStNu	Uint64	-	-		Associated SpecialUseAirspace storage number
TimeCode	PrimaryTimeCode	0-4	-		Time codes for primary records
UpperLimit	Sint32	-	Ft		Upper limit
UpLiAlRefe	AltitudeReference	0-4	-		Reference for upper limit altitude

H.36 Runway

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirIcaCod	String	4 chars	-	2103	Associated Airport ICAO code.
AirpoIden	String	6 chars	-	2108	Associated Airport identifier.
AirStoNumb	Uint64	-	-		Associated Airport storage number.
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of the runway approach end.
Bearing	Float32	+/-180	Deg		Magnetic bearing.
CeLiLiFla	Logical	Boolean	-		Indicates presence of lights on center line.
ClosedFlag	Logical	Boolean	-		Indicates if the runway is closed or unusable.
Country	CountryEntry	0-336	-	2116	Runway country.
Descriptio	String	memo	-		Runway description.
DisThrDist	Uint32	-	Ft		Distance between the beginning of the runway and the displaced threshold.
Point2	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of the displaced threshold (latitude, longitude, elevation).
Ident	String	6 chars	-	2102	Runway identifier.
LanAidCate	LandingAidCategory	0-9	-		Category of the primary landing aid (ILS, MLS, GLS).
LanAidIden	String	6 chars	-		Primary landing aid (ILS, MLS or GLS) identifier.

Attribute Name	Data Type	Range	Unit	Key	Description
LandiDista	Uint32	-	Ft		Landing distance available.
Length	Uint32	-	Ft		Runway length.
LightSyste	LightingSystem	0-64	-		Lighting system 1.
LightSyst1	LightingSystem	0-64	-		Lighting system 2.
LightSyst2	LightingSystem	0-64	-		Lighting system 3.
LightSyst3	LightingSystem	0-64	-		Lighting system 4.
LightSyst4	LightingSystem	0-64	-		Lighting system 5.
LightSyst5	LightingSystem	0-64	-		Lighting system 6.
LightSyst6	LightingSystem	0-64	-		Lighting system 7.
LightSyst7	LightingSystem	0-64	-		Lighting system 8.
MaxTirPres	MaximumTirePressure	0-4	Psi		Maximum tire pressure authorized.
PavemClass	Uint32	-	-		Pavement classification number.
PavEvaMeth	PavementEvaluationMethod	0-2	-		Pavement evaluation method.
PavSubCate	PavementSubgradeCategory	0-4	-		Pavement subgrade category.
PavemTyp	PavementType	0-3	-		Type of pavement.
SeLaAiCate	LandingAidCategory	0-9	-		Category of the second landing aid (ILS, MLS, GLS).
SeLaAiIden	String	6 chars	-		Second landing aid (ILS, MLS or GLS) identifier.
Slope	Float32	-	%		Runway gradient.

Attribute Name	Data Type	Range	Unit	Key	Description
Point3	GeoCoordinate	x,y,z	-		Position (longitude, latitude, elevation) of the runway stop end.
StopwLengt	Uint32	-	Ft		Length of the area beyond the takeoff runway.
StoSurTyp	RunwaySurfaceType	0-21	-		Stopway surface type.
SurfaTyp	RunwaySurfaceType	0-21	-		Runway surface type.
TakeoDista	Uint32	-	Ft		Takeoff distance available.
ThrCroHeig	Uint32	-	Ft		Height above the landing threshold on a normal glidepath.
TouZonElev	Float32	-	Ft		Highest elevation in the first 3000 ft of landing surface.
TrueBearin	Float32	+/-180	Deg		Runway true bearing.
TruNorRefe	Logical	Boolean	-		True North reference flag.
Width	Uint32	-	Ft		Runway width.

H.37 Sid

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirStoNumb	Uint64	-	-		Airport storage number
Altitude1	Sint32	-	Ft		First altitude limit
AltitTyp	AltitudeType	0-4	-		Altitude 1 type
Altitude2	Sint32	-	Ft		Second altitude limit
AltitTyp1	AltitudeType	0-4	-		Altitude 2 type
AltitDescr	AltitudeDescription	0-13	-		Altitude description
ArcRadius	Float32	-	Nm		Arc radius
CenterFix	String	10 chars	-		Point which defines the center of the arc flight path
CeFiIcCod	String	4 chars	-		ICAO code of the center fix
Country	CountryEntry	0-336	-	2116	Country associated with the terminal procedure
Course	Float32	+/-180	Deg		Outbound course from waypoint in fix ident
FixDetails	FixDetails	0-9	-		Fix details
FixFunctio	FixFunction	0-7	-		Fix function
FixIcaCod	String	4 chars	-		ICAO code of the fix point

Attribute Name	Data Type	Range	Unit	Key	Description
FixIdent	String	10 chars	-		Fix identifier
FlyOveTyp	FlyOverType	0-4	-		Fly over type
MagCouIndi	MagneticTrueIndication	0-6	-		Indicates if the course provided is magnetic course
NavaiCount	CountryEntry	0-336	-		Country where recommended navaid 1 is located
Point2	GeoCoordinate	x,y,z	-		Navaid 1 DME position (longitude, latitude, altitude)
NavKeyCod	Uint32	-	-		Distinguish between navaid of same type with same ident in same country
NavMagVari	Float32	+/-180	Deg		Recommended navaid 1 magnetic variation
Point3	GeoCoordinate	x,y,z	-		Navaid 1 position (longitude, latitude, altitude)
NavaiTyp	SegmentNavaidType	0-13	-		Recommended navaid 1 type
NavaiCoun1	CountryEntry	0-336	-		Country where recommended navaid 2 is located
Point4	GeoCoordinate	x,y,z	-		Navaid 2 DME position (longitude, latitude, altitude)
NavKeyCod1	Uint32	-	-		Distinguish between navaid of same type with same ident in same country
NavMagVar1	Float32	+/-180	Deg		Recommended navaid 2 magnetic variation
Point5	GeoCoordinate	x,y,z	-		Navaid 2 position (longitude, latitude, altitude)
NavaiTyp1	SegmentNavaidType	0-13	-		Recommended navaid 2 type
PathTermin	PathTermination	0-23	-		Path and Termination
ReNaIcCod	String	4 chars	-		ICAO code of the recommended navaid 1

Attribute Name	Data Type	Range	Unit	Key	Description
RecNavIden	String	10 chars	-		Recommended navaid identifier 1
RecNavIde1	String	10 chars	-		Recommended navaid identifier 2
ReqNavPerf	Float32	-	Nm		Required navigation performance
RouteDista	Float32	-	Nm		Distance in nautical miles from waypoint in fix ident
RouteType	RouteType	0-4	-		Termination Procedure Type
SpeAirCate	AircraftCategory	0-4	-		Aircraft category that speed limit 1 applies to
SpeedAltit	Sint32	-	Ft		Altitude where speed limit 1 applies
SpeedLimit	Uint32	-	Kts		Speed limit 1
SpeAirCat1	AircraftCategory	0-4	-		Aircraft category that speed limit 2 applies to
SpeedAlti1	Sint32	-	Ft		Altitude where speed limit 2 applies
SpeedLimi1	Uint32	-	Kts		Speed limit 2
SuTeDaStNu	Uint64	-	-		Storage number of associated Supplemental Terminal Data record
ThrCroHeig	Uint32	-	Ft		Threshold crossing height
TransAltit	Sint32	-	Ft		Transition altitude
TurnDirect	TurnDirection	0-3	-		Turn direction
TurDirVali	Logical	Boolean	-		Turn direction valid
WaypoCount	CountryEntry	0-336	-		Waypoint country
WaypoDescr	WaypointDescription	0-15	-		Waypoint description

Attribute Name	Data Type	Range	Unit	Key	Description
WaypoDista	Float32	-	Nm		Nautical miles between fix point and recommended navaid 1 (RHO)
WaypoDist1	Float32	-	Nm		Nautical miles between fix point and recommended navaid 2
WayMagBear	Float32	+/-180	Deg		Magnetic bearing between fix point and recommended navaid 1 (THETA)
WayMagBea1	Float32	+/-180	Deg		Magnetic bearing between fix point and recommended navaid 2
WayMagVari	Float32	+/-180	Deg		Waypoint magnetic variation
Point1	GeoCoordinate	x,y,z	-		Waypoint position (longitude, latitude, altitude)
AirIcaCod	String	4 chars	-		ICAO code of the associated airport
AirpoIden	String	6 chars	-	2102	Identifier of the associated airport
Ident	String	8 chars	-	2108	SID/STAR/Approach identifier
SequeNumbe	Uint32	-	-		Sequence number
SidRouTyp	SidRouteType	0-12	-		SID route type
TransIden	String	60 chars	-		Transition identifier

H.38 Special Use Airspace

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.

AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirwaLeve	AirwayLevel	0-3	-		Airspace structure in which boundary is effective (high/low)
ComCalSig	String	50 chars	-	2111	Call sign of the communications facilities
ContrAgenc	String	60 chars	-		Office responsible for air traffic within airspace
Country1	CountryEntry	0-336	-	2116	Country in which the special use airspace is located
Country2	CountryEntry	0-336	-		Country in which the special use airspace is located
Country3	CountryEntry	0-336	-		Country in which the special use airspace is located
Country4	CountryEntry	0-336	-		Country in which the special use airspace is located
EffecDat	String	12 chars	-		Effective date of the special use airspace
EffecTim	String	50 chars	-		Times at which given airspace iWs to be in effect
Frequency1	Uint64	-	Hz		Frequency for communicating with identified facility
Frequency2	Uint64	-	Hz		Frequency 2 used for communicating with identifed facility
IcaoCode	String	4 chars	-		ICAO code of the special use airspace
Ident	String	6 chars	-	2102	ICAO ident of special use airspace
LowEffAlti	Sint32	-	Ft		Lower vertical limit of the given airspace
LoEfAlRefe	AltitudeReference	0-4	-		Lower effective altitude reference
Name	String	50 chars	-		Official name of the special use airspace
Point1	GeoCoordinate	x,y,z	-		Reference Position (longitude, latitude, altitude)
Remark	String	memo	-		Essential information related to the given special use airspace

Sector	String	2 chars	-	2117	Designation for the section of the special use airspace
Type	AirspaceRestrictionType	0-9	-		Special use airspace type
UppEffAlti	Sint32	-	Ft		Upper vertical limit of the given airspace
UpEfAlRefe	AltitudeReference	0-4	-		Upper effective altitude reference
WeathCondi	WeatherCondition	0-7	-		Meteorological conditions in which the airspace can be used

H.39 Star

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirStoNumb	Uint64	-	-		Airport storage number
Altitude1	Sint32	-	Ft		First altitude limit
AltitTyp	AltitudeType	0-4	-		Altitude 1 type
Altitude2	Sint32	-	Ft		Second altitude limit
AltitTyp1	AltitudeType	0-4	-		Altitude 2 type
AltitDescr	AltitudeDescription	0-13	-		Altitude description
ArcRadius	Float32	-	Nm		Arc radius
CenterFix	String	10 chars	-		Point which defines the center of the arc flight path
CeFiIcCod	String	4 chars	-		ICAO code of the center fix
Country	CountryEntry	0-336	-	2116	Country associated with the terminal procedure
Course	Float32	+/-180	Deg		Outbound course from waypoint in fix ident
FixDetails	FixDetails	0-9	-		Fix details
FixFunctio	FixFunction	0-7	-		Fix function
FixIcaCod	String	4 chars	-		ICAO code of the fix point

Attribute Name	Data Type	Range	Unit	Key	Description
FixIdent	String	10 chars	-		Fix identifier
FlyOveTyp	FlyOverType	0-4	-		Fly over type
MagCouIndi	MagneticTrueIndication	0-6	-		Indicates if the course provided is magnetic course
NavaiCount	CountryEntry	0-336	-		Country where recommended navaid 1 is located
Point2	GeoCoordinate	x,y,z	-		Navaid 1 DME position (longitude, latitude, altitude)
NavKeyCod	Uint32	-	-		Distinguish between navaid of same type with same ident in same country
NavMagVari	Float32	+/-180	Deg		Recommended navaid 1 magnetic variation
Point3	GeoCoordinate	x,y,z	-	A	Navaid 1 position (longitude, latitude, altitude)
NavaiTyp	SegmentNavaidType	0-13	-		Recommended navaid 1 type
NavaiCoun1	CountryEntry	0-336	-		Country where recommended navaid 2 is located
Point4	GeoCoordinate	x,y,z	-		Navaid 2 DME position (longitude, latitude, altitude)
NavKeyCod1	Uint32	-	-		Distinguish between navaid of same type with same ident in same country
NavMagVar1	Float32	+/-180	Deg		Recommended navaid 2 magnetic variation
Point5	GeoCoordinate	x,y,z	-		Navaid 2 position (longitude, latitude, altitude)
NavaiTyp1	SegmentNavaidType	0-13	-		Recommended navaid 2 type
PathTermin	PathTermination	0-23	-		Path and Termination
ReNaIcCod	String	4 chars	-		ICAO code of the recommended navaid 1

Attribute Name	Data Type	Range	Unit	Key	Description
RecNavIden	String	10 chars	-		Recommended navaid identifier 1
RecNavIde1	String	10 chars	-		Recommended navaid identifier 2
ReqNavPerf	Float32	-	Nm		Required navigation performance
RouteDista	Float32	-	Nm		Distance in nautical miles from waypoint in fix ident
RouteType	RouteType	0-4	-		Termination Procedure Type
SpeAirCate	AircraftCategory	0-4	-		Aircraft category that speed limit 1 applies to
WWSpeedAltit	Sint32	-	Ft		Altitude where speed limit 1 applies
SpeedLimit	Uint32	-	Kts		Speed limit 1
SpeAirCat1	AircraftCategory	0-4	-		Aircraft category that speed limit 2 applies to
SpeedAlti1	Sint32	-	Ft		Altitude where speed limit 2 applies
SpeedLimi1	Uint32	-	Kts		Speed limit 2
SuTeDaStNu	Uint64	-	-		Storage number of associated Supplemental Terminal Data record
ThrCroHeig	Uint32	-	Ft		Threshold crossing height
TransAltit	Sint32	-	Ft		Transition altitude
TurnDirect	TurnDirection	0-3	-		Turn direction
TurDirVali	Logical	Boolean	-		Turn direction valid
WaypoCount	CountryEntry	0-336	-		Waypoint country

Attribute Name	Data Type	Range	Unit	Key	Description
WaypoDescr	WaypointDescription	0-15	-		Waypoint description
WaypoDista	Float32	-	Nm		Nautical miles between fix point and recommended navaid 1 (RHO)
WaypoDist1	Float32	-	Nm		Nautical miles between fix point and recommended navaid 2
WayMagBear	Float32	+/-180	Deg		Magnetic bearing between fix point and recommended navaid 1 (THETA)
WayMagBea1	Float32	+/-180	Deg		Magnetic bearing between fix point and recommended navaid 2
WayMagVari	Float32	+/-180	Deg		Waypoint magnetic variation
Point1	GeoCoordinate	x,y,z	-		Waypoint position (longitude, latitude, altitude)
AirIcaCod	String	4 chars	-		ICAO code of the associated airport
AirpoIden	String	6 chars	-	2102	Identifier of the associated airport
Ident	String	8 chars	-	2108	SID/STAR/Approach identifier
SequeNumbe	Uint32	-	-		Sequence number
StaRouTyp	StarRouteType	0-12	-		STAR route type
TransIden	String	6 chars	-		Transition identifier
VertiAngl	Float32	+/-180	Deg		Descent angle for the procedure

H.40 Supplemental Terminal Data

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AgencRespo	String	8 chars	-		Military or federal agency primarily responsible for terminal procedure
AirpoIden	String	6 chars	-	2108	Airport/Heliport identifierW
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of airport
AirStoNumb	Uint64	-	-		Airport/Heliport storage number
AltMinTyp	AlternateMinimumType	0-2	-		Alternate minimum not standard or not authorized
Country	CountryEntry	0-336	-	2116	Country associated with supplemental terminal procedure data
EmeSafAlti	Uint32	-	Ft		Safe altitude providing obstacle clearance [above MSL]
IcaoCode	String	4 chars	-		Terminal procedure ICAO code
Ident	String	40 chars	-	2102	Terminal procedure identifier
OperaAgenc	String	255 chars	-		Host country agency with authority for the terminal procedure
Remark	String	memo	-		Essential information applying to the entire procedure
RouQuaTyp	RouteQualifierType	0-2	-		Supplements route type - applies to GPS & RNAV type procedures
RouteType	RouteType	0-4	-		Terminal procedure route type

TakMinTyp	TakeoffMinimumType	0-1	-		Takeoff minimum not standard and/or departure procedure are published
TransAltit	Uint32	-	Ft		Altitude below which vertical position controlled by reference to altitudes [above MSL]
TransLeve	Uint32	-	Ft		Lowest flight level above transition altitude [above MSL]

H.41 Terminal Procedure Climb

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirpoIden	String	6 chars	-	2108	Airport/Heliport identifier
AirStoNumb	Uint64	-	-		Airport/Heliport storage number
ClimbAltit	Uint32	-	Ft		Altitude to which climb rate applies [above MSL]
ClimbFootn	String	90 chars	-		Footnote associated with climb information
CliRatTyp	ClimbRateType	0-4	-		Minimum rate, or ATC climb rate if higher than min. climb rate
Country	CountryEntry	0-336	-	2116	Country associated with terminal procedure climb data
DesceRat	Uint32	-	Ft/m		Minimum or ATC climb rate/descent [vertical velocity ft/min]
IcaoCode	String	4 chars	-		Terminal procedure ICAO code
Ident	String	40 chars	-	2102	Terminal procedure identifier
MinCliRat	Uint32	-	Kts		Minimum climb rate based on 60 knots
OccurNumbe	Uint32	-	-		Number of occurrences for a given runway
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of airport
RouteType	RouteType	0-4	-		Terminal procedure route type
Runwalden	String	6 chars	-		Runway at which the climb rate information applies

H.42 Terminal Procedure Feeder Route

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirpoIden	String	6 chars	-	2108	Airport identifier
AirStoNumb	Uint64	-	-		Airport storage number
Altitude	Sint32	-	Ft		Referenced altitude associated with feeder route segment
Country	CountryEntry	0-336	-	2116	Country associated with terminal procedure feeder route
Course	Float32	+/-180	Deg		Course from waypoint 1 to waypoint 2 in route segment
IcaoCode	String	4 chars	-		Feeder route ICAO code
Ident	String	10 chars	-	2102	Feeder route identifier
MagCouIndi	MagneticTrueIndication	0-6	-		Indicates if course is given in degrees magnetic, true or other
RouteDista	Float32	-	Nm		Distance between waypoint 1 and waypoint 2
RouteType	RouteType	0-4	-		Terminal procedure route type
SequeNumbe	Uint32	-	-	2115	Feeder route sequence number
TerProIden	String	40 chars	-	2126	Terminal procedure identifier
WaypoCount	CountryEntry	0-336	-		Waypoint 1 country
WaypoIden	String	6 chars	-		Waypoint 1 identifier

Point1	GeoCoordinate	x,y,z	-		Waypoint 1 position (longitude, latitude, altitude)
WaypoCoun1	CountryEntry	0-336	-		Waypoint 2 country
WaypoIden1	String	6 chars	-		Waypoint 2 identifier
Point2	GeoCoordinate	x,y,z	-		Waypoint 2 position (longitude, latitude, altitude)

H.43 Terminal Procedure Minima

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirpoIden	String	6 chars	-	2108	Airport/Heliport identifier
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of airport
AirStoNumb	Uint64	-	-		Airport/Heliport storage number
ApproTyp	String	30 chars	-	2107	Type of approach on which minimum data is based
CaADeHeigh	Uint32	-	Ft		Height above highest elevation in the touchdown zone - for a straight in or glideslope approach [above MSL]
CaAHeAbTou	Uint32	-	Ft		Height above highest elevation in the touchdown zone
CaAPrVisib	Float32	-	m		Designated visibility for the approach
CaARuVisib	Float32	-	m		Determined by atmospheric conditions or instrumentally derived value for runway visual range
CaAWeCeili	Float32	-	m		Height equal to or greater than decision height or minimum descent altitude above airport or heliport elevation
CaBDeHeigh	Uint32	-	Ft		Height above highest elevation in the touchdown zone - for a straight in or glideslope approach [above MSL]
CaBHeAbTou	Uint32	-	Ft		Height above highest elevation in the touchdown zone
CaBPrVisib	Float32	-	m		Designated visibility for the approach

Attribute Name	Data Type	Range	Unit	Key	Description
CaBRuVisib	Float32	-	m		Determined by atmospheric conditions or instrumentally derived value for runway visual range
CaBWeCeili	Float32	-	m		Height equal to or greater than decision height or minimum descent altitude above airport or heliport elevation
CaCDeHeigh	Uint32	-	Ft		Height above highest elevation in the touchdown zone - for a straight in or glideslope approach [above MSL]
CaCHeAbTou	Uint32	-	Ft		Height above highest elevation in the touchdown zone
CaCPrVisib	Float32	-	m		Designated visibility for the approach
CaCRuVisib	Float32	-	m		Determined by atmospheric conditions or instrumentally derived value for runway visual range
CaCWeCeili	Float32	-	m		Height equal to or greater than decision height or minimum descent altitude above airport or heliport elevation
CaDDeHeigh	Uint32	-	Ft		Height above highest elevation in the touchdown zone - for a straight in or glideslope approach [above MSL]
CaDHeAbTou	Uint32	-	Ft		Height above highest elevation in the touchdown zone
CaDPrVisib	Float32	-	m		Designated visibility for the approach
CaDRuVisib	Float32	-	m		Determined by atmospheric conditions or instrumentally derived value for runway visual range
CaDWeCeili	Float32	-	m		Height equal to or greater than decision height or minimum descent altitude above airport or heliport elevation
CaEDeHeigh	Uint32	-	Ft		Height above highest elevation in the touchdown zone - for a straight in or glideslope approach [above MSL]

Attribute Name	Data Type	Range	Unit	Key	Description
CaEHeAbTou	Uint32	-	Ft		Height above highest elevation in the touchdown zone
CaEPrVisib	Float32	-	m		Designated visibility for the approach
CaERuVisib	Float32	-	m		Determined by atmospheric conditions or instrumentally derived value for runway visual range
CaEWeCeili	Float32	-	m		Height equal to or greater than decision height or minimum descent altitude above airport or heliport elevation
Country	CountryEntry	0-336	-	2116	Country associated with terminal procedure minima data
IcaoCode	String	4 chars	-		Terminal procedure ICAO code
Ident	String	40 chars	-	2102	Terminal procedure identifier
RouteType	RouteType	0-4	-		Terminal procedure route type
Remark	String	memo	-		Remarks give conditions affecting published approach minimums

H.44 VfrRoute

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirIcaCod	String	4 chars	-		ICAO code of the associated airport/heliport
AirpoIden	String	6 chars	-	2111	Identifier of the associated airport/heliport
Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of airport
AirStoNumb	Uint64	-	-		Storage number of the associated airport/heliport
Country	CountryEntry	0-336	-	2116	Country where the airport/heliport is located
Remark	String	memo	-		Essential information pertaining to part or to all route procedures at the airport/heliport
RouteIdent	String	6 chars	-	2102	Route identifier
RouteName	String	40 chars	-		Route name

H.45 VfrRouteSegment

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirIcaCod	String	4 chars	-	2103	ICAO code of the associated airport/heliport
AirpoIden	String	6 chars	-	2111	Identifier of the associated airport/heliport
AirStoNumb	Uint64	-	-		Storage number of the associated airport/heliport
Altitude	Uint32		Ft		Reference altitude [above sea level]
Country	CountryEntry	0-336	-	2116	Country where the airport/heliport is located
Course	Float32	+/-180	Deg		Inbound course to the point/checkpoint
CoursRefer	MagneticTrueIndication	0-6	-		Course reference (magnetic/true)
Point2	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) 0.5nm, at 90 degree angle to heading, to left of checkpoint
MgrsPositi	String	20 chars	-		MGRS position given using the UTM or the UPS grid
PathType	PathType	0-6	-		Defines how the route is used (eg: arrival, departure, etc.)
PointName	String	25 chars	-		Official name of point/checkpoint
PointDescr	String	40 chars	-		Landmark, graphical description of point/checkpoint
PoiRepTyp	PointReportingType	0-2	-		Indicates if point is compulsory for graphic presentation of the route

Point1	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) of point/checkpoint
Point3	GeoCoordinate	x,y,z	-		Position (longitude, latitude, altitude) 0.5nm, at 90 degree angle to heading, to right of checkpoint
RouteIdent	String	6 chars	-	2102	Route identifier
RouteName	String	40 chars	-		Route name
SegAltDesc	SegmentAltitudeDescription	0-5	-		Defines how the given altitude applies to the segment
SegmeNam	String	25 chars	-		Official segment name
SegmeNumbe	Uint32	-	-	2115	Defines relative position of segment in total VFR route segment
SegTurDire	PathTurnDirection	0-2	-		Direction in which course turns are to be made
SegmeTyp	SegmentType	0-3	-		Indicates if segment is a starting, next, or ending segment
SOEAAFla	Logical	Boolean	-		Flag indicating whether or not the segment starts or ends at an airport/heliport
VfRoStNumb	Uint64	-	-		Storage number of the associated VFR route record

H.46 Waypoint

Attribute Name	Data Type	Range	Unit	Key	Description
StoraNumbe	Uint64	-	-	2101	Storage number.
AHGT	Logical	1	-		Absolute Height above surface level Flag. Always true.
AirIcaCod	String	4 chars	-		ICAO code of the associated airport.
AirpoIden	String	6 chars	-		Ident of the associated airport.
AirStoNumb	Uint64	-	-		Storage number of the associated airport.
Bearing	Float32	+/-180	Deg		Bearing from navaid to waypoint
BeariRefer	MagneticTrueIndication	0-6	-		Bearing reference (magnetic, true, or 'grid')
ColloNavai	Logical	Boolean	-		Waypoint collocated with a navaid flag
Country	CountryEntry	0-336	-	2116	Country where the waypoint is located
Distance	Float32	-	Nm		Distance from navaid to waypoint
DynMagVari	Float32	+/-180	Deg		Dynamic magnetic variation
FixType	FixType	0-16	-		Fix Type
IcaoCode	String	4 chars	-	2103	ICAO code of waypoint
Ident	String	6 chars	-	2102	Waypoint Identifier
Name	String	50 chars	-		Waypoint name/description
NameFormat	NameFormatType	0-16	-		Format of waypoint name field

NavaiCount	CountryEntry	0-336	-		Country where navaid is located
NavaiIden	String	6 chars	-		Navaid identifier
NavKeyCod	Uint32	-	-		Distinguish between same type navaid with same ident and country
NavaidType	NavaidType	0-15	-		Navaid type
Point1	GeoCoordinate	x,y,z	-		Waypoint Position (longitude, latitude, altitude)
RnavWaypoi	Logical	Boolean	-		Waypoint is a RNAV waypoint
RouteType	RouteType	0-4	-		Route type
RunIcaCod	String	6 chars	-		Runway ICAO code
RunwaIden	String	6 chars	-		Runway identifier
RvsmIndica	RvsmIndicator	0-5	-		Waypoint RVSM indicator
StateName	StateEntry	0-51	-		State or province where waypoint is located
Type	WaypointType	0-15	-		Waypoint type
Usage	WaypointUsageType	0-9	-		Waypoint usage type
WayRecTyp	FixRecordType	0-8	-	2122	Waypoint record type

Appendix I

I. Navaids Attribution Enumeration Values

This section describes the attributes specific to each NAV category whose values are enumerated in accordance to this appendix.

Enumeration Name	Enumerator Description	Values
BoxRegionType		
	Remained Region	0
	Added Region	1
	Removed Region	2
AircraftCategory		
	All aircrafts	0
	Jets only	1
	Turbo props only	2
	Other	3
	Not Defined	4
AircraftGroup		
	All Aircraft	0
	All Aircraft, Cruise speed 250 kts or less	1
	Non-Jet and Turbo Prop	2
	Multi-Engine Props Only	3
	Jets & Turbo Props/Spec., Cruise Spd 190kts or greater	4
	Helicopter Only	5
	Jet Power	6
	Turbo-Prop/Special, Cruise Speed 190 kts or greater	7
	Non-Jet, Non-Turbo Prop	8
	Non-Jet, Cruise Speed 190 kts or greater	9
	Non-Jet, Cruise Speed 189 kts or less	10
	Aircraft as defined in a Continuation Record Note	11
	Single Engine	12

Enumeration Name	Enumerator Description	Values
	Twin Engine	13
	Non Turbo Jets	14
	Non Jets	15
	Props	16
	Turbo Props	17
	Turbo Jets	18
	Water Turbo Jets	19
	Water Turbo Props	20
	Not defined	21
AirspaceBoundaryType		
	Advisory Area (ADA or UDA)	0
	Air Defense Identification Zone (ADIZ)	1
	Air Route Traffic Control Center (ARTCC)	2
	Area Control Center (ACC)	3
	Buffer Zone (BZ)	4
	Control Area or Special Rules Area	5
	Ctrl/Special Rules/Military Traffic Zone	6
	Flight Information Region (FIR)	7
	Ocean Control Area (OCA)	8
	Radar Area	9
	Terminal Control Area (TCA or MTCA)	10
	Upper Flight Information Region (UIR)	11
	Mode C Defined Areas	12
	Other	13

Enumeration Name	Enumerator Description	Values
	Not Defined	14
AirspaceRestrictionType		
	Alert	0
	Caution	1
	Danger	2
	Military Operations Area	3
	Prohibited	4
	Restricted	5
	Temporary Reserved Airspace	6
	Training	7
	Warning	8
	Not Defined	9
AirspaceType		
	Class C Airspace (was ARSA within the USA)	0
	Control Area - ICAO Designation (CTA)	1
	Terminal Control Area - ICAO Desig (TMA or TCA)	2
	Radar Zone or Radar Area (was TRSA in the USA)	3
	Class B Airspace (was TCA within the USA)	4
	Class D Airspace in USA/Control Zone for ICAO (CTR)	5
	Advisory Area (ADA or UDA)	6
	Air Defense Identification Zone (ADIZ)	7
	Air Route Traffic Control Center (ARTCC)	8
	Area Control Center (ACC)	9
	Buffer Zone (BZ)	10

Enumeration Name	Enumerator Description	Values
	Control Area (CTA/UTA)/Special Rules Area (SRA - UK)	11
	Ctrl/Special Rules/Military Traffic Zone	12
	Ocean Control Area (OCA)	13
	Radar Area	14
	Terminal Control Area (TCA or MTCA)	15
	Mode C Defined Areas	16
	Other	17
	Not Defined	18
AirwayLevel		
	All Altitudes	0
	High Level Airway	1
	Low Level Airway	2
	Not Defined	3
AlternateMinimumType		
	Alternate Minimum Not Standard	0
	Alternate Minimum Not Authorized	1
	Not Defined	2
AltitudeDescription		
	At or above Alt1	0
	At or below Alt1	1
	At Alt1	2
	Between two altitudes	3
	At or above Alt2	4
	At Alt1 & Glideslope altitude Alt2	5

Enumeration Name	Enumerator Description	Values
	At or above Alt1 & Glideslope Alt Alt2	6
	At Alt1 & Glideslope Intercept Alt2	7
	At or above Alt1 & GS Intercept Alt2	8
	At or above Alt1 & Vertical Angle Alt2	9
	As assigned	10
	Recommended altitude	11
	Glideslope intercept altitude in Alt2	12
	Not Defined	13
AltitudeReference		
	Above Mean Sea Level	0
	Above Ground Level	1
	By Notam	2
	Altitude not limited	3
	Not Defined	4
AltitudeType		
	Feet above sea level	0
	Radar altimeter	1
	Missed approach point	2
	Transition level	3
	Not Defined	4
AltitudeUnit		
	Flight Level	0
	Meters	1
	Feet	2

Enumeration Name	Enumerator Description	Values
	Not Defined	3
ApproachPerformance		
	Not defined	0
ApproachRouteType		
	Approach Transition	0
	Localizer/Backcourse Approach	1
	Flight Management System Approach	2
	Instrument Guidance System (IGS) Approach	3
	Instrument Landing System (ILS) Approach	4
	Ground Based Augmentation Sys/GLS Approach	5
	Satellite Based Augmentation Sys Approach	6
	Localizer Only (LOC) Approach	7
	Microwave Landing System (MLS) Approach	8
	Non Directional Beacon (NDB) Approach	9
	Global Positioning System (GPS) Approach	10
	Area Navigation (RNAV) Approach	11
	Tacan Approach	12
	Simplified Directional Facility Approach	13
	VOR Approach	14
	Microwave Landing System Type A Approach	15
	Localizer Directional Aid (LDA) Approach	16
	Microwave Landing System Type B & C Approach	17
	Missed Approach	18

Enumeration Name	Enumerator Description	Values
	ILS Back Course Approach	19
	ILS Cat II Approach	20
	VORDME/VORTAC Approach	21
	VOR Circling Approach	22
	NDB Circling Approach	23
	RNAV (GPS) Non-Precision Approach	24
	ILS Cat III Approach	25
	LAAS-GPS/GLS (PAPP record required)	26
	WAAS-GPS (PAPP record required)	27
	RNAV (GPS) Overlay Approach	28
	PAR Approach	29
	NDB/DME Approach	30
	VOR (Based on VORDME or VORTAC) Approach	31
	MLS Cat II Approach	32
	ADF Approach	33
	SDF Approach	34
	MLS Cat III Approach	35
	RNAV (GPS) Precision Approach (Other)	36
	ILS Localizer only Circling Approach	37
	ILS Back Course Circling Approach	38
	Not Defined	39
ArcSegmentDerivation		
	Distance and Bearing	0
	End Coordinates	1

Enumeration Name	Enumerator Description	Values
	Derived by Plotted Coordinates	2
	Not Defined	3
ApproachSegmentType		
	Straight-In Approach	0
	Not Defined	1
AssociatedMarkerType		
	Inner Marker Beacon	0
	Middle Marker Beacon	1
	Outer Marker Beacon	2
	Back Marker Beacon	3
	Not Defined	4
AssociatedNavaid		
	Locator	0
	Non-Locator Navaid	1
	Not Defined	2
AtsRouteSegmentType		
	End of Continuous ATS route procedure	0
	Uncharted A-Route intersection	1
	Not Defined	2
BoundaryCode		
	USA	0
	Canada and Alaska	1
	Pacific	2
	Latin America	3

Enumeration Name	Enumerator Description	Values
	South America	4
	South Pacific	5
	Europe	6
	Eastern Europe	7
	Middle East-South Asia	8
	Africa	9
	Not Defined	10
BoundaryShape		
	Arc by edge	0
	Circle	1
	Great Circle	2
	Rhumb Line	3
	Counter Clockwise ARC	4
	Clockwise ARC	5
	Point (without radius or bearing)	6
	Generalized	7
	Not Defined	8
CivilMilitaryType		
	CIVIL	0
	MILITARY	1
	CIVIL/MILITARY	2
	CIVIL - MINOR OR NO FACILITIES	3
	MILITARY - MINOR OR NO FACILITIES	4
	PRIVATE	5

Enumeration Name	Enumerator Description	Values
	Not Defined	6
ClearanceStatus		
	Airport of Entry	0
	Landing Rights Airport	1
	Airport of Entry/Landing Rights Airport	2
	Not Defined	3
ClimbRateType		
	Minimum Climb Rate	0
	ATC Climb Rate	1
	Not Defined	2
CommsDetails		
	Air/Ground	0
	VHF Direction Finding Service	1
	Remote Communications Air to Ground	2
	Language other than English	3
	Military Use Frequency	4
	Pilot Controlled Light	5
	Remote Communications Outlet	6
	Not Defined	7
CommsEncryption		
	Off	0
	Not Defined	1
CommsFlightType		
	IFR Flight	0

Enumeration Name	Enumerator Description	Values
	VFR Flight	1
	Oceanic FIR/UIR	2
	Other FIR/UIR	3
	Not Defined	4
CommsType		
	Area Control Center	0
	Airlift Command Post	1
	Approach Control	2
	Arrival Control	3
	Automatic Terminal Info Service	4
	Automatic Weather Observing Service	5
	Clearance Delivery	6
	Clearance, Pre-Taxi	7
	Control Area (Terminal)	8
	Control	9
	Departure Control	10
	Director (Approach Control Radar)	11
	Enroute Flight Advisory Service	12
	Emergency	13
	Flight Service Station	14
	Ground Comm Outlet	15
	Ground Control	16
	Gate Control	17
	Helicopter Frequency	18

Enumeration Name	Enumerator Description	Values
	Information	19
	Multicom	20
	Operations	21
	Radio	22
	Radar	23
	Remote Flight Service Station	24
	Ramp/Taxi Control	25
	Airport Radar Service Area	26
	Terminal Control Area (TCA)	27
	Terminal Control Area (TMA)	28
	Terminal	29
	Terminal Radar Service Area	30
	Transcriber Weather Broadcast	31
	Tower, Air Traffic Control	32
	Upper Area Control	33
	Unicom	34
	Volmet	35
	Ground Control Approach	36
	Parameters (French Radio)	37
	Common Traffic Advisory Frequency	38
	Air/Ground	39
	Approach/Departure Control	40
	Air Route Traffic Control Center	41
	Ground Control/Clearance Delivery	42

Enumeration Name	Enumerator Description	Values
	Command Post	43
	Pilot to Dispatcher	44
	Pilot to Metro Service	45
	Airport Advisory Service	46
	Air Route Traffic Control	47
	Preflight	48
	Single Frequency Approach	49
	Miscellaneous	50
	Centralized Approach Control	51
	Aerodrome Flight Info Service	52
	Remote Communications Outlet	53
	Automated Surface Observation System	54
	Flight Communications Center	55
	Flight Operations Center	56
	Airport Weather Information Broadcast	57
	Not Defined	58
ComponentType		
	Locator	0
	Dme	1
	Localizer	2
	Glide Slope	3
	Back Course Marker	4
	Inner Marker	5
	Middle Marker	6

Enumeration Name	Enumerator Description	Values
	Outer Marker	7
	MLS Localizer	8
	MLS DME	9
	Not Defined	10

Enumeration Name	Enumerator Description	Value
CountryEntry		
	Unidentified	0
	Afghanistan	1
	Africa - Central	2
	Africa - East	3
	Africa - South	4
	Africa - West	5
	Alaska	6
	Albania	7
	Algeria	8
	American Samoa	9
	American Samoa/Samoa	10
	Andorra	11
	Andorra/Spain	12
	Angola	13
	Anguilla Island	14
	Antarctica	15
	Antigua and Barbuda	16
	Argentina	17
	Argentina/Antarctica	18
	Armenia	19
	Armenia/Azerbaijan/Georgia/Russian Federation	20
	Armenia/Azerbaijan/Kazakhstan/Turkmenistan/Uzbekistan	21
	Aruba	22

Enumeration Name	Enumerator Description	Value
	Ashmore and Cartier Island	23
	Asia - Far East	24
	Asia - Middle East	25
	Asia - South	26
	Australia	27
	Australia associated islands	28
	Austria	29
	Austria/Liechtenstein	30
	Azerbaijan	31
	Azerbaijan/Kazakhstan/Russian Federation	32
	Bahamas	33
	Bahrain	34
	Bahrain/Iraq-Saudi Arabia Neutral Zone	35
	Baker Island	36
	Bangladesh	37
	Barbados	38
	Bassas	39
	Belarus	40
	Belarus/Russian Federation	41
	Belgium	42
	Belize	43
	Benin	44
	Bermuda	45
	Bhutan	46

Enumeration Name	Enumerator Description	Value
	Bolivia	47
	Bosnia and Herzegovina	48
	Botswana	49
	Bouvet Island	50
	Brazil	51
	British Indian Ocean Territory	52
	British Virgin Islands	53
	Brunei Darussalam	54
	Bulgaria	55
	Burkina Faso	56
	Burma (Myanmar)	57
	Burundi	58
	Cambodia	59
	Cameroon	60
	Canada	61
	Canada - Uplands CFB	62
	Canada - Weather Centres	63
	Cape Verde	64
	Cayman Islands	65
	Central African Republic	66
	Central America/Mexico/West Caribbean	67
	Chad	68
	Chile	69
	Chile/Antarctica	70

Enumeration Name	Enumerator Description	Value
	China	71
	Christmas Island	72
	Clipperton Island	73
	Cocos (Keeling) Island	74
	Colombia	75
	Comoros	76
	Congo	77
	Continental China	78
	Cook Islands	79
	Coral Sea Islands	80
	Costa Rica	81
	Croatia	82
	Cuba	83
	Cyprus	84
	Czech Republic	85
	Democratic People's Republic of Korea	86
	Democratic Republic of the Congo	87
	Denmark	88
	Denmark and associated islands	89
	Djibouti	90
	Dominica	91
	Dominican Republic	92
	East Caribbean	93
	East Timor	94

Enumeration Name	Enumerator Description	Value
	Ecuador	95
	Egypt	96
	El Salvador	97
	Equatorial Guinea	98
	Eritrea	99
	Estonia	100
	Ethiopia	101
	Europa Island	102
	Europe - North	103
	Europe - South	104
	Europe - West	105
	Ex-URSS region	106
	Falklands Islands	107
	Faroe Islands	108
	Federal Republic of Yugoslavia	109
	Fiji	110
	Fiji and surrounding islands	111
	Finland	112
	France	113
	France and associated islands	114
	French Antilles	115
	French Guyana	116
	French Polynesia	117
	French Polynesia/Pitcairn Island	118

Enumeration Name	Enumerator Description	Value
	French Southern and Antarctic Islands	119
	Gabon	120
	Gambia	121
	Gaza Strip	122
	Georgia	123
	Germany	124
	Ghana	125
	Gibraltar	126
	Glorioso Islands	127
	Greece	128
	Greenland	129
	Grenada	130
	Guadeloupe	131
	Guam	132
	Guatemala	133
	Guernsey	134
	Guinea	135
	Guinea-Bissau	136
	Guyana	137
	Haiti	138
	Hawaii	139
	Honduras	140
	Hong Kong	141
	Hong Kong/Paracel Islands	142

Enumeration Name	Enumerator Description	Value
	Howland Island	143
	Hungary	144
	Iceland	145
	Iles Wallis et Futuna	146
	India	147
	Indonesia	148
	Indonesia/East Timor	149
	Iran	150
	Iraq	151
	Iraq/Iraq-Saudi Arabia Neutral Zone	152
	Iraq-Saudi Arabia Neutral Zone	153
	Ireland	154
	Isle of Man	155
	Israel	156
	Israel/Gaza Strip	157
	Italy	158
	Italy and enclaved territories	159
	Ivory Coast	160
	Jamaica	161
	Jamaica and surrounding islands	162
	Jan Mayen	163
	Japan	164
	Jarvis Island	165
	Jersey	166

Enumeration Name	Enumerator Description	Value
	Johnston Atoll	167
	Jordan	168
	Jordan/The West Bank	169
	Juan de Nova Island	170
	Kazakhstan	171
	Kazakhstan/Kyrgyzstan/Uzbekistan	172
	Kazakhstan/Tajikistan/Turkmenistan/Uzbekistan	173
	Kenya	174
	Kingman Reef	175
	Kiribati	176
	Kiribati and Line Island	177
	Kiribati/Jarvis Island	178
	Kiribati/Tuvalu	179
	Kuwait	180
	Kyrgyzstan	181
	Laos People's Democratic Republic	182
	Latvia	183
	Lebanon	184
	Lesotho	185
	Liberia	186
	Libyan Arab Jamahiriya	187
	Liechtenstein	188
	Lithuania	189
	Luxembourg	190

Enumeration Name	Enumerator Description	Value
	Macao	191
	Madagascar	192
	Madagascar and surrounding islands	193
	Malawi	194
	Malaysia	195
	Malaysia/Brunei Darussalam	196
	Maldives	197
	Mali	198
	Malte	199
	Mariana Islands	200
	Mariana Islands (including Guam)	201
	Marshall Islands	202
	Martinique	203
	Mauritania	204
	Mauritius	205
	Mayotte	206
	Mexico	207
	Mexico and surrounding islands	208
	Micronesia	209
	Micronesia/Palau	210
	Midway Islands	211
	Monaco	212
	Mongolia	213
	Montserrat	214

Enumeration Name	Enumerator Description	Value
	Morocco	215
	Morocco/Western Sahara	216
	Mozambique	217
	Mozambique and surrounding islands	218
	Namibia	219
	Nauru	220
	Navassa Island	221
	Nepal	222
	Netherlands	223
	Netherlands Antilles	224
	Netherlands Antilles/Aruba	225
	New Caledonia	226
	New Zealand	227
	New Zealand/Antarctica	228
	Nicaragua	229
	Niger	230
	Nigeria	231
	Niue Island	232
	Norfolk Island	233
	Norway	234
	Norway and associated territories	235
	Oceania - East	236
	Oceania - North-East	237
	Oceania - West	238

Enumeration Name	Enumerator Description	Value
	Oman	239
	Pacific	240
	Pakistan	241
	Palau	242
	Panama	243
	Papua New Guinea	244
	Paracel Islands	245
	Paraguay	246
	Peru	247
	Philippines	248
	Philippines/Spratly Islands	249
	Pitcairn Island	250
	Poland	251
	Portugal	252
	Puerto Rico	253
	Puerto Rico and surrounding Caribbean islands	254
	Qatar	255
	Republic of Korea	256
	Republic of Moldova	257
	Reunion	258
	Romania	259
	Russian Federation	260
	Rwanda	261
	Saint Lucia	262

Enumeration Name	Enumerator Description	Value
	Saint Vincent and the Grenadines	263
	San Marino	264
	Sao Tome and Principe	265
	Saudi Arabia	266
	Senegal	267
	Seychelles	268
	Sierra Leone	269
	Singapore	270
	Slovakia	271
	Slovenia	272
	Solomon Islands	273
	Somalia	274
	South Africa	275
	South Africa and surrounding islands	276
	South America	277
	Spain	278
	Spain - Canary Islands	279
	Spratly Islands	280
	Sri Lanka	281
	St. Kitts and Nevis	282
	St.Helena and Ascension Island	283
	St.Pierre and Miquelon	284
	Sudan	285
	Suriname	286

Enumeration Name	Enumerator Description	Value
	Svalbard	287
	Swaziland	288
	Sweden	289
	Switzerland	290
	Syrian Arab Republic	291
	Taiwan	292
	Tajikistan	293
	Thailand	294
	The former Yugoslav Republic of Macedonia	295
	The West Bank	296
	Togo	297
	Tokelau	298
	Tonga	299
	Trinidad and Tobago	300
	Tromelin Island	301
	Tunisia	302
	Turk and Caicos Islands	303
	Turkey	304
	Turkmenistan	305
	Tuvalu	306
	Uganda	307
	Ukraine	308
	Ukraine/Russian Federation	309
	United Arab Emirates	310

Enumeration Name	Enumerator Description	Value
	United Kingdom	311
	United Kingdom and associated islands	312
	United Republic of Tanzania	313
	United States	314
	United States (Central North-East)	315
	United States (Central North-West)	316
	United States (Central South)	317
	United States (North-East)	318
	United States (North-West)	319
	United States (South-East)	320
	United States (South-West)	321
	Uruguay	322
	US territories - North Pacific Ocean	323
	Uzbekistan	324
	Vanuatu	325
	Vatican City	326
	Venezuela	327
	Viet Nam	328
	Virgin Islands	329
	Virgin Islands/British Virgin Islands	330
	Wake Island	331
	Western Sahara	332
	Western Samoa	333
	Yemen	334

Enumeration Name	Enumerator Description	Value
	Zambia	335
	Zimbabwe	336

Enumeration Name	Enumerator Description	Value
CruiseTable		
	ICAO Standard Cruise Table	0
	Exception to ICAO Standard Cruise Table	1
	Modified Cruise Table	2
	Exception to Modified Cruise Table	3
	Not Defined	4
DataSource		
	ARINC 424	0
	DAFIF	1
DataTransferStatus		
	Error Data Lost	0
	Data Transfer Completed	1
	Data Transfer In Progress	2
DayOfWeek		
	Monday	0
	Tuesday	1
	Wednesday	2
	Thursday	3
	Friday	4
	Saturday	5
	Sunday	6
	Not Defined	7
Direction		
	East	0

Enumeration Name	Enumerator Description	Value
	West	1
	Not defined	2
DirectionRestriction		
	Forward Direction Route Coded	0
	Backward Direction Route Coded	1
	No Direction Restriction	2
	Not Defined	3
DistanceDescription		
	Out to Specified Distance	0
	Beyond Specified Distance	1
	Not Defined	2
EmissionType		
	A0 - Unmodulated Carrier	0
	A1 - Carrier Keyed	1
	A2 - Tone Keyed Modulation	2
	Not Defined	3
EnrouteAirwayRouteType		
	Airline Airway (Tailored Data)	0
	Control	1
	Direct Route	2
	Helicopter Airway	3
	Officially Designated Airway	4
	RNAV Airway	5
	Undesignated ATS Route	6

Enumeration Name	Enumerator Description	Value
	Not Defined	7
ExclusionIndicator		
	All Altitudes in Both Directions Restricted	0
	All Altitudes in Backward Direction Restricted	1
	All Altitudes in Forward Direction Restricted	2
	Not an all altitudes restriction	3
	Not Defined	4
FacilityRecordType		
	Airport	0
	VHF Navaid	1
	NDB Navaid	2
	Terminal NDB	3
	Not Defined	4
FirUirType		
	FIR	0
	UIR	1
	Combined FIR/UIR	2
	Not Defined	3
FixDetails		
	Initial Approach Fix	0
	Intermediate Approach Fix	1
	Initial Approach Fix with Holding	2
	Initial Approach Fix with Final Approach Crse Fix	3
	Final End Point Fix	4

Enumeration Name	Enumerator Description	Value
	Published/Database Final Approach Fix	5
	Holding Fix	6
	Final Approach Course Fix	7
	Published Missed Approach Point Fix	8
	Not Defined	9
FixFunction		
	Unnamed Stepdown Fix After Final Approach Fix	0
	Unnamed Stepdown Fix Before Final Approach Fix	1
	ATC Compulsory Waypoint	2
	Oceanic Gateway Waypoint	3
	First Leg of Missed Approach Procedure	4
	Path Point Fix	5
	Named Stepdown Fix	6
	Not Defined	7
FixRecordType		
	Airport	0
	VHF Navaid	1
	NDB Navaid	2
	Terminal NDB	3
	Enroute Waypoint	4
	Airport Waypoint	5
	Heliport Waypoint	6
	Runway	7
	SID	8

Enumeration Name	Enumerator Description	Value
	STAR	9
	Navaid (VHF or NDB)	10
	Waypoint (Terminal or Enroute)	11
	ATS Route	12
	Not Defined	13
FixPointType		
	VOR (navaid)	0
	VORTAC (navaid)	1
	TACAN (navaid)	2
	VORDME (navaid)	3
	NDB (navaid)	4
	NDBDME (navaid)	5
	DME (navaid)	6
	Atlantic (ATS Route)	7
	Bahama (ATS Route)	8
	Corridor (ATS Route)	9
	Advisory (ADR) (ATS Route)	10
	Direct, Control Area Routes (ATS Route)	11
	Military (ATS Route)	12
	North American (NAR) (ATS Route)	13
	Oceanic (ATS Route)	14
	RNAV (ATS Route)	15
	Substitute, Canadian Control Area Tracks (ATS Route)	16
	TACAN (ATS Route)	17

Enumeration Name	Enumerator Description	Value
	Airway (ATS Route)	18
	Not Defined	19
FixRecordType		
	Airport	0
	VHF Navaid	1
	NDB Navaid	2
	Terminal NDB	3
	Enroute Waypoint	4
	Airport Waypoint	5
	Heliport Waypoint	6
	Runway	7
	Not Defined	8
FixType		
	Final Approach Fix	0
	Initial and Final Approach Fix	1
	Final Approach Course Fix	2
	Intermediate Approach Fix	3
	Off-Route Intersection	4
	Initial Approach Fix	5
	Final Approach Course Fix at Initial Approach Fix	6
	Final Approach Course Fix at Interm. Approach Fix	7
	Missed Approach Fix	8
	Initial Approach Fix and Missed Approach Fix	9
	Oceanic Entry/Exit Waypoint	10

Enumeration Name	Enumerator Description	Value
	Unnamed Stepdown Fix	11
	Named Stepdown Fix	12
	FIR/UIR or Controlled Airspace Intersection	13
	Lat/Long Intersection, Full Degree of Latitude	14
	Lat/Long Intersection, Half Degree of Latitude	15
	Not Defined	16
FlyOverType		
	Flyover -End SID/STAR Rte, APCH Transition/Final Approach	0
	End of Terminal Procedure Route Type	1
	Uncharted Airway Intersection	2
	Fly-Over Waypoint (overfly)	3
	Not Defined	4
FrequencyClass		
	UHF/VHF	0
	LF/MF	1
	Not defined	2
FrequencyDirectionRestriction		
	East direction only	0
	West direction only	1
	Both East and West	2
	Not defined	3
FrequencyType		
	Aerodrome Traffic Frequency	0
	Common Traffic Advisory Frequency	1

Enumeration Name	Enumerator Description	Value
	Mandatory Frequency	2
	Secondary Frequency	3
	Air/Ground	4
	Discrete Frequency	5
	Air/Air	6
	Not Defined	7
GlsStationType		
	LAAS/GLS	0
	SCAT-1	1
	Not defined	2
GpsFmsIndicator		
	No GPS or FMS Overlay Authorized	0
	GPS Overlay, Navaids Operating & Monitored	1
	GPS Overlay, Navaids Installed/Not Monitored	2
	GPS Overlay, Title includes GPS	3
	FMS Overlay Authorized	4
	FMS and GPS Overlay Authorized	5
	Not Defined	6
GuardTransmit		
	Receive Voice Communications	0
	Transmit Voice Communications	1
	Receive and Transmit Voice Comms	2
	Not Defined	3
HoldingPatternType		

Enumeration Name	Enumerator Description	Value
	High Altitude	0
	Low Altitude	1
	SID	2
	STAR	3
	Approach	4
	Missed Approach	5
	All Altitude	6
	Not Defined	7
IlsBackCourse		
	Usable	0
	Unusable	1
	Restricted	2
	Not Defined	3
LandingAidCategory		
	ILS Localizer Without Glideslope	0
	CAT I	1
	CAT II	2
	CAT III	3
	IGS	4
	LDA With Glideslope	5
	LDA Without Glideslope	6
	SDF With Glideslope	7
	SDF Without Glideslope	8
	Not Defined	9

Enumeration Name	Enumerator Description	Values
LightingSystem		
	Unidentified	0
	PCL - Pilot Controlled Lighting	1
	SF - Sequenced Flashing Lights	2
	TDZL - Touchdown Zone Lighting	3
	CL - Centerline Lighting System	4
	HIRL - High Intensity Runway Lights	5
	MIRL - Medium Intensity Runway Lighting System	6
	LIRL - Low Intensity Runway Lighting System	7
	RAIL - Runway Alignment Lights	8
	REIL - Runway End Identifier Lights	9
	A - ALSF-2	10
	A1 - ALSF-1	11
	A2 - SALS or SALSF	12
	A3 - SSALR	13
	A4 - MALS and MALSF or SSALS and SSALF	14
	A5 - MALSR	15
	AF - Overrun Centerline	16
	AI - Centerline and Bar	17
	B - US Configuration (b)	18
	BE - Hong Kong Curve	19
	BF - Center row	20
	BG - Left Center Row	21
	BN - Former NATO Standard ©	22

Enumeration Name	Enumerator Description	Values
	BO - Center Row	23
	BP - NATO standard	24
	BQ - Center and Double Row	25
	BR - Portable Approach	26
	BS - Center Row	27
	G - Helicopter Approach Lighting System (HALS)	28
	J2 - CALVERT II (BRITISH)	29
	E - Two Parallel row	30
	F - Left Row (High Intensity)	31
	I - Air Force Overrun	32
	J - CALVERT I (BRITISH)	33
	M - Single Row Centerline	34
	N - Narrow Multi-cross	35
	O - Centerline High Intensity	36
	Q - Alternate Centerline and Bar	37
	S - Cross	38
	T - Center Row	39
	U - Singapore Centerline	40
	X - Centerline 2 Crossbars	41
	ODALS - Omni-directional Approach Lighting System	42
	V(VASI) - Visual Approach Slope Indicator	43
	V1 (T-VASI) - T-Visual Approach Slope Indicator	44
	V2 (PVASI) - Pulsating Visual Approach Slope Indicator	45
	V3 (JUMBO) - VASI with a TCH to accommodate long bodied or jumbo aircraft	46

Enumeration Name	Enumerator Description	Values
	V4 - Tri-color Arrival Approach (TRICOLOR)	47
	V5 (APAP) - Alignment of Elements System	48
	RETIL - Rapid Exit Taxiway Indicator Lighting	49
	PAPI - Precision Approach Path Indicator	50
	OLS - Optical landing System	51
	WAVEOFF	52
	PORTABLE	53
	FLOODS	54
	LIGHTS	55
	LCVASI - Low Cost Visual Approach Slope Indicator	56
	Lighting Provisional3	57
	Lighting Provisional4	58
	Lighting Provisional5	59
	Lighting Provisional6	60
	Lighting Provisional7	61
	Lighting Provisional8	62
	Lighting Provisional9	63
	Lighting Provisional10	64

Enumeration Name	Enumerator Description	Values
LocationReference		
	Prior to threshold/approach end (near end)	0
	On the runway	1
	On the overrun (far end)	2
	Not defined	3
MagneticTrueIndication		
	Magnetic	0
	True	1
	Mixed Magnetic and True	2
	Other than Magnetic or True	3
	Not Defined	4
MarkerPower		
	Low	0
	High	1
	Not Defined	2
MarkerShape		
	Elliptical	0
	Bone	1
	Not Defined	2
MarkerType		
	Inner Marker Beacon	0
	Middle Marker Beacon	1
	Outer Marker Beacon	2
	Back Marker Beacon	3

Enumeration Name	Enumerator Description	Values
	Bone Marker Beacon	4
	Fan Marker Beacon	5
	Low Power Fan Marker Beacon	6
	Z Marker Beacon	7
	Not Defined	10
MaximumTirePressure		
	High - No Limit	0
	Medium - Limited to 217 psi	1
	Low - Limited to 145 psi	2
	Very Low - Limited to 73 psi	3
	Not Defined	4
MilitaryRouteType		
	Instrument Route	0
	Visual Route	1
	Slow Route	2
	Not Defined	3
MlsCollocation		
	DME Collocated With MLS Azimuth	0
	DME Collocated With MLS Elevation	1
	DME Non Collocated With MLS	2
	Not Defined	3
Modulation		
	Amplitude Modulated Frequency	0
	Frequency Modulated Frequency	1

Enumeration Name	Enumerator Description	Values
	Not Defined	2
MonitoredFrequency		
	VHF Emergency Frequency 121.5	0
	UHF Emergency Frequency 243.0	1
	VHF/UHF Emergency Frequencies	2
	VHF 121.5 and VHF/UHF Emergency Freq	3
	UHF 243.0 and VHF/UHF Emergency Freq	4
	VHF 121.5 and UHF 243.0 Emergency Freq	5
	Not Defined	6
NameFormatType		
	Abeam Fix	0
	Bearing and Distance Fix	1
	Airport Name as Fix	2
	FIR Fix	3
	Phonetic Letter Name Fix	4
	Airport Ident as Fix	5
	Latitude/Longitude Fix	6
	Multiple Word Name Fix	7
	Navaid Ident as Fix	8
	Published Five-Letter Name Fix	9
	Published Less Than 5-Letter Fix	10
	Published More Than 5-Letter Fix	11
	Airport/Runway Related Fix	12
	UIR Fix	13

Enumeration Name	Enumerator Description	Values
	Official 5-letter Localizer Name	14
	Unofficial 5-letter Localizer	15
	Not Defined	16
NavaidCollocation		
	Collocated Navaid	0
	Non Collocated Navaid	1
	DME Collocated With ILS Localizer	2
	DME Collocated With ILS Glide Slope	3
	DME Non Collocated With ILS	4
	DME Collocated With MLS Azimuth	5
	DME Collocated With MLS Elevation	6
	DME Non Collocated With MLS	7
	Not Defined	8
NavaidRangePower		
	Terminal	0
	Low Altitude	1
	High Altitude	2
	200 Watts or More	3
	50 to 1999 Watts	4
	25 to Less Than 50 Watts	5
	Less Than 25 Watts	6
	Not Defined	7
NavaidStatus		
	In-Service	0

Enumeration Name	Enumerator Description	Values
	Out of Service	1
	On Test	2
	Not Defined	3
NavaidType		
	VOR	0
	DME	1
	VOR/DME	2
	TACAN - Channels 17-59 and 70-126	3
	Military TACAN - Channels 1-16 and 60-69	4
	VORTAC	5
	ILS/DME	6
	ILS/TACAN	7
	MLS/Narrow Spectrum DME	8
	MLS/Precision DME	9
	NDB	10
	NDB-DME	11
	SABH	12
	Marine Beacon	13
	VOR Test Station	14
	Not Defined	15
NotamSystem		
	FAA/DOD Full Coverage	0
	FAA/DOD Partial Coverage	1
	US Army Flight Operations Detachment	2

Enumeration Name	Enumerator Description	Values
	German Federal Armed Forces	3
	Not Defined	4
ObjectStatus		
	Station is alive / Reset status to alive	0
	Station is killed / Set status to killed	1
	Leave station status as it is	2
ObjectType		
	Airport	0
	AirspaceBoundary	1
	AirwayRestriction	2
	Approach	3
	Comms	4
	ControlledAirspace	5
	EnrouteAirway	6
	FirUir	7
	Gate	8
	Gls	9
	Helipad	10
	Heliport	11
	HoldingPattern	12
	Ils	13
	Marker	14
	Mls	15
	Msa	16

Enumeration Name	Enumerator Description	Values
	Navaid	17
	OffRouteTerrainClearanceAlt	18
	PresetSite	19
	RestrictiveAirspace	20
	Runway	21
	Sid	22
	SpecialUseAirspace	23
	Star	24
	SupplementalTerminalData	25
	TerminalProcedureClimb	26
	TerminalProcedureFeederRoute	27
	TerminalProcedureMinima	28
	VfrRoute	29
	Waypoint	30
ObjectUpdateType		
	Object has not been updated	0
	Object has been deleted from database	1
	Object has changed	2
	Object has been added to database	3
	Object status has changed	4
OperatingHours		
	24 Hours	0
	Sunrise to Sunset	1
	No Hours Listed	2

Enumeration Name	Enumerator Description	Values
	Refer to Remarks	3
	Unknown Hours	4
PadShape		
	Rectangular	0
	Circular	1
	Not Defined	2
ParachuteJumpAreaType		
	Bearing/Distance to a point	0
	A point	1
	Bearing/Distance to an area	2
	Geographic area (defined by coords)	3
	Area defined by 2 brgs & 2 distances	4
	Multiple areas defined by brg/distance	5
	Unspecified, Call Tower	6
	Not Defined	7
PathDataSelector		
	Not defined	0
PathTermination		
	Initial Fix (IF)	0
	Track to a Fix (TF)	1
	Course to a Fix (CF)	2
	Direct to a Fix (DF)	3
	Fix to an Altitude (FA)	4
	Track from a Fix from a Distance (FC)	5

Enumeration Name	Enumerator Description	Values
	Track from a Fix to DME Distance (FD)	6
	From a Fix to Manual Termination (FM)	7
	Course to an Altitude (CA)	8
	Course to a DME Distance (CD)	9
	Course to an Intercept (CI)	10
	Course to a Radial Termination (CR)	11
	Constant Radius Arc (RF)	12
	Arc to a Fix (AF)	13
	Heading to Altitude Termination (VA)	14
	Heading to DME Distance Termin. (VD)	15
	Heading to an Intercept (VI)	16
	Heading to a Manual Termination (VM)	17
	Heading to a Radial Termination (VR)	18
	Procedure Turn (PI)	19
	Crse Reversal Altitude Termination (HA)	20
	Crse Reversal Single Circuit Term. (HF)	21
	Course Reversal Manual Termination (HM)	22
	Not Defined	23
PathTurnDirection		
	Left	0
	Right	1
	Not Defined	2
PathType		
	Arrival Route	0

Enumeration Name	Enumerator Description	Values
	Departure Route	1
	Holding Pattern	2
	Part of a Terminal Traffic Pattern	3
	VFR Transition	4
	Other	5
	Not Defined	6
PavementEvaluationMethod		
	Technical	0
	By Experience Using Pavement	1
	Not Defined	2
PavementSubgradeCategory		
	High	0
	Medium	1
	Low	2
	Ultra-Low	3
	Not Defined	4
PavementType		
	Rigid	0
	Flexible	1
	Water	2
	Not Defined	3
PointReportingType		
	Compulsory Reporting Point	0
	Non-Compulsory Reporting Point	1

Enumeration Name	Enumerator Description	Values
	Not Defined	2
PointFunction		
	Alternate Entry Point	0
	Alternate Exit Point	1
	Alternate Entry/Exit Point	2
	Entry Point (Starting Point)	3
	Turning Point	4
	Exit Point (Ending Point)	5
	Not Defined	6
PreferredRouteType		
	N-Ameri Rtes for N-Atlantic Traffic - Common	0
	Preferential Routes	1
	Pacific Oceanic Transition Routes (PACOTS)	2
	TACAN Routes (Australia)	3
	N-Ameri Rtes for N-Atlantic Trffic -Noncommon	4
	Preferred/Preferential Overflight Routes	5
	Preferred Routes	6
	Traffic Orientation System Routes (TOS)	7
	Tower Enroute Control Routes (TEC)	8
	Not Defined	9
PresetSiteType		
	Gate	0
	CAL Site	1
	Hold	2

Enumeration Name	Enumerator Description	Values
	Takeoff	3
	Opposite End	4
	Not Defined	5
	Ramp	6
	Parking Spot	7
	Taxi	8
PrimaryTimeCode		
	Active Continuously Including Holidays	0
	Active Continuously Excluding Holidays	1
	Active Non-Continuously, see Cont. Rec	2
	Active Times Announced by NOTAM	3
	Not Defined	4
RadioClassCode		
	Non-Directional Beacon, 50-2000 Watts	0
	Interference-Free 40NM up to 18000 feet	1
	Interference-Free 25NM up to 12000 feet	2
	Non-Directional Beacon, 50 Watts or less	3
	Non-Directional Beacon, 2000 Watts & up	4
	Interference-Free Service Varies	5
	Compass Locator, 25 Watts or less, 15NM	6
	Not Defined	7
RangeReliability		
	Terminal Within 25 nm	0
	Low Altitude - Within 40 nm	1

Enumeration Name	Enumerator Description	Values
	High Altitude - Within 130 nm	2
	Extended High Altitude - Beyond 130 nm	3
	Out of Service	4
	High Level	5
	Low Level	6
	High and Low Level	7
	RNAV	8
	Terminal	9
	Not Defined	10
RefuelingAltitudeDescription		
	At or above Altitude 1	0
	At or below Altitude 1	1
	Between Altitude 1 and 2	2
	At Altitude 1	3
	Not defined	4
RefuelingDirection		
	North	0
	South	1
	East	2
	West	3
	Northeast	4
	Northwest	5
	Southeast	6
	Southwest	7

Enumeration Name	Enumerator Description	Values
	Not defined	8
RefuelingOperationType		
	Anchor	0
	Track	1
	Anchor or Track	2
	Not defined	3
RefuelingPointType		
	Air refueling initial point	0
	Air refueling control point	1
	Navigation check point	2
	Exit point	3
	Entry point (anchors only)	4
	Anchor point (anchors only)	5
	Anchor pattern (anchors only)	6
	Not defined	7
RestrictionType		
	Altitude Exclusion	0
	Cruising Table Replacement	1
	Seasonal Restriction	2
	Note Restriction	3
	Not Defined	4
ReturnCode		
	Ok	0
	Fail	1

Enumeration Name	Enumerator Description	Values
	Not Found	2
	Request Pending	3
	Request In Progress	4
	Request Completed	5
	Unappropriate Container Type Ident	6
	Container Ownership Unappropriate	7
	Status unavailable	8
	User cancelled operation	9
	File name not specified	10
	Database Name not found	11
	Client already registered	12
	Client is not registered	13
	Client is unauthorized	14
	Request is not registered	15
	Request is unauthorized	16
	Duplicate Item	17
	No Associated Runway	18
	Kill command doesn't match Navaid component	19
	Unique Id doesn't match NavObject component	20
	NavObject already exists in the database	21
	Local area has not been defined	22
	Service unavailable in current LOF DLL	23
	Gaussian's Coefficient are unavailable	24
	Gaussian's Coefficients model are out of date	25

Enumeration Name	Enumerator Description	Values
	Theoretical Result. Computed with a magnetic model (WMM or IGRS)	26
	Accessing wrong magnetic model	27
	DataType mismatch	28
	Another client is already registered as editor	29
	Edition mode is not active	30
	Edition mode activated	31
	Edition mode deactivated	32
	The supported Interface is not implemented	33
	The requested service is not supported on server	34
	Null	250
	No key is defined	251
	Null Object	252
	Insertion Fail	253
	Removal Fail	254
	File Opened	255
	File Closed	256
	File Not Found	257
	Parsing In Progress	258
	Database Empty	259
	Container Owner Unappropriate	260
	No Object Found	261
	Key wrongly assigned	262
	Restriction Not Satisfied	263
	Copy Failed	264

Enumeration Name	Enumerator Description	Values
	Selected Nav Type does not exists	265
	The kill station command has been sent	266
RouteAltitudeDescription		
	At or above Altitude 1	0
	At or below Altitude 1	1
	Between Altitude 1 and 2	2
	At Altitude 1	3
	As Assigned	4
	Not defined	5
RouteQualifier1		
	DME required	0
	RNAV/E if applicable	1
	RNAV/F if applicable	2
	GPS required	3
	GPS required, DME/DME to RNP Not Authorized	4
	DME not required	5
	GPS or DME/DME to RNP required	6
	DME/DME required	7
	VOR/DME RNAV	8
	Not Defined	9
RouteQualifier2		
	Primary Missed Approach	0
	Secondary Missed Approach	1
	Engine Out Missed Approach	2

Enumeration Name	Enumerator Description	Values
	Procedure with Circle-to-Land Minimums	3
	Procedure with Straight-In Minimums	4
	Procedure Designed for Helicopter to Runway	5
	Not Defined	6
RouteQualifierType		
	RNAV, GPS required, DME/DME to RNP Not Auth.	0
	RNAV, GPS or DME/DME to RNP authorized	1
	Not Defined	2
RouteStatus		
	Open	0
	Closed	1
	Restricted	2
	Alternate	3
	Seasonal, Conditional	4
	Not defined	5
RouteType		
	SID	0
	STAR	1
	Approach	2
	Multiple	3
	Not Defined	4
RouteUse		
	Point-to-Point	0
	Area-to-Area	1

Enumeration Name	Enumerator Description	Values
	Not Defined	2
RoutingType		
	Designated Airway	0
	Direct to Fix	1
	Initial Fix	2
	Route via Fix	3
	Route via Fix not permitted	4
	Standard Instrument Departure	5
	Standard Terminal Arrival & Profile Descent	6
	Not Defined	7
RunwaySurfaceType		
	Asphalt, Asphaltic Concrete, Tar, Macadam	0
	Brick - Laid or Mortared	1
	Concrete	2
	Composite - 50 percent or more of runway is permanent	3
	Part concrete, asphalt, or bitumen-bound macadam	4
	Permanent - Surface type unknown	5
	Bituminous, tar or asphalt mixed in place, oiled	6
	Clay	7
	Composite - less than 50 percent of runway is permanent	8
	Coral	9
	Graded or rolled earth, grass on graded earth	10
	Grass or earth not graded or rolled	11
	Gravel	12

Enumeration Name	Enumerator Description	Values
	Ice	13
	Laterite	14
	Macadam - crushed rock water bound	15
	Membrane - plastic or other fiber material	16
	Mix in place using non-bituminous binders (eg: portland)	17
	Pieced steel planking	18
	Sand	19
	Snow	20
	Not Defined	21
RvsmIndicator		
	Entry/Exit	0
	Entry Only	1
	Exit Only	2
	RVSM Transition Waypoint	3
	RVSM on Airway or Stand Alone	4
	Not Defined	5
SegmentAltitudeDescription		
	At or above altitude specified	0
	At or below altitude specified	1
	As assigned	2
	At altitude specified	3
	Recommended altitude	4
	Not Defined	5
SegmentNavaidType		

Enumeration Name	Enumerator Description	Values
	VOR	0
	VOR-TAC	1
	TACAN	2
	VOR-DME	3
	NDB	4
	NDB-DME	5
	DME	6
	ILS Locator	7
	ILS DME	8
	ILS Localizer	9
	Waypoint	10
	MLS	11
	MLS-DME	12
	Not Defined	13
SegmentType		
	Starting Segment	0
	Next Segment	1
	Ending Segment	2
	Not Defined	3
ServerState		
	Off	0
	Initializing	1
	Online	2
	Partially Operational	3

Enumeration Name	Enumerator Description	Values
	Not Operational	4
	Not Responding	5
	Not Available	6
	Parsing in Progress	7
ServiceIndicator		
	Airport Advisory Service	0
	Community Aerodrome Radio Station	1
	Departure Service (not Control Unit)	2
	Flight Information Service	3
	Initial Contact	4
	Arrival Service (not Control Unit)	5
	Pre-Departure Clearance (Data Link)	6
	Aerodrome Flight Information Service	7
	Terminal Area Control (not Control Unit)	8
	Aeronautical Enroute Information Service	9
	Not Defined	10
ServiceProvider		
	Not Defined	0
SidRouteType		
	Engine Out SID	0
	SID Runway Transition	1
	SID or SID Common Route	2
	SID Enroute Transition	3
	RNAV SID Runway Transition	4

Enumeration Name	Enumerator Description	Values
	RNAV SID or RNAV SID Common Route	5
	RNAV SID Enroute Transition	6
	FMS SID Runway Transition	7
	FMS SID or SID Common Route	8
	FMS SID Enroute Transition	9
	Vector SID Runway Transition	10
	Vector SID Enroute Transition	11
	Not Defined	12
SignalEmission		
	Double Sideband (A3)	0
	Single Sideband, Reduced Carrier (A3A)	1
	Two Independent Sidebands (A3B)	2
	Single Sideband, Full Carrier (A3H)	3
	Single Sideband, Suppressed Carrier (A3J)	4
	Lower (single) Sideband, Carrier Unknown	5
	Upper (single) Sideband, Carrier Unknown	6
	Not Defined	7
SignalModulation		
	400 Hz	0
	1020 Hz	1
	Not Defined	2
SpeedUnit		
	TAS in Knots	0
	TAS in Mach	1

Enumeration Name	Enumerator Description	Values
	TAS in Kilometers/Hour	2
	Not Defined	3
StarRouteType		
	STAR Enroute Transition	0
	STAR or STAR Common Route	1
	STAR Runway Transition	2
	RNAV STAR Enroute Transition	3
	RNAV STAR or RNAV STAR Common Route	4
	RNAV STAR Runway Transition	5
	Profile Descent Enroute Transition	6
	Profile Descent or Prof. Desc. Common Route	7
	Profile Descent Runway Transition	8
	FMS STAR Enroute Transition	9
	FMS STAR or STAR Common Route	10
	FMS STAR Runway Transition	11
	Not Defined	12

Enumeration Name	Enumerator Description	Value
StateEntry		
	Unidentified	0
	Alabama	1
	Alaska	2
	Arizona	3
	Arkansas	4
	California	5
	Colorado	6
	Connecticut	7
	Delaware	8
	District of Columbia	9
	Florida	10
	Georgia	11
	Hawaii	12
	Idaho	13
	Illinois	14
	Indiana	15
	Iowa	16
	Kansas	17
	Kentucky	18
	Louisiana	19
	Maine	20
	Maryland	21
	Massachusetts	22

Enumeration Name	Enumerator Description	Value
	Michigan	23
	Minnesota	24
	Mississippi	25
	Missouri	26
	Montana	27
	Nebraska	28
	Nevada	29
	New Hampshire	30
	New Jersey	31
	New Mexico	32
	New York	33
	North Carolina	34
	North Dakota	35
	Ohio	36
	Oklahoma	37
	Oregon	38
	Pennsylvania	39
	Rhode Island	40
	South Carolina	41
	South Dakota	42
	Tennessee	43
	Texas	44
	Utah	45
	Vermont	46

Enumeration Name	Enumerator Description	Value
	Virginia	47
	Washington	48
	West Virginia	49
	Wisconsin	50
	Wyoming	51

Enumeration Name	Enumerator Description	Values
SynchronisationType		
	Synchronous	0
	Asynchronous	1
	Not Defined	2
TakeoffMinimumType		
	Takeoff Not Standard	0
	Not Defined	1
TimeCode		
	Active Continuously Including Holidays	0
	Active Continuously Excluding Holidays	1
	Active for Time of Operation Excluding Holidays	2
	Active for Time of Operation Including Holidays	3
	Not Defined	4
TimeIndicator		
	Time Codes are Local Time	0
	Time Codes adjusted for Daylight Savings Time	1
	Times shown in Universal Coordinated Time	2
	Not Defined	3
TrackDescription		
	Automatically at the fix after one full circuit	0
	Automatically at a fix after reaching an altitude	1
	Manually	2
	Not Defined	3
TurnDirection		

Enumeration Name	Enumerator Description	Values
	Left	0
	Right	1
	Left or Right	2
	Not Defined	3
WaypointDescription		
	Airport as Waypoint	0
	Essential Waypoint	1
	Off Airway Waypoint	2
	Runway/Helipad as Waypoint	3
	Heliport as Waypoint	4
	NDB Navaid as Waypoint	5
	Phantom Waypoint	6
	Non-Essential Waypoint	7
	Transition Essential Waypoint	8
	VHF Navaid as Waypoint	9
	Airport or Heliport as Waypoint	10
	VOR, VORDME, VORTAC as Waypoint	11
	Tacan as Waypoint	12
	NDB, NDBDME as Waypoint	13
	ILS as Waypoint	14
	Not Defined	15
WaypointType		
	Arc Center Fix Waypoint	0
	Combined Named Intersection and RNAV Waypoint	1

Enumeration Name	Enumerator Description	Values
	Unnamed Charted Intersection	2
	Middle Marker as Waypoint	3
	NDB Navaid as Waypoint	4
	Terminal NDB Navaid as Waypoint	5
	Outer Marker as Waypoint	6
	Named Intersection	7
	Uncharted Airway Intersection	8
	VFR Waypoint	9
	RNAV Waypoint	10
	Unnamed Charted Off Route Fix	11
	Named NDB	12
	Off Route NDB	13
	Named Off Route Fix	14
	Not Defined	15
WaypointUsageType		
	High and Low Altitude	0
	High Altitude	1
	Low Altitude	2
	Terminal Use Only	3
	RNAV	4
	Runway or Displaced Threshold	5
	Pitch and Catch (RNAV)	6
	Off Route Intersection in FAA Airspace	7
	ATCAA and SUAS Waypoints in FAA High Alt	8

Enumeration Name	Enumerator Description	Values
	Not Defined	9
WeatherBroadcast		
	Automatic Transcribed Weather Broadcast	0
	Scheduled Weather Broadcast	1
	Not Defined	2
WeatherCondition		
	Visual Flight Rules	0
	Instrument Flight Rules	1
	Visual Meteorological Rules	2
	Instrument Meteorological Rules	3
	Notice to Airmen	4
	Visual and Instrument Flight Rules	5
	Visual and Instrument Meteorological Rules	6
	Not Defined	7
GroupPermission		
	User	0
	Editor	1
	Admin	2
LoggerMessage		
	Number of duplicate %s	6
	IDENT %s has %s duplicates	7
OutputTypeEnum		
	Container output	0
	Dual container output	1

Enumeration Name	Enumerator Description	Values
	Status output	2
	Container and status output	3
	Parser status output	4
	String array output	5
	Value output	6

Appendix J

J. XML Schema Definitions

The CDB specification makes an extensive use of XML to describe several parts of the specification. XML is used to describe CDB metadata, to store global datasets, to add attributes to OpenFlight models, to describe base and composite materials, etc.

This appendix lists all the schemas used by the Specification. A schema is required to formally define the format of an XML file and to validate its content.

> Note: As of version 3.2 of the Specification, XML schemas are no longer provided in text form in appendix J. Instead, the actual XSD files are delivered with the CDB Specification Distribution Package.

The following XML Schemas can be found in the `\CDB\Metadata\Schema` subdirectory of the CDB Specification Distribution Package:

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`

J.1 The CDB Namespace

The CDB Specification makes use of several XML namespaces to isolate the definitions of its schemas. The name of these namespaces is built around a base URL whose name is "`http://www.CDB-Spec.org/`"[15].

J.2 Schema Conventions

The target namespace of all CDB schemas follow this pattern:

[15] As of December 2013, this URL does not correspond to an active Web site. An XML namespace does not need to be the address of a Web site, although it is practical to do so because it ensures its uniqueness.

"http://www.CDB-Spec.org/Schema/[*Name*]/[*Version*]"

Where the *Name* is identical to the filename portion of the file containing the schema and *Version* is the version number of the schema.

To illustrate how a target namespace is composed, here is the target namespace of the schema found in Version.xsd (item 12 in the list above):

"http://www.CDB-Spec.org/Schema/Version/3.2"

Appendix K

K. CDB Coordinate Systems

K.1 Spatial Reference Frames (SRF) and Coordinate Systems

The handling of spatial data requires a good deal of rigor to accurately describe the position of points in space. Furthermore, it requires the ability to define directions and distances. Generally, this is accomplished through the use of coordinate systems.

It is often convenient to represent position in several different spatial reference frames. Each spatial reference frame provides a particular way of defining positions within its domain. This level of abstraction also permits spatial reference frames to be decomposed into a series of, or even a hierarchy of reference, each relative to another reference frames. This mechanism permits objects to be independently defined (positioned, oriented, and scaled) with respect to a local spatial reference frame and then be later incorporated into other reference frames. The reference frames can be abstract mathematical constructs or they can be bound to real world objects (e.g., a flap located defined in a flap reference frame which in turn is bound to an aircraft wing reference frame).

The CDB specification defines the conceptual model and the methodologies that allow the description, and transformation or conversion, of geometric properties within a set of spatial reference frames supported by the Specification. The CDB Spatial Reference Model (SRM) supports an unambiguous specification of the positions, directions, and distances associated with spatial information. It also defines algorithms for precise transformation of positions, directions and distances among different spatial reference frames.

K.2 CDB Approach

One of the primary objectives of the CDB specification is to provide the means to represent the entire earth. Secondly, it must handle spatial data with a good deal of rigor to accurately describe the position of points in space, and must do so at the level of fidelity commensurate with the precision that is now possible in modern simulators.

The size, content, fidelity and precision of synthetic environments now warrant a different approach, an approach that entirely avoids the “problem with projections” and other approximations used in the past. To this end, the CDB specification mandates the use of geographic coordinates to represent the shape of terrain surfaces and the position of all cultural features. Since no projections are involved, full geometric coherence is assured without compromise and all four key spatial properties can be achieved simultaneously:

(1) Preservation of distance
(2) Preservation of direction
(3) Preservation of area
(4) Preservation of shape

K.3 CDB SRF Specifications, and Algorithmic Development

The CDB specification is based on a surface geodetic coordinate system, i.e., points on the earth surface are specified in geographic lat/long/elevation coordinates.

The CDB embeds modeled point features (e.g., the representation of 3-D objects, moving and/or static) within a General Cartesian Coordinate System. Its use is generally constrained to objects that are small in comparison to the earth. As shown below, a modeled point feature can be referenced anywhere on the earth by providing the model's orientation (the AO1 attribute specified in Chapter 5) and the model's origin using a set of geographic lat/long/elevation coordinates. Note that the model's z-axis implicitly points upward with respect to the earth surface.

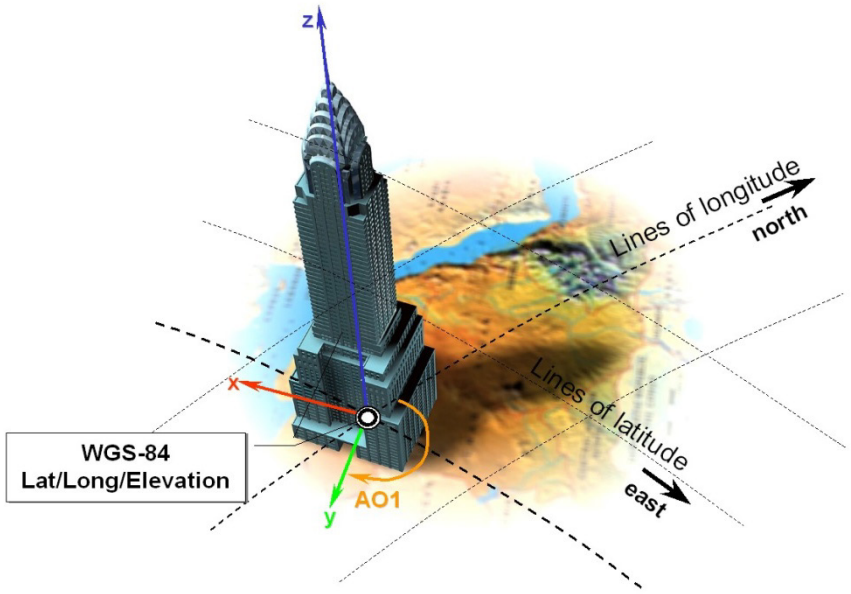


Figure K-1: Cartesian Model positioned to WGS-84 Coordinates

The earth shape is described by the WGS-84 reference ellipsoid. The Specification also defines three related set of Spatial Frames (and associated coordinate systems) for use in conjunction with the surface geodetic coordinate system; they are:

(1) Earth-centered Cartesian (Geocentric)
(2) Generic Cartesian
(3) Local Vertical (LVCS)

K.3.1 Geographical Coordinate System (Geodetic)

The geographical coordinate system (also called the geodetic coordinate system), where the coordinates are longitude, latitude and altitude above mean sea level, is the most commonly used coordinate system today. Geographical latitude □ and longitude □ are the angles of the normal on the reference ellipsoid along the point to the equator and zero meridian. The angles are normally given as degrees, minutes and seconds. Altitude above mean sea level is the distance above and normal to the geoid in meters. The WGS 84 ellipsoid represents the actual geoid within an accuracy of 100 meters. The prime meridian and the equator are the reference planes used to define latitude and longitude.

In other terms, the geographic latitude – there are many other defined latitudes – of a point is the angle between the equatorial plane and a line normal to the reference ellipsoid's surface. The geographic longitude of a point is the angle between a reference plane, Greenwich, and a plane passing through the point, both planes being perpendicular to the equatorial plane. The geographic height at a point is the distance from the reference ellipsoid to the point in a direction normal to the reference ellipsoid.

Table K-1: Geographic Coordinate System (Geodetic)

Field	Specification
Properties	Orthogonal.
CS parameters and constraints	*a*: major semi-axis length *b*: minor semi-axis length Constraints: $a > b$: (oblate ellipsoid)
Coordinate components	□: longitude in radians, and □: geodetic latitude in radians.
Domain of the generating function or mapping equations	$-\pi/2 < \phi < \pi/2$ $-\pi < \lambda \leq \pi$
Domain of the inverse of the generating function or mapping equations	$\frac{x^2}{a^2} + \frac{y^2}{a^2} + \frac{z^2}{b^2} = 1$ and $z \neq \pm b$.

Field	Specification
Notes	(1) The CS surface is the oblate ellipsoid (or sphere) surface excluding the pole points. (2) The geodetic 3D CS induces this CS on the 3rd coordinate surface at any point for which $h = 0$. (3) If a = b, the geodetic latitude □ coincides with the spherical latitude □. (4) For WGS-84 a= 6,378,137 m and b = 6,356,752 m. The inverse flattening ratio f^{-1} is 298.257223563.

K.3.2 Earth-Centered SRF (aka Rectangular Geocentric SRF)

The earth-centered spatial reference frame defines a three-dimensional Euclidian space with respect to the geometric center of the reference ellipsoid, the center of the earth. The reference datum of the Earth-centered SRF is based on the WGS-84 ellipsoid reference model. In this SRF, the z-axis is pointing at the North Pole, the x-axis is pointing at the intersection of the equator and the Greenwich meridian, the prime meridian, and the y-axis is pointing at the intersection of the equator and 90° east longitude. The associated coordinate system is called the World Coordinate System (WCS); its units are meters. The WCS is used to specify the 3D position of objects with respect to the earth-centered SRF. This coordinate system is used as an intermediate system to convert geodetic coordinates to LVCS and vice versa.

K.3.3 General Cartesian SRF

The Cartesian spatial reference frame defines a three-dimensional Euclidian space with respect to an arbitrary origin. The reference datum specifying the origin and the orientation of the SRF is arbitrary, i.e. the reference datum can be specified within a geocentric SRF, a LVCS SRF or any other SRF. The SRF is right-handed and orthonormal. In this SRF system, the z-axis is pointing up and both the x-axis and y-axis lie in the horizontal plane. The associated coordinate system is called the General Cartesian Coordinate System; coordinates are specified in meters. This coordinate system is used for the representation of 3-D objects, moving and/or static. Its use is generally constrained to objects that are small in comparison to the earth[16].

K.3.4 Local Vertical SRF

The Local Vertical SRF (LVCS) spatial reference frame defines a three-dimensional Euclidian space. It is a SRF similar to the Geocentric SRF except that the origin of the SRF is translated and rotated to a point on the surface of the WGS-84 ellipsoid. At that point, the x-y plane is tangent to the surface of the earth and the z-axis is normal to the ellipsoid. The associated coordinate system is called the local vertical coordinate system; the coordinates are specified in meters. In this coordinate system, the z-axis is pointing up, the y-axis is pointing north and the

16 To ensure that the object preserves its shape, size, orientation, and relative geometry.

x-axis is pointing east. Its use is generally constrained to a surface that is small in comparison to the earth[17].

K.4 Geodetic to Geocentric Transformation

Information providers and information consumers must transform geodetic information to geocentric information according to the following:

(1) If $<\varphi, \lambda, h>$ represents the geodetic coordinates to be transformed, where φ is the latitude, λ is the longitude, and h is the WGS84 height above mean-sea-level; and

(2) If $<x, y, z>$ represents the geocentric coordinates; then using the WGS84 ellipsoid equatorial radius, a, of 6,378,137.0m and the WGS84 ellipsoid polar radius, b, of 6,356,752.314245m, the flattening f, the eccentricity e and the radius of curvature as a function of latitude $N(\varphi)$ are given by equation eq. A-1:

$$f = (a-b)/a$$
$$e^2 = 2f - f^2$$
$$N(\varphi) = a/\sqrt{1 - e^2 \sin^2 \varphi} \qquad \text{(eq. A-1)}$$

From these equations, we define the transformation of each geodetic coordinate as:

$$x = (N(\varphi) + h)\cos\varphi \cos\lambda$$
$$y = (N(\varphi) + h)\cos\varphi \sin\lambda \qquad \text{(eq. A-2)}$$
$$z = (N(\varphi)(1 - e^2) + h)\sin\varphi$$

17 To ensure that the object preserves its shape, size, orientation, and direction

K.5 Geocentric to Geodetic Transformation

Geocentric coordinates cannot be transformed to the geodetic coordinate system directly. Instead, a successive approximation approach is used to compute the new coordinates. The following describes the algorithm to convert geocentric coordinates <*x, y, z*> to geodetic coordinates <□, □, *h*>, where □ is the latitude, □ is the longitude, and *h* is the WGS84 height above mean-sea-level. First, using the WGS84 ellipsoid equatorial radius, *a* = 6,378,137.0 m and the WGS84 ellipsoid polar radius, *b* = 6,356,752.314245 m, the flattening *f* and the eccentricity *e* of the ellipsoid are given by equation A-3:

$$f = (a-b)/a$$
$$e^2 = 2f - f^2 \qquad \text{(eq. A-3)}$$

We first compute the longitude □ with equation A-4:

$$\lambda = \tan^{-1}\left(\frac{y}{x}\right) \qquad \text{(eq. A-4)}$$

We then compute a first approximation of the latitude assuming a spherical earth model with equation A-5:

$$\varphi = \tan^{-1}\left(\frac{z}{\sqrt{x^2 + y^2}}\right) \qquad \text{(eq. A-5)}$$

Then, we iteratively compute the radius of curvature as a function of latitude *N(□)* and, as a result we iteratively converge to a new, more accurate latitude □' with equation A-6:

$$N(\varphi) = a/\sqrt{1 - e^2 \sin^2 \varphi} \qquad \text{(eq. A-6)}$$

$$\varphi' = \tan^{-1}\left(\frac{z + N(\varphi)\ e^2 \sin\varphi}{\sqrt{x^2 + y^2}}\right)$$

For each iteration, φ is replaced with φ', until the difference between the two values is less than a preset allowable error. The resulting latitude error will be less than φ. Finally, we compute the height above mean-sea-level *h* with equation A-7

$$h = \frac{\sqrt{x^2 + y^2}}{\cos\varphi} - N(\varphi) \qquad \text{(eq. A-7)}$$

K.6 Geodetic to LVCS Coordinate Transformation

The transformation of a geodetic coordinate into an LVCS coordinate is decomposed into two parts:

(1) Apply a coordinate transformation to each coordinate of an object from the geodetic coordinate system to the rectangular geocentric coordinate system.

(2) Then apply a second transformation to go from the geocentric coordinate system to LVCS.

The first transformation, from geodetic to rectangular geocentric is described in section K.4. The transformation is applied to the origin of the object. The result of this transformation is the origin x_0 of the object in the geocentric coordinate system. Then for each coordinate x of the object, we apply the geodetic to geocentric transformation to coordinate x and we then compute the translation vector t between x and x_0 in the geocentric coordinate system with equation A-8

$$t = x - x_0 \qquad \text{(eq. A-8)}$$

The second transformation, from geocentric to LVCS is presented here as an algorithm to transform all coordinates of an object from the geodetic coordinate system to LVCS. The transformation from geodetic to LVCS first requires the assembly of a 3x3 rotation matrix M with equation A-9:

$$M = \begin{bmatrix} -\sin\lambda_0 & \cos\lambda_0 & 0 \\ -\sin\varphi_0\cos\lambda_0 & -\sin\varphi_0\sin\lambda_0 & \cos\varphi_0 \\ \cos\varphi_0\cos\lambda_0 & \cos\varphi_0\sin\lambda_0 & \sin\varphi_0 \end{bmatrix} \quad \text{(eq. A-9)}$$

Where: □*0* and □*0* = the latitude and longitude of the origin of the object.

Finally, the rotation matrix M is applied to the translation vector t to obtain each coordinate $\mathbf{x_L}$ in the local vertical coordinate system with equation A-10:

$$x_L = Mt \quad \text{(eq. A-10)}$$

K.7 Angular Displacements to Linear Displacement

For WGS84, which is an elliptical representation of the earth, the transformation from angular displacements to equivalent linear displacements in a tangential plane is slightly different than that for a spherical earth.

For WGS84 we get…

$$\delta X = \rho_t \cos(lat)\delta lon$$
$$\delta Y = \rho_m \delta lat$$

… as opposed to for a spherical earth

$$\delta X = \rho \cos(lat)\delta lon$$
$$\delta Y = \rho \delta lat$$

where…

$\delta X, \delta Y$	are the linear displacements along the x and y axes.
ρ_m, ρ_t	are the meridional and transverse radiuses of curvature.
ρ	is the radius of the spherical earth.
$\delta lat, \delta lon$	are small displacements at location *lat/lon*

we have…

$$\rho_m = \frac{a(1-e^2)}{\left[1-e^2 \sin^2(lat)\right]^{3/2}}$$
$$\rho_t = \frac{a}{\sqrt{1-e^2 \sin^2(lat)}}$$
$$e^2 = 1 - \frac{b^2}{a^2}$$
$$f = \frac{a-b}{a} \Rightarrow b = a(1-f)$$

where…

e^2	is the square of the eccentricity
a,b	are the semi-major an the minor axes of the earth
f	is the flattening

K.8 3D Model Coordinate System

This section describes the transformations required to go to-and-from the DIS/HLA and the CDB moving model coordinate systems.

The CDB 3D model coordinate system conventions were presented earlier in Chapter 6.

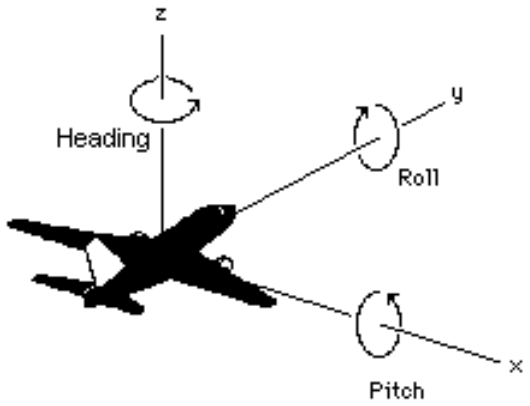


Figure K-2: CDB 3D Model Coordinate System

The DIS coordinate system is used on a HLA network and is represented on the following figure.

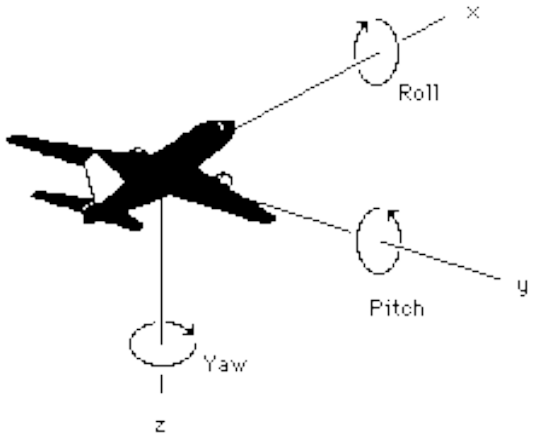


Figure K-3: DIS Entity[18] Coordinate System

The two coordinate systems differ in the axis conventions (Z is up in the CDB while Z is down in DIS). Furthermore, the position of the origin also differs; DIS requires that the origin of its

[18] DIS refers to a 3D model as an entity.

coordinate system be located at the center of the entity's bounding box excluding its articulated and attached parts[19]. The CDB specification uses a different convention.

The transformation from the CDB coordinate system to the DIS coordinate system involves one translation followed by two rotations. The translation represents the offset to the DIS origin as defined in chapter 6. Assume that P_0 represents the coordinate of the DIS origin.

$$P_0 = (x_0, y_0, z_0) \qquad \text{(eq. A-11)}$$

The two rotations are relatively simple. First, rotate 180° about the X-axis. This rotation will position the Z-axis in its correct position. Equation A-12 represents this rotation.

$$M_x = \begin{Bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{Bmatrix} \qquad \text{(eq. A-12)}$$

Second, rotate -90° about this new Z-axis. This last rotation completes the transformation and is represented by equation A-13.

$$M_z = \begin{Bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{Bmatrix} \qquad \text{(eq. A-13)}$$

Now, if we combine equations A-11, A-12 and A-13, we can transform a point P expressed in the CDB coordinate system into point P' in the DIS coordinate system. Equation A-14 presents the complete transformation.

[19] This definition can be found on page 3 of IEEE Std 1278.1-1995. Note that the CDB provides the means to store the DIS origin within the coordinate system space of the model.

$$P' = M_z M_x (P - P_0) \qquad \text{(eq. A-14)}$$

The combined matrix gives equation A-15 and the resulting individual terms are presented in A-16.

$$M_{zx} = \begin{Bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{Bmatrix} \qquad \text{(eq. A-15)}$$

$$\begin{aligned} x' &= y - y_0 \\ y' &= x - x_0 \\ z' &= z_0 - z \end{aligned} \qquad \text{(eq. A-16)}$$

If a single transformation matrix M is preferred then Matrix M_{zx} and point P_0 are combined to obtain the set of equations A-17.

$$P' = MP$$

where...

$$M = \begin{Bmatrix} 0 & 1 & 0 & -y_0 \\ 1 & 0 & 0 & -x_0 \\ 0 & 0 & -1 & z_0 \\ 0 & 0 & 0 & 1 \end{Bmatrix} \qquad \text{(eq. A-17)}$$

and...

$$P = \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$

To convert from the DIS coordinate system back to the CDB coordinate system, the inverse transformation is applied. Knowing that unscaled rotation matrices (the upper 3 x 3 portion of M) have the property that their inverse is their transpose, we obtain the set of equations A-18.

$$P = M^{-1}P'$$

$$where...$$

$$M^{-1} = \begin{Bmatrix} 0 & 1 & 0 & x_0 \\ 1 & 0 & 0 & y_0 \\ 0 & 0 & -1 & z_0 \\ 0 & 0 & 0 & 1 \end{Bmatrix} \qquad \text{(eq. A-18)}$$

Appendix L

L. CDB Base Materials

The complete list of CDB Base Materials is part of the metadata provided with the CDB Specification Distribution Package and can be found in the following file:

```
\CDB\Metadata\Materials.xml
```

Note: As of CDB Specification 3.2, the list of CDB Base Materials is no longer presented here to avoid the risk of miscorrelation between the appendix and the metadata. The list is now exclusively found in the Metadata folder.

Appendix M

M. CDB Directory Naming and Structure

In previous versions of the Specification, Appendix M was used to present the complete list of names allowed to construct the directories of the CDB. As of version 3.2, the appendix has been replaced by a combination of folder hierarchy and metadata files delivered with the CDB Specification Distribution Package.

The `/CDB` folder hierarchy provides a complete list of directory and file name patterns of the CDB; it summarizes the structure of the CDB presented in chapter 3. The following files are necessary to expand the patterns:

- `/CDB/Metadata/Feature_Data_Dictionary.xml` provides the list of directory names associated with FACC codes.
- `/CDB/Metadata/Moving_Model_Codes.xml` provides the list of names for DIS Entity Kinds, Domains, and Categories.
- `/CDB/Metadata/DIS_Country_Codes.xml` contains the list of DIS Country Names.

Together, these files provide all the information required to build the names of all directories permitted by the Specification.

Appendix N

N. CDB Feature Data Dictionary

The CDB Feature Data Dictionary (FDD) is provided with the CDB Specification in the form of an XML file. An XML Stylesheet is provided to format and display the dictionary inside a standard Web browser. Furthurmore, the XML Schema defining the format of the FDD can also be found in the Schema subdirectory of the CDB Specification Distribution Package.

See `/CDB/Metadata/Feature_Data_Dictionary.xml` for the complete list of FACC codes supported by the Specification.

Note: As of CDB Specification 3.2, the CDB FDD is no longer presented here to avoid the risk of miscorrelation between the appendix and the metadata. The FDD is now exclusively found in the Metadata folder.

Appendix O

O. List of Texture Component Selectors

The following table provides the list of codes to use to build CDB model texture filenames.

Texture Kind CS1 (Sxxx)	Texture Index CS2 (Txxx)	Description
002 – Month	001	January
	002	February
	003	March
	004	April
	005	May
	006	June
	007	July
	008	August
	009	September
	010	October
	011	November
	012	December
003 – Season	001	Spring
	002	Summer
	003	Autumn
	004	Winter
004 – Uniform Paint Scheme	001	Grey
	002	White
	003	Green
	004	Black
	005	Beige

Texture Kind CS1 (Sxxx)	Texture Index CS2 (Txxx)	Description
	006	Blue
	007	Red
	008	Yellow
	009	Brown
	010	Pink
	011	Purple
	012	Burgundy
	013	Orange
	014	Light Blue
	015	Khaki
	016	Dark Grey
	017	Amber
	018	Gold
	019	Silver
	020	Copper
005 – Camouflage Paint Scheme	001	Desert
	002	Winter
	003	Forest
	004	Generic
	005	Urban
006 – Airline Paint Scheme	001	AAH Aloha Airlines Inc.
	002	AAL American Airlines Inc.
	003	AAR Asiana Airlines Inc.
	004	AAW Afriqiyah Airways
	005	ABR Air Contractors (UK) Limited

Texture Kind CS1 (Sxxx)	Texture Index CS2 (Txxx)	Description
	006	ACA Air Canada
	007	ACI Air Caledonie International
	008	ADR Adria Airways - The Airline of Slovenia
	009	AEA Air Europa Lineas Aereas, S.A.
	010	AEE Aegean Airlines S.A.
	011	AEW Aerosvit Airlines
	012	AFG Ariana Afghan Airlines
	013	AFL Aeroflot Russian Airlines
	014	AFR Air France
	015	AGN Air Gabon
	016	AHY Azerbaijan Hava Yollary
	017	AIC Air-India Limited
	018	AIZ Arkia - Israeli Airlines Ltd
	019	AJM Air Jamaica
	020	ALK SriLankan Airlines Limited
	021	AMC Air Malta p.l.c.
	022	AML Air Malawi Limited
	023	AMU Air Macau Company Limited
	024	AMX Aeromexico
	025	ANA All Nippon Airways Co. Ltd.
	026	ANG Air Niugini Pty Limited
	027	ANS Air Nostrum L.A.M.S.A.
	028	ANZ Air New Zealand Limited
	029	ARG Aerolineas Argentinas
	030	ASA Alaska Airlines Inc.

Texture Kind CS1 (Sxxx)	Texture Index CS2 (Txxx)	Description
	031	ATC Air Tanzania Company Ltd.
	032	AUA Austrian Airlines, Osterreichische
	033	AUI Ukraine International Airlines
	034	AUT Cielos del Sur S.A.
	035	AVA Aerovias del Continente Americano – Avianca
	036	AVN Air Vanuatu (Operations) Limited
	037	AWE America West Airlines Inc.
	038	AZA Alitalia - Linee Aeree Italiane
	039	AZW Air Zimbabwe (Pvt) Ltd.
	040	BAG dba Luftfahrtgesellschaft mbH
	041	BAW British Airways p.l.c.
	042	BBC Biman Bangladesh Airlines
	043	BCS European Air Transport
	044	BCY Cityjet
	045	BEE Jersey European Airways Limited
	046	BER Air Berlin GmbH & Co. Luftverkehrs KG
	047	BKP Bangkok Airways Co. Ltd.
	048	BLF Blue1 Oy
	049	BLV Bellview Airlines Ltd.
	050	BMA British Midland Airways Ltd.
	051	BOT Air Botswana Corporation
	052	BPA Blue Panorama Airlines S.p.A.
	053	BRA SAS Braathens AS
	054	BRU Belavia
	055	BRZ Samara Airlines

Texture Kind CS1 (Sxxx)	Texture Index CS2 (Txxx)	Description
	056	BWA BWIA West Indies Airways Limited
	057	CAL China Airlines
	058	CAW Comair Ltd.
	059	CCA Air China Limited
	060	CDG Shandong Airlines
	061	CES China Eastern Airlines
	062	CHH Hainan Airlines Company Limited
	063	CLH Lufthansa CityLine GmbH
	064	CLX Cargolux Airlines International S.A.
	065	CMI Continental Micronesia, Inc.
	066	CMP Compania Panamena de Aviacion, S.A.
	067	CNW China Northwest Airlines
	068	COA Continental Airlines, Inc.
	069	CPA Cathay Pacific Airways Ltd.
	070	CPN Caspian Airlines Service Company Ltd.
	071	CRL CORSAIR
	072	CSA Czech Airlines a.s., CSA
	073	CSN China Southern Airlines
	074	CTN Croatia Airlines
	075	CUB Cubana de Aviacion S.A.
	076	CXA Xiamen Airlines
	077	CYH China Yunnan Airlines
	078	CYP Cyprus Airways Limited
	079	DAH Air Algerie
	080	DAL Delta Air Lines Inc.

Texture Kind CS1 (Sxxx)	Texture Index CS2 (Txxx)	Description
	081	DAN Maersk Air A.S.
	082	DAT Delta Air Transport N.V.
	083	DHK DHL Air Limited
	084	DHX DHL International E.C.
	085	DLH Deutsche Lufthansa AG
	086	DNM Denim Air
	087	DTA TAAG - Linhas Aereas de Angola
	088	EIN Aer Lingus Limited
	089	ELG ALPI Eagles S.p.A.
	090	ELL Estonian Air
	091	ELY El Al Israel Airlines Ltd.
	092	ETD Etihad Airways
	093	ETH Ethiopian Airlines Enterprise
	094	EVA EVA Airways Corporation
	095	EWG Eurowings AG
	096	FCN Falcon Air AB
	097	FDX FedEx
	098	FIN Finnair Oyj
	099	FJI Air Pacific Ltd.
	100	GBL GB Airways Ltd.
	101	GEC Lufthansa Cargo AG
	102	GFA Gulf Air Company G.S.C.
	103	GHA Ghana Airways Corp.
	104	GIA Garuda Indonesia
	105	HCY Helios Airways

Texture Kind CS1 (Sxxx)	Texture Index CS2 (Txxx)	Description
	106	HDA Hong Kong Dragon Airlines Limited
	107	HEJ Hellas Jet S.A.
	108	HHN Hahn Air Lines
	109	HLF Hapag Lloyd Fluggesellschaft
	110	HZL Hazelton Airlines dba Regional Express
	111	IAC Indian Airlines
	112	IAW Iraqi Airways
	113	IBB Binter Canarias
	114	IBE Iberia - Lineas Aereas de Espana
	115	ICE Icelandair
	116	ICL C.A.L. Cargo Airlines Ltd.
	117	IRA Iran Air
	118	IRC Iran Aseman Airlines
	119	IRM Mahan Airlines
	120	ISR Israir Airlines and Tourism Ltd.
	121	ISS Meridiana S.p.A.
	122	IYE Yemenia - Yemen Airways
	123	JAI Jet Airways (India) Limited
	124	JAL Japan Airlines International Co., Ltd.
	125	JAT Jat Airways
	126	JAZ JALways Co. Ltd.
	127	JKK Spanair S.A.
	128	KAC Kuwait Airways
	129	KAL Korean Air Lines Co. Ltd.
	130	KHA Kitty Hawk Aircargo, Inc.

Texture Kind CS1 (Sxxx)	Texture Index CS2 (Txxx)	Description
	131	KLM KLM Royal Dutch Airlines
	132	KOR Air Koryo
	133	KQA Kenya Airways
	134	KRP Carpatair S.A.
	135	LAA Libyan Arab Airlines
	136	LAM LAM - Linhas Aereas de Mocambique
	137	LAN Lan Airlines S.A.
	138	LAP TAM - Transportes Aereos del
	139	LBC Albanian Airlines MAK S.H.P.K.
	140	LBH Laker Airways (Bahamas) Limited
	141	LCO Lan Chile Cargo S.A.
	142	LDA Lauda Air Luftfahrt AG
	143	LDI Lauda Air S.p.A.
	144	LGL Luxair
	145	LIL Lithuanian Airlines
	146	LLB Lloyd Aereo Boliviano S.A. (LAB)
	147	LOT LOT - Polish Airlines
	148	LPE Lan Peru S.A.
	149	LRC Lineas Aereas Costarricenses S.A.
	150	LTU LTU International Airways
	151	LXR Air Luxor, S.A.
	152	MAH Malev Hungarian Airlines Limited
	153	MAK Macedonian Airlines
	154	MAS Malaysia Airline System Berhad
	155	MAU Air Mauritius

Texture Kind CS1 (Sxxx)	Texture Index CS2 (Txxx)	Description
	156	MAZ Zambian Airways
	157	MDG Air Madagascar
	158	MEA Middle East Airlines AirLiban
	159	MGL MIAT - Mongolian Airlines
	160	MGX Montenegro Airlines
	161	MLD Air Moldova
	162	MPX Aeromexpress S.A. de C.V.
	163	MRS Air Marshall Islands, Inc.
	164	MSR Egyptair
	165	MXA Compania Mexicana de Aviacion
	166	NBK Albarka Air Services Ltd.
	167	NCA Nippon Cargo Airlines
	168	NMB Air Namibia
	169	NTW Nationwide Airlines (Pty) Ltd.
	170	NWA Northwest Airlines, Inc.
	171	OAL Olympic Airlines
	172	OAS Oman Aviation Services Co. (SAOG)
	173	PAL Philippine Airlines, Inc.
	174	PAO Polynesian Limited
	175	PGA Portugalia - Companhia Portuguesa de
	176	PIA Pakistan International Airlines
	177	PLK Pulkovo Aviation Enterprise
	178	PNW Palestinian Airlines
	179	PUA Pluna Lineas Aereas Uruguayas S.A.
	180	QFA Qantas Airways Ltd.

Texture Kind CS1 (Sxxx)	Texture Index CS2 (Txxx)	Description
	181	QTR Qatar Airways(Q.C.S.C)
	182	RAM Royal Air Maroc
	183	RBA Royal Brunei Airlines Sdn. Bhd.
	184	REU Air Austral
	185	RJA Royal Jordanian
	186	ROT TAROM - Transporturile Aeriene Romane
	187	RSN Royal Swazi National Airways Corp.
	188	RWD Rwandair Express
	189	SAA South African Airways
	190	SAS Scandinavian Airlines System (SAS)
	191	SAT SATA - Air Acores
	192	SBI Siberia Airlines
	193	SER Aero California
	194	SEY Air Seychelles Limited
	195	SFR Safair (Proprietary) Ltd.
	196	SIA Singapore Airlines Limited
	197	SKX Skyways AB
	198	SLA Sierra National Airlines
	199	SLK SilkAir (S) Pte. Ltd.
	200	SLM Surinam Airways Ltd.
	201	SNG Air Senegal International
	202	SOL Solomon Airlines
	203	SQC Singapore Airlines Cargo Pte. Ltd.
	204	SUD Sudan Airways Co. Ltd.
	205	SVA Saudi Arabian Airlines

Texture Kind CS1 (Sxxx)	Texture Index CS2 (Txxx)	Description
	206	SWD Southern Winds S.A.
	207	SWR SWISS International Air Lines Ltd
	208	SYR Syrian Arab Airlines
	209	TAI Taca International Airlines, S.A.
	210	TAM TAM Linhas Aereas S.A.
	211	TAP TAP - Air Portugal
	212	TAR Tunisair
	213	TAY TNT Airways S.A.
	214	THA Thai Airways International Public
	215	THT Air Tahiti Nui
	216	THY Turkish Airlines Inc.
	217	TMA Trans-Mediterranean Airways
	218	TNA TransAsia Airways Corporation
	219	TSO Transaero Airlines
	220	TUA Turkmenistan Airlines
	221	UAE Emirates
	222	UAL United Airlines, Inc.
	223	UPS UPS
	224	USA US Airways, Inc.
	225	UYC Cameroon Airlines
	226	VAP Phuket Airlines Co., Ltd.
	227	VDA Volga-Dnepr Airline Joint Stock
	228	VIR Virgin Atlantic Airways Limited
	229	VLE Volare Airlines S.p.A.
	230	VLK Vladivostok Air JSC

Texture Kind CS1 (Sxxx)	Texture Index CS2 (Txxx)	Description
	231	VRG Varig S.A.
	232	VSP Viacao Aerea Sao Paulo, S.A. (VASP)
	233	VTA Air Tahiti
	234	WIF Wideroe's Flyveselskap A.S.
	235	WNT Cargojet Airways Ltd.
	236	CRX Crossair
	237	WJA WestJet Airlines Ltd.
	238	JAS Japan Air System
	239	NWW North West Airlines
	240	MEP Midwest Express Airlines
	241	TWA Trans World Airlines
	242	SAB Sabena
	243	TUI Tuninter
	244	SRT Trans Asian Airlines
	245	JBU JetBlue Airways
	246	TSC Air Transat
	247	SWG Sunwing Airlines
	248	FFM Firefly
	249	BVT Berjaya Air
	250	VLG Vueling Airlines
	251	SKY Skymark Airlines
	252	JST Jetstar Airways
	253	ABX ABX Air
	254	CQH Spring Airlines
	255	POE Porter Airlines

Texture Kind CS1 (Sxxx)	Texture Index CS2 (Txxx)	Description
	256	EAQ Eastern Australia
	257	EZY EasyJet
	258	NLY Niki
	259	VOZ Virgin Australia
	260	KNA Kunming Airlines
	261	CSC Sichuan Airlines
	262	VRD Virgin America
	263	DKH Juneyao Airlines
	264	KEN Kenmore Air
	265	XAK Air Kenya
	266	NZM Mount Cook Airline
	267	FDA Fuji Dream Airlines
	268	TAE TAME (Línea Aérea del Ecuador)
	269	CFE BA CityFlyer
	270	JZA Jazz Aviation
	271	CSH Shanghai Airlines
	272	BEE Flybe
	273	TYR Tyrolean Airways
	274	SWA Southwest Airlines
	275	XME Australian Air Express
	276	BEL Brussels Airlines
	277	GCR Tianjin Airlines
	278	VOI Volaris
	279	ARA Arik Air
	280	LNI Lion Air

Texture Kind CS1 (Sxxx)	Texture Index CS2 (Txxx)	Description
	281	RYR Ryanair
	282	SHU Aurora
	283	NIG Aero Contractors
	284	SCW Malmö Aviation
	285	NAX Norwegian Air Shuttle
	286	RAR Air Rarotonga
009 – Quarter	001	First quarter of the year
	002	Second quarter of the year
	003	Third quarter of the year
	004	Fourth quarter of the year
054 – Contaminant	001	Wet Surface
	002	Snowy Surface
	003	Icy Surface
	004	Slushy Surface
	005	Patchy Wet Surface
	006	Patchy Snowy Surface
	007	Patchy Icy Surface
	008	Patchy Sandy Surface
	009	Patchy Dirty Surface
	010	Volcanic Ash
	011	Patchy Volcanic Ash
055 – Skid Mark	001	Tire Mark

Examples:

- A geospecific City Hall especially decorated for the Halloween during the month (S002) of October (T010) could have a texture named Geocell_D301_S002_T010_LOD_UREF_RREF_City-Hall.rgb.
- The texture of a geotypical house used during the first (T001) quarter (S009) of the year could be named D501_S009_T001_Wxx_House.rgb.

- Similarly, the uniform (S004) grey (T001) texture used with a Cobra helicopter could be named D601_S004_T001_Wxx_Cobra.rgb.
- A 1024 by 1024 (W10) texture representing an M1A2 tank desert (T001) camouflage (S005) could be stored in a file named D601_S005_T001_W10_M1A2.rgb.
- An Airbus 380 model 800 operated by the Emirates (T221) Airlines (S006) could be stored in a file named D601_S006_T221_Wxx_A380-800.rgb.

Notes:

- Texture Kind 002 and 009 are complete; the number of months and quarters will not change.
- Texture Kind 004 will expand as new colors are added. Color names are defined here: http://en.wiktionary.org/wiki/Appendix:Colors.
- Texture Kind 005, the Camouflage Paint Scheme, follows a similar numbering scheme as the HLA's RPR-FOM Version 2 Draft 17. The list will expand as new camouflages are needed or new values added to the RPR-FOM.
- Texture Kind 006 will expand as ICAO assigns new airline acronyms.
- Texture Kind 054 and 055 will expand as new contaminants and skid marks are deemed necessary.

Appendix P

P. SGI Image File Format

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

For Version 1.00 of this spec see:

http://paulbourke.net/dataformats/sgirgb/sgiversion.html

Draft version 0.97

The SGI Image File Format

Paul Haeberli

paul@sgi.com

Silicon Graphics Computer Systems

This is the definitive document describing the SGI image file format. This is a low level spec that describes the actual byte level format of SGI image files. On SGI machines the preferred way of reading and writing SGI image files is to use the image library -limage. This library provides a set of functions that make it easy to read and write SGI images. If you are on an SGI workstation you can get info on -limage by doing:

```
% man 4 rgb
```

A note on byte order of values in the SGI image files

In the following description a notation like bits[7..0] is used to denote

a range of bits in a binary value. Bit 0 is the lowest order bit in the value.

All short values are represented by 2 bytes. The first byte stores the high order 8 bits of the value: bits[15..8]. The second byte stores the low order 8 bits of the value: bits[7..0].

So, this function will read a short value from the file:

```
unsigned short getshort(inf)
FILE *inf;
{
  unsigned char buf[2];

  fread(buf,2,1,inf);
  return (buf[0]<<8)+(buf[1]<<0);
}
```

All long values are represented by 4 bytes. The first byte stores the high order 8 bits of the value: bits[31..24]. The second byte stores bits[23..16]. The third byte stores bits[15..8]. The forth byte stores the low order 8 bits of the value: bits[7..0].

So, this function will read a long value from the file:

```
static long getlong(inf)
FILE *inf;
{
  unsigned char buf[4];

  fread(buf,4,1,inf);
  return (buf[0]<<24)+(buf[1]<<16)+(buf[2]<<8)+(buf[3]<<0);
}
```

The general structure of an SGI image file is as shown below:

The header indicates whether the image is run length encoded (RLE).

If the image is not run length encoded, this is the structure:

The Header
The Image Data

If the image is run length encoded, this is the structure:

The Header
The Offset Tables
The Image Data

The Header

The header consists of the following:

Size	Type	Name	Description
2 bytes	short	MAGIC	IRIS image file magic number
1 byte	char	STORAGE	Storage format
1 byte	char	BPC	Number of bytes per pixel channel
2 bytes	ushort	DIMENSION	Number of dimensions
2 bytes	ushort	XSIZE	X size in pixels
2 bytes	ushort	YSIZE	Y size in pixels
2 bytes	ushort	ZSIZE	Number of channels
4 bytes	long	PIXMIN	Minimum pixel value
4 bytes	long	PIXMAX	Maximum pixel value
4 bytes	char	DUMMY	Ignored
80 bytes	char	IMAGENAME	Image name
4 bytes	long	COLORMAP	Colormap ID
404 bytes	char	DUMMY	Ignored

Here is a description of each field in the image file header:

MAGIC - This is the decimal value 474 saved as a short. This identifies the file as an SGI image file.

STORAGE - specifies whether the image is stored using run length encoding (RLE) or not (VERBATIM). If RLE is used, the value of this byte will be 1. Otherwise the value of this byte will be 0. The only allowed values for this field are 0 or 1.

BPC - describes the precision that is used to store each channel of an image. This is the number of bytes per pixel component. The majority of SGI image files use 1 byte per pixel component, giving 256 levels. Some SGI image files use 2 bytes per component. The only allowed values for this field are 1 or 2.

DIMENSION - described the number of dimensions in the data stored in the image file. The only allowed values are 1, 2, or 3. If this value is 1, the image file consists of only 1 channel and only 1 scanline (row). The length of this scanline is given by the value of XSIZE below. If this value is 2, the file consists of a single channel with a number of scanlines. The width and height of the image are given by the values of XSIZE and YSIZE below. If this value is 3, the file consists of a number of channels. The width and height of the image are given by the values of XSIZE and YSIZE below. The number of channels is given by the value of ZSIZE below.

XSIZE - The width of the image in pixels

YSIZE - The height of the image in pixels

ZSIZE - The number of channels in the image. B/W (greyscale) images are stored as 2 dimensional images with a ZSIZE of 1. RGB color images are stored as 3 dimensional images with a ZSIZE of 3. An RGB image with an ALPHA channel is stored as a 3 dimensional image with a ZSIZE of 4. There are no inherent limitations in the SGI image file format that would preclude the creation of image files with more

```
than 4 channels.

PIXMIN - The minimum pixel value in the image.  The value of
0 may be used if no pixel has a value that is smaller than 0.

PIXMAX - The maximum pixel value in the image.  The value of
255 may be used if no pixel has a value that is greater than 255.
This is the value that is considered to be full brightness in
the image.

DUMMY - This 4 bytes of data should be set to 0.

IMAGENAME - An null terminated ascii string of up to 79 characters
terminated by a null may be included here.  This is not commonly
used.

COLORMAP - This controls how the pixel values in the file should be
interpreted.  It can have one of these four values:

    0:  NORMAL - The data in the channels represent B/W values
      for images with 1 channel, RGB values for images with 3
      channels, and RGBA values for images with 4 channels.
      Almost all the SGI image files are of this type.

    1:  DITHERED - The image will have only 1 channel of data.
      For each pixel, RGB data is packed into one 8 bit value.
      3 bits are used for red and green, while blue uses 2 bits.
      Red data is found in bits[2..0], green data in bits[5..3],
      and blue data in bits[7..6].  This format is obsolete.

    2:  SCREEN - The image will have only 1 channel of data.
      This format was used to store color-indexed pixels.
      To convert the pixel values into RGB values a colormap
      must be used.  The appropriate color map varies from
      image to image.  This format is obsolete.

    3:  COLORMAP - The image is used to store a color map from
```

```
            an SGI machine.  In this case the image is not displayable
            in the conventional sense.

      DUMMY - This 404 bytes of data should be set to 0. This makes the
      header exactly 512 bytes.


The Image Data (if not RLE)

    If the image is stored verbatim (without RLE), then image data directly
    follows the 512 byte header.  The data for each scanline of the first
    channel is written first.  If the image has more than 1 channel, all
    the data for the first channel is written, followed by the remaining
    channels.  If the BPC value is 1, then each scanline is written as XSIZE
    bytes.  If the BPC value is 2, then each scanline is written as XSIZE
    shorts.  These shorts are stored in the byte order described above.


The Offset Tables (if RLE)

    If the image is stored using run length encoding, offset tables
    follow the header that describe what the file offsets are to the
    RLE for each scanline.  This information only applies if the value
    for STORAGE above is 1.

            Size   | Type    | Name       | Description

     tablen longs | long    | STARTTAB   | Start table
     tablen longs | long    | LENGTHTAB  | Length table

    One entry in each table is needed for each scanline of RLE data.  The
    total number of scanlines in the image (tablen) is determined by the
    product of the YSIZE and ZSIZE.  There are two tables of longs that
    are written. Each consists of tablen longs of data.  The first
    table has the file offsets to the RLE data for each scanline in the
    image.  In a file with more than 1 channel (ZSIZE > 1) this table first
    has all the offsets for the scanlines in the first channel, followed
```

```
be offsets for the scanlines in the second channel, etc.  The second
table has the RLE data length for each scanline in the image.  In a
file with more than 1 channel (ZSIZE > 1) this table first has all the
RLE data lengths for the scanlines in the first channel, followed
be RLE data lengths for the scanlines in the second channel, etc.

To find the the file offset, and the number of bytes in the RLE data
for a particular scanline, these two arrays may be read in and indexed as
follows:

 To read in the tables:

     unsigned long *starttab, *lengthtab;

     tablen = YSIZE*ZSIZE*sizeof(long);
     starttab = (unsigned long *)mymalloc(tablen);
     lengthtab = (unsigned long *)mymalloc(tablen);
     fseek(inf,512,SEEK_SET);
     readlongtab(inf,starttab);
     readlongtab(ing,lengthtab);


 To find the file offset and RLE data length for a scanline:

     rowno is an integer in the range 0 to YSIZE-1
     channo is an integer in the range 0 to ZSIZE-1

     rleoffset = starttab[rowno+channo*YSIZE]
     rlelength = lengthtab[rowno+channo*YSIZE]

It is possible for two identical rows (scanlines) to share compressed
data.  A completely white image could be written as a single compressed
row and having all table entries point to that row.  Another little hack
that should work is if you are writing out a RGB RLE file, and a
particular scanline is achromatic (greyscale), you could just make the
r, g and b rows point to the same data!!
```

```
The Image Data (if RLE)

    This information only applies if the value for STORAGE above is 1.  If
    the image is stored using run length encoding, the image data follows
    the offset tables above.  The RLE data is not in any particular order.
    The offset tables above are used to locate the rle data for any scanline.

    The RLE data must be read in from the file and expanded into pixel
    data in the following manner:

    If BPC is 1, then there is one byte per pixel.  In this case the
    RLE data should be read into an array of chars.  To expand
    data, the low order seven bits of the first byte: bits[6..0]
    are used to form a count.  If the high order bit of the first
    byte is 1: bit[7], then the count is used to specify how many
    bytes to copy from the RLE data buffer to the destination.
    Otherwise, if the high order bit of the first byte is 0: bit[7],
    then the count is used to specify how many times to repeat the
    value of the following byte, in the destination.  This process
    continues until a count of 0 is found.  This should decompress
    exactly XSIZE pixels.

     Here is example code to decompress a scanline:

         expandrow(optr,iptr,z)
         unsigned char *optr, *iptr;
         int z;
         {
           unsigned char pixel, count;

           optr += z;
           while(1) {
               pixel = *iptr++;
               if ( !(count = (pixel & 0x7f)) )
                 return;
               if(pixel & 0x80) {
                 while(count--) {
```

```
                    *optr = *iptr++;
                    optr+=4;
                  }
                } else {
                  pixel = *iptr++;
                  while(count--) {
                    *optr = pixel;
                    optr+=4;
                  }
                }
              }
            }
```

If BPC is 2, there is one short (2 bytes) per pixel. In this case the RLE data should be read into an array of shorts. To expand data, the low order seven bits of the first short: bits[6..0] are used to form a count. If bit[7] of the first short is 1, then the count is used to specify how many shorts to copy from the RLE data buffer to the destination. Otherwise, if bit[7] of the first short is 0, then the count is used to specify how many times to repeat the value of the following short, in the destination. This process proceeds until a count of 0 is found. This should decompress exactly XSIZE pixels. Note that the byte order of short data in the input file should be used, as described above.

Implementation notes

Implementation of both RLE and VERBATIM format for images with BPC of 1 is required since the great majority of SGI images are in this format. Support for images with a 2 BPC is encouraged.

If the ZSIZE of an image is 1, it is assumed to represent B/W values. If the ZSIZE is 3, it is assumed to represent RGB data, and if ZSIZE is 4, it is assumed to contain RGB data with alpha.

The origin for all SGI images is the lower left hand corner. The

```
     first scanline (row 0) is always the bottom row of the image.

 Naming Conventions

     On SGI systems, SGI image files end with the extension .bw if
     they are B/W images, they end in .rgb if they contain RGB image
     data, and end in .rgba if they are RGB images with alpha channel.

     Sometimes the .sgi extension is used as well.

 An example

This program will write out a valid B/W SGI image file:

#include “stdio.h”

#define IXSIZE      (23)
#define IYSIZE      (15)

putbyte(outf,val)
FILE *outf;
unsigned char val;
{
    unsigned char buf[1];

    buf[0] = val;
    fwrite(buf,1,1,outf);
}

putshort(outf,val)
FILE *outf;
unsigned short val;
{
    unsigned char buf[2];

    buf[0] = (val>>8);
    buf[1] = (val>>0);
```

```
    fwrite(buf,2,1,outf);
}

static int putlong(outf,val)
FILE *outf;
unsigned long val;
{
    unsigned char buf[4];

    buf[0] = (val>>24);
    buf[1] = (val>>16);
    buf[2] = (val>>8);
    buf[3] = (val>>0);
    return fwrite(buf,4,1,outf);
}

main()
{
    FILE *of;
    char iname[80];
    unsigned char outbuf[IXSIZE];
    int i, x, y;

    of = fopen("example.rgb","w");
    if(!of) {
        fprintf(stderr,"sgiimage: can't open output file\n");
        exit(1);
    }
    putshort(of,474);       /* MAGIC                        */
    putbyte(of,0);          /* STORAGE is VERBATIM          */
    putbyte(of,1);          /* BPC is 1                     */
    putshort(of,2);         /* DIMENSION is 2               */
    putshort(of,IXSIZE);    /* XSIZE                        */
    putshort(of,IYSIZE);    /* YSIZE                        */
    putshort(of,1);         /* ZSIZE                        */
    putlong(of,0);          /* PIXMIN is 0                  */
    putlong(of,255);        /* PIXMAX is 255                */
```

```
    for(i=0; i<4; i++)      /* DUMMY 4 bytes       */
        putbyte(of,0);
    strcpy(iname,"No Name");
    fwrite(iname,80,1,of);  /* IMAGENAME           */
    putlong(of,0);          /* COLORMAP is 0       */
    for(i=0; i<404; i++)    /* DUMMY 404 bytes     */
        putbyte(of,0);

    for(y=0; y<IYSIZE; y++) {
        for(x=0; x<IXSIZE; x++)
            outbuf[x] = (255*x)/(IXSIZE-1);
        fwrite(outbuf,IXSIZE,1,of);
    }
    fclose(of);
}
```

Appendix Q

Q. Table of Dataset Codes

The table below summarizes the CDB dataset codes along with their names and their applicability to the three active versions of the CDB Specification.

Dataset		Specification	
Name	Code	3.0	3.2
Elevation	001	√	√
MinMaxElevation	002	√	√
MaxCulture	003	√	√
Imagery	004	√	√
RMTexture	005	√	√
RMDescriptor	006	√	√
Reserved	007		
Reserved	008		
Reserved	020		
GSFeature	100	√	√
GTFeature	101	√	√
GeoPolitical	102	√	√
VectorMaterial	200	√	√
RoadNetwork	201	√	√
RailRoadNetwork	202	√	√
PowerLineNetwork	203	√	√
HydrographyNetwork	204	√	√

Dataset		Specification	
Name	Code	3.0	3.2
GSModelGeometry	300	√	√
GSModelTexture	301	√	√
GSModelSignature	302	√	√
GSModelDescriptor	303	√	√
GSModelMaterial	304		√
GSModelInteriorGeometry	305		√
GSModelInteriorTexture	306		√
GSModelInteriorDescriptor	307		√
GSModelInteriorMaterial	308		√
GSModelCMT	309		√
T2DModelGeometry	310		√
NavData	400	√	√
Navigation	401	√	√
GTModelGeometry	500	√	√
	510		√
GTModelTexture	501	√	
	511		√
GTModelSignature	502	√	
	512		√
GTModelDescriptor	503	√	√
GTModelMaterial	504		√

Dataset		Specification	
Name	Code	3.0	3.2
GTModelCMT	505		√
GTModelInteriorGeometry	506		√
GTModelInteriorTexture	507		√
GTModelInteriorDescriptor	508		√
GTModelInteriorMaterial	509		√
GTModelInteriorCMT	513		√
MModelGeometry	600	√	√
MModelTexture	601	√	√
MModelSignature	602	√	
	606		√
MModelDescriptor	603	√	√
MModelMaterial	604		√
MModelCMT	605		√
Metadata	700		√
ClientSpecific	701		√
Reserved for CDB Extensions	9xx		

	Dataset Code is not used
√	Dataset Code is in use
	Dataset Code is deprecated

Dataset Code is reserved

Appendix R

R. Derived Datasets within the CDB

As seen throughout this document, the CDB Specification provides all the means and mechanisms to populate all the simulation datasets without involving data duplication by using Industry Standards. However, there are situations where a specific dataset information type needs to be derived from another existing one in order to specialize further the information into another dataset type or form.

This consideration becomes a grey area where the off-line tools' capability and the run-time simulation clients' performance levels enforces this data derivation.

It is such a case with the Mip-Map data, Min-Max Elevation datat, Tile Presence data, RCS data, and Raster Material data for example.

Source Dataset	Data Manipulation Description	Resulting Dataset(s)
Elevation Dataset	In order to produce the various Level Of Details within the Elevation Dataset, it is often necessary to over-sample or sub-sample a primary set of data values. Since those values within the LOD hierarchy may come from a single data source, the LODs can be seen as derived information which can better accommodate the client needs based on their performance level.	Elevation LODs
Elevation Dataset	For clients that need to compute line of sights (LOS) between simulation entities spread across a vast terrain area, it is imperative to have a fast way of knowing the minimum and maximum elevations within a tile without loading the entire elevation data grid. The min/max elevation dataset can be used to ensure a fast pre-determination of entities occultation state with the terrain. The min/max data is stored in the form of a quad-tree pyramid and is based on the area covered at the given depth level of the quad-tree. For example, for the maximum dataset the top will contain the maximum for the whole of the geocell, the next pyramid level contains maximum data for each the quarter geocells and so on. Similarly for the minimum the top represents the minimum for the whole of the	Min-Max Elevation

Source Dataset	Data Manipulation Description	Resulting Dataset(s)
	geocell going down as for maximums. Currently the pyramid size is fixed and goes down to level 9 which covers areas that are approximately 256x256 meters square; note that the depth level can be modified to a finer or coarser level but level 9 is suggested as a reasonable compromise of performance vs. storage. A tool will pre-determine the minimum and maximum elevations within a geocell's elevations and generate the quad-trees described previously; note that the tool will use all of the elevation data that is present in the elevation dataset to determine the maximums or minimums in a given area. The tool will provide Min-Max values to client devices through the Min-Max Elevation datasets in the CDB.	
Vector Datasets (Point, Lineal and Areal Features)	The Max Culture Height data is produced for clients that need to compute line of sights (LOS) between simulation entities spread across a vast terrain area that take into account the maximum cultural feature heights. The dataset helps rapidly assess an intersection status of line-of-sight with cultural features. This dataset is derived from the Vector Datasets of the CDB for corresponding tiles. The storage is done via a quad-tree similar to that of the min/max elevation the top of the pyramid represents the height of the highest cultural feature in the dataset going down to a suggested depth level of 9.	Max Culture Height
3D Model (GT, GS, MM) Datasets	The polar diagram data (covering all aspect angles) of the RCS dataset for Geotypical, Geospecific or Moving Models cannot readily be computed at run-time due to the complex mathematical computing algorithms and resources required to determine the Electro-Magnetic Energy absorption levels by the model's materials, the corner reflections, the multi-path reflections, EM parameters (frequency, polarization) effects, and so on. Therefore, off-line COTS tools are used to analyze the 3D model geometry and its materials in order to produce the RCS dataset specifically for different frequencies and polarizations.	RCS (Radar Cross Section)

Source Dataset	Data Manipulation Description	Resulting Dataset(s)
Vector Datasets (Point, Lineal and Areal Features)	Since the material attribution is normally done in the vector data, a rasterization operation among all features is required to come up with a raster grid of attributed materials.	Raster Material

Appendix S

S. Default Read and Write values to be used by Simulator Client-Devices

As seen throughout this document, the CDB Specification provides guidelines with respect to default values in cases where no data could be read from the CDB for requested datasets. Those default parameters are captured in a Metadata file within the CDB. The Table below summarizes all the Default Parameters Names and the suggested initial values to be used by client-devices. In cases where the default parameter would be missing altogether from **\CDB\Metadata\Defaults.xml**, Client-Devices shall use the "Default Value" found in the fourth column. A "Read" default refers to the value being assumed while reading the CDB data. A "Write" default refers to the value being written to the file when content-generation tools have partial source data.

Parameter Name	Dataset	Type	Default Value	R/W
Default_Elevation-1	001_Elevation	float	0 m	R
Default_Elevation-[2-99]	001_Elevation	float	0 m	R
Default_Primary_Elevation_Control	001_Elevation	integer	INSIDE (1)	R
Default_Subordinate_Elevation_Control	001_Elevation	integer	NO_ELEVATION (0)	R
Default_Bathymetry	001_Elevation	float	0 m	R
Default_Tide	001_Elevation	float	2.5 m	R
Default_MinElevation_CaseI	002_MinMaxElevation	float	Default_Elevation-1	R
Default_MaxElevation_CaseI	002_MinMaxElevation	float	Default_Elevation-1	R
Default_MinElevation_CaseII	002_MinMaxElevation	float	-400 m	R
Default_MaxElevation_CaseII	002_MinMaxElevation	float	8846 m	R
Default_MinElevation_CaseIII	002_MinMaxElevation	float	8846 m	W
Default_MaxElevation_CaseIII	002_MinMaxElevation	float	-400 m	W
Default_MaxCulture_CaseI	003_MaxCulture	float	600 m	R
Default_MaxCulture_CaseII	003_MaxCulture	float	0 m	R
Default_VSTI_Y_Mono	004_Imagery	float	0.5	R

Parameter Name	Dataset	Type	Default Value	R/W
Default_VSTI_Y_Red	004_Imagery	float	0.5	R
Default_VSTI_Y_Green	004_Imagery	float	0.5	R
Default_VSTI_Y_Blue	004_Imagery	float	0.5	R
Default_VSTLM_Mono	004_Imagery	float	0.0	R
Default_VSTLM_Red	004_Imagery	float	0.0	R
Default_VSTLM_Green	004_Imagery	float	0.0	R
Default_VSTLM_Blue	004_Imagery	float	0.0	R
Default_Imagery_Gamma	004_Imagery	float	1.0	R
Default_RoadNetwork_LTN	201_RoadNetwork	integer	2	R
Default_RailRoadNetwork_LTN	202_RailRoadNetwork	integer	1	R
Default_GSModelTexture_Gamma	301_GSModelTexture	float	1.0	R
Default_GSModelInteriorTexture_Gamma	306_GSModelInteriorTexture	float	1.0	R
Default_GTModelTexture_Gamma	511_GTModelTexture	float	1.0	R
Default_GTModelInteriorTexture_Gamma	507_GTModelInteriorTexture	float	1.0	R
Default_MModelTexture_Gamma	601_MModelTexture	float	1.0	R
Default_Base_Material		string	BM_LAND-MOOR	R
Default_Material_Layer		integer	0	R
Default_AO1		float	0.0	R
Default_SCAL[x,y,z]		float	1.0	R
Default_TRF		integer	4	R

Appendix T

T. JEPG 2000 File Format Syntax

The following pages are an extract from the JPEG 2000 Standard Annex I which describes the JP2 File Format Syntax.

Annex I

JP2 file format syntax

(This annex forms an integral part of this Recommendation | International Standard)

I.1 File format scope

This annex of this Recommendation | International Standard defines an optional file format that applications may choose to use to contain JPEG 2000 compressed image data. While not all applications will use this format, many applications will find that this format meets their needs. However, those applications that do implement this file format shall implement it as described in this entire annex of this Recommendation | International Standard.

This annex of this Recommendation | International Standard

- specifies a binary container for both image and metadata
- specifies a mechanism to indicate image properties, such as the tonescale or colourspace of the image
- specifies a mechanism by which readers may recognize the existence of intellectual property rights information in the file
- specifies a mechanism by which metadata (including vendor specific information) can be included in files specified by this Recommendation | International Standard

I.2 File format definitions

I.2.1 Glossary

Auxiliary component: A component from the codestream that is used by the application outside the scope of colourspace conversion. For example, an opacity component or a depth component would be an auxiliary component.

Box: A building block defined by a unique box type and length. Some particular boxes may contain other boxes.

Box contents: Refers to the data wrapped within the box structure. The contents of a particular box are stored within the DBox field within the Box data structure as defined in Annex I.6

Box type: Specifies the kind of information that shall be stored with the box. The type of a particular box is stored within the TBox field within the Box data structure as defined in Annex I.6.

Colour component: A component from the codestream that functions as an input to a colour transformation system. For example, a red component or a greyscale component would be a colour component.

Container box: An box that itself contains a contiguous sequence of boxes (and only a contiguous sequence of boxes). As the JP2 file contains only a contiguous sequence of boxes, the JP2 file is itself considered a superbox. When used as part of a relationship between two boxes, the term superbox refers to the box which directly contains the other box.

JP2 file: The name of file in the file format described in this specification. Structurally, a JP2 file is a contiguous sequence of boxes.

nnn: A three-digit number preceded by a backslash indicates the value of a single byte within a character string, where the three digits specify the octal value of that byte.

I.2.2 Acronyms

ASCII: American Standard Code for Information Interchange

ICC: International Color Consortium

PCS: Profile Connection Space

UCS: Universal Character Set

URL: Uniform Resource Locator

UTF-8: UCS Transformation Format 8

UUID: Universal Unique Identifier

XML: Extensible Markup Language

I.3 File format normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation |International Standard.At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation |International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below.Members of IEC and ISO maintain registers of currently valid International Standards.The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

- Coded character set—7 bit, American Standard Code for Information Interchange, ANSI X3.4–1986.
- International Color Consortium, ICC profile format specification. ICC.1:1998–09 <http://www.color.org/ICC-1_1998-09.PDF>
- International Electrotechnical Commission. Colour management in multimedia systems: Part 2: Colour Management, Part 2–1: Default RGB colour space—sRGB. IEC 61966–2–1 199x. 9 October 1998 <http://w3.hike.te.chiba-u.ac.jp/IEC/100/ PT61966/parts/> or <http://www.sRGB.com/>.
- W3C, Extensible Markup Language (XML 1.0), Rec-xml-19980210, <http://www.w3.org/TR/REC-xml>
- IETF RFC 2279 UTF–8, A transformation format of ISO 10646. January 1998.
- ISO/IEC 11578:1996 Information technology—Open Systems Interconnection—Remote Procedure Call, <http://www.iso.ch/cate/d2229.html>

I.4 Introduction

The JPEG 2000 file format (JP2 format) provides a foundation for storing application specific data (metadata) in association with a JPEG 2000 codestream, such as that information which is required to display the image. As many applications require a similar set of information to be associated with the compressed image data, it is useful to define the format of that set of data along with the definition of the compression technology and codestream syntax.

Conceptually, the JP2 format encapsulates the JPEG 2000 codestream along with other core pieces of information about that codestream. The building-block of the JP2 format is called an box. All data is encapsulated in boxes. This Recommendation | International Standard defines several types of boxes; the definition of each specific box type defines the kinds of data that may be found within an box of that type. Some boxes will be defined to contain other boxes.

I.4.1 File identification

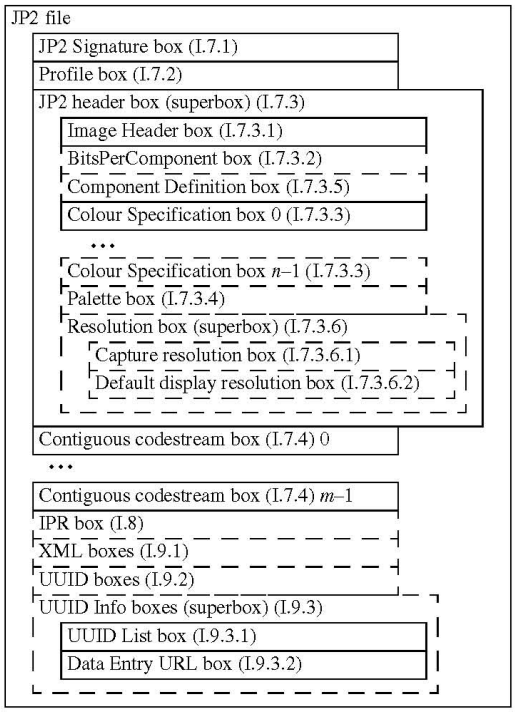
JP2 files can be identified using several mechanisms. When stored in traditional computer file systems, JP2 files should be given the file extension ".jp2" (readers shall also recognize files with the extension ".JP2"). On Macintosh file systems, JP2 files should be given the type code 'jp2\040'.

I.4.2 File organization

A JP2 file represents a collection of boxes. Some of those boxes are independent, and some of those boxes contain other boxes. The binary structure of a file is a contiguous sequence of boxes. The start of the first box shall be the first byte of the file, and the last byte of the last box shall be the last byte of the file.

The binary structure of an box is defined in Annex I.6.

Logically, the structure of a JP2 file is as shown in Figure I-1.



Boxes with dashed borders are optional in conforming JP2 files. However, an optional box may define mandatory boxes within that optional box. In that case, if the optional box exists, those mandatory boxes within the optional box shall exist. If the optional box does not exist, then the mandatory boxes within those boxes shall also not exist.

This illustration specifies only the containment relationship between the boxes in the file. A particular order of those boxes in the file is not generally implied. However, the Signature box shall be the first box in a JP2 file and the JP2 header box shall fall before the Contiguous codestream box.

Note that the file is a strict sequence of boxes. Other boxes may be found between the boxes defined in this Recommendation | International Standard. However, all such data shall be in the box format; no other data shall be found in the file.

Figure I-1 — Conceptual structure of a JP2 file

As shown in Figure I-1, a JP2 file contains a JP2 Signature box, JP2 header box, and one or more Contiguous codestream boxes. A JP2 file may also contain other box as determined by the file writer. That JP2 header box contains other boxes, such as the Image Header box and one or more Colour Specification boxes.

I.4.3 Greyscale/Colour/Palette/multi-component specification

The JP2 file format provides two methods to specify the colourspace of the image. The enumerated method specifies the colourspace of an image by specifying a numeric value that represents a well-defined colourspace definition. In this Recommendation | International Standard, images in the sRGB colourspace and greyscale images can be defined using the enumerated method.

The JP2 file format also provides for the specification of the colourspace of an image by embedding an ICC profile in the file. That profile shall be of either the Monochrome or Three-Channel Matrix-Based class of input profiles as defined by the ICC Profile Format Specification, version 2.2.0. This allows for the specification of a wide range of greyscale and RGB class colourspaces, as well as a few other spaces that can be represented by those two profiles classes. See Annex J.5 for a more detailed description of the legal colourspace transforms, how those transforms are stored in the file, and how to process an image using that transform without using an ICC colour management engine. Note though, that while restricted, these ICC profiles are fully compliant ICC profiles and the image can thus be processed through any ICC compliant engine that supports version 2.2.0 or greater profiles.

In addition to specifying the colourspace of the image, this Recommendation | International Standard provides a means by which a single component palettized image can be decoded and converted back to multiple-component form by the translation from index space to multiple-component space. Any such depalettization is applied before the colourspace of is interpreted. In the case of palettized images, the specification of the colourspace of the image is applied to the multiple-component values stored in the palette.

I.4.4 Inclusion of opacity and transparency components

The JP2 file format provides a means to indicate the presence of auxiliary components, such as opacity and transparency, to define the type of those components, and to specify the ordering of all components. When a reader opens the JP2 file, it will determine the ordering and type of each component. The application must then match the component definition and ordering from the JP2 file with the component ordering as defined by the colourspace specification. Once the file components have been mapped to the colour components, the decompressed image can be processed through any needed colourspace transformations.

In many applications, components other than the colour components are required. For example, many images used on web pages contain opacity information; the browser uses this information to blend the image into the background. It is thus desirable to include both the colour and auxiliary components with a single codestream.

I.4.5 Metadata

One important aspect of the JP2 format is the ability to add metadata to a JP2 file. Because all data is encapsulated in boxes, and all boxes have types, the format provides a simple mechanism for a reader to extract relevant information, while ignoring any box that contains information that is not understood by that particular reader. In this way, new boxes can be created, either through this or other Recommendations | International Standards or private implementation. Also, any new box added to a JP2 file shall not change the visual appearance of the image.

I.4.6 Compliance

All conforming files shall contain all boxes required by this Recommendation | International Standard, and those boxes shall be as defined in this Recommendation | International Standard. Also, all conforming readers shall correctly interpret all boxes defined in this Recommendation | International Standard and thus shall correctly interpret all conforming files.

I.5 Greyscale/Colour/Palettized/multi-component specification architecture

One of the most important aspects of a file format is that it specifies the colourspace of the contained image data. In order to properly display or interpret the image data, it is essential that the colourspace of that data is properly characterized. The JP2 format provides a multi-level mechanism for characterizing the colourspace of an image. The format also provides a mechanism to specify that an image is not photographic (such as multi-spectral data).

I.5.1 Enumerated method

The simplest method for characterizing the colourspace of an image is to specify an integer code representing the colourspace in which the image is encoded. This method handles the specification of sRGB and greyscale images. Extensions to this method can be used to specify other colourspaces, including the definition of multi-component images.

For example, the image file may indicate that a particular image is encoded in the sRGB colourspace. To properly interpret and display the image, an application must natively understand the definition of the sRGB colourspace. Because an application must natively understand each specified colourspace, the complexity of this method is dependent on the exact colourspaces specified. Also, complexity of this mechanism is proportional to the number of colourspaces that are specified and required for conformance. While this method provides a high level of interoperability for images encoded using colourspaces for which correct interpretation is required for conformance, this method is very inflexible. This Recommendation | International Standard defines a specific set of colourspaces for which interpretation is required for conformance.

I.5.2 Restricted ICC profile method

An application may also specify the colourspace of an image using a restricted set of ICC profiles. This method handles the specification of a the most commonly used RGB and greyscale class colourspaces through a low-complexity method.

An ICC profile is a standard representation of the transformation required to convert one colourspace into another colourspace. With respect to image file format, the ICC profile specification defines a type of profile that specifies how samples in a particular colourspace are converted into a standard space (the Profile Connection Space (PCS)). Depending on the original colourspace of the samples, this transformation may be either very simple or very complex.

The ICC Profile Format Specification does define a specific class of ICC profiles that are easy to implement. The ICC Profile Format Specification defines Monochrome Input and Three-Color Matrix-Based Input Profiles for which the transformation from the source colourspace to the PCS is limited to the application of a non-linearity curve and a 3x3 matrix. It is practical to expect all applications, including simple devices, to be able to process the image through the specified transformation. Thus all conforming applications are required to correctly interpret the colourspace of any image that specifies the colourspace using this subset of possible ICC profile types.

For this Recommendation | International Standard, the class of allowed profiles shall use the XYZ relative version of the PCS.

For the JP2 file format, profiles shall conform to the ICC profile definition as defined by the ICC Profile Format Specification, version 2.2.0, as well as the restrictions specified above. See Annex J.5 for a more detailed description of the legal colourspace transforms, how those transforms are stored in the file, and how to process an image using that transform without using an ICC colour management engine.

I.5.3 Using multiple methods

Architecturally, the format allows for multiple methods to be embedded in a file, providing the reader a choice as to what image processing path should be used to interpret the colourspace of the image. For JP2 files, a conforming reader shall use the first method found in the file (in the first colourspace specification box in the JP2 Header box) and ignore all other methods (found in additional colourspace specification boxes) found in the file.

I.5.4 Palettized images

In addition to specifying the interpretation of the image in terms of colourspace, this Recommendation | International Standard allows for the decoding of single component images where the value of that single component represents an index into a palette of colours. Input of a decompressed sample to the palette converts the single value to a multiple-component tuple. The value of that tuple represents the colour of that sample; that tuple shall then be interpreted according to the other colour specification methods (Enumerated or Restricted ICC) as if that multiple-component sample had been directly extracted from a multiple-component codestream.

I.5.5 Interactions with the decorrelating multiple component transform

The specification of colour within the JP2 file format is independent of the use of a multiple component transformation within the codestream (the CSSiz parameter of the SIZ marker segment as specified in Annex A.5.1 and Annex G). The colourspace transformations specified through the sequence of colour transformation boxes shall be applied to the image samples after the reverse multiple component transformation has been applied to the decompressed samples. While the

application of these decorrelating component transformations is separate, the application of an encoder-based multiple component transformation will often improve the compression of colour image data.

I.6 Box definition

Physically, each object in the file is encapsulated within a binary structure called an box. That binary structure is as follows:

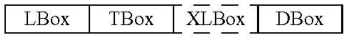


Figure I-2 — Organization of an Box

LBox: Box Length. This field specifies the length of the box, stored as a 4-byte big endian unsigned integer. This value includes all of the fields of the box, including the length and type. If the value of this field is 1, then the XLBox field shall exist and the value of that field shall be the actual length of the box. If the value of this field is 0, then the length of the box was not known when the LBox field was written. In this case, this box contains all data up to the end of the file. If an box of length 0 is contained within another box (its superbox), then the length of that superbox shall also be 0. This means that this box is the last box in the file. The values 2–7 are reserved for other use.

TBox: Box Type. This field specifies the type of data found in the DBox field. The value of this field is encoded as a 32-bit big endian unsigned integer. However, boxes are generally referred to by a ASCII character string translation of the integer value. For all box types defined within this Recommendation I International Standard, box types will be indicated as both character string (normative) and as 32-bit hexadecimal integers (informative). Also, a space character is shown in the character string translation of the box type as "\040".

XLBox: Box Extended Length. This field specifies the actual length of the box if the value of the LBox field is 1. This field is stored as an 8-byte big endian unsigned integer. The value includes all of the fields of the box, including the LBox, TBox and XLBox fields.

DBox: Box Data. This field contains the data for the portion of the object contained within this box. The format of that data is dependent on the box type and will be defined individually for each type.

Table I-1 — Binary structure of an box

Field name	Size (bits)	Value
LBox	32	0, 1, 8—(2^{32}–1)
TBox	32	Varies
XLBox	LBox=1, 64 LBox≠1, 0	16—(2^{64}–1) Not applicable
DBox	Varies	Varies

For example, consider the following illustration of a sequence of boxes, including one box that contains other boxes:

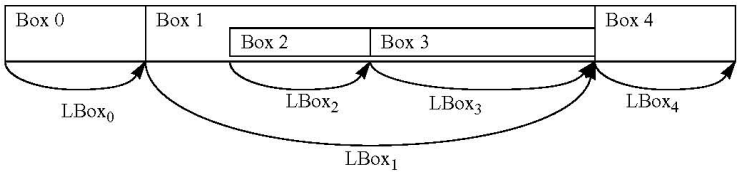


Figure I-3 — Illustration of box lengths

As shown in Figure I-3, the length of each box includes any boxes contained within that box. For example, the length of Box 1 includes the length of Boxes 2 and 3, in addition to the LBox and TBox fields for Box 1 itself. In this case, if the type of Box 1 was not understood by a reader, it would not recognize the existence of boxes 2 and 3 because they would be completely skipped by jumping the length of box 2 from the beginning of box 2.

The following table lists all boxes defined by this Recommendation | International Standard. Indentation within the table indicates the hierarchical containment structure of the boxes within a JP2 file:

Table I-2 — Boxes defined within this Recommendation | International Standard

Box name	Type	Container box	Required?	Notes
JP2 Signature box	'jP\032\032' (X'6A501A1A')	No	Required	This box uniquely identifies the file as a JP2 file.
Profile box	'prfl' (X'7072666C')	No	Required	This box specifies profile and compatibility information
JP2 Header box	'jp2h' (X'6A703268')	Yes	Required	This box contains a series of boxes that contain header-type information about the file.
Image Header box	'ihdr' (X'69686472')	No	Required	This box contains the size of the image and other related fields.
BitsPerComponent box	'bpcc' (X'62706363')	No	Optional	This box specifies the bit depth of the components in the file in cases where the bit depth is not constant across all components.
Colour Specification	'colr' (X'636F6C72')	No	Required	This box specifies the colourspace of the image.
Palette	'pclr' (X'70636C72')	No	Optional	This box specifies the palette which maps a single component in index space to a multiple-component image.
Component Definition box	'cdef' (X'63646566')	No	Optional	This box specifies the type and ordering of the components within the codestream.
Resolution box	'res ' (X'72657320')	Yes	Optional	This box specifies the resolution of the image.

Table I-2 — Boxes defined within this Recommendation | International Standard

Box name	Type	Container box	Required?	Notes
Capture resolution box	'resc' (X'72657363')	No	Optional	This box specifies the resolution at which the image was captured.
Default Display resolution box	'resd' (X'72657364')	No	Optional	This box specifies the default resolution at which the image should be displayed.
Contiguous Codestream boxes	'jp2c' (X'6A703263')	No	Required	This box contains the codestream as defined by Annex A of this Recommendation	International Standard
Intellectual Property box	'jp2i' (X'6A703269')	No	Optional	This box contains intellectual property information about the image.
XML box	'xml\040' (X'786D6C20')	No	Optional	This box provides a tool by which vendors can add XML formatted information to a JP2 file.
UUID box	'uuid' (X '75756964')	No	Optional	This box provides a tool by which vendors can add additional data to a file without risking conflict with other vendors.
UUID Info box	'uinf' (X'75696E66')	Yes	Optional	This box provides a tool by which a vendor may provide access to additional information associated with a UUID
UUID list box	'ulst' (X'75637374')	No	Optional	This box specifies a list of UUID's.
URL box	'url\040' (X'75726C20')	No	Optional	This box specifies a URL.

I.7 Defined boxes

The following boxes shall properly be interpreted by all conforming readers. Each of these boxes conforms to the standard box structure as defined in Annex I.6. The following sections define the value of the DBox field from Table I-1 (the contents of the box). It is assumed that the LBox, TBox and XLBox fields exist for each box in the file as defined in Annex I.6.

I.7.1 JP2 Signature box

The JP2 signature box identifies that the format of this file was defined by the JPEG 2000 Recommendation | International Standard, as well as provides a small amount of information which can help determine the validity of the rest of the file. The JP2 signature box shall be the first box in the file, and all files shall contain one and only one JP2 signature box.

The type of the JP2 signature box shall be 'jP\032\032' (X'6A501A1A'). The length of this box shall be 12 bytes. The contents of this box shall be the 4-byte character string '<CR><LF><X'87'><LF>' (X'0D0A870A'). For file verification purposes, this box can be considered a fixed-length 12-byte string which shall have the value:
X'0000 000C 6A50 1A1A 0D0A 870A'.

The combination of the particular type and contents for this box enable an application to detect a common set of file transmission errors. The CR-LF sequence in the contents catches bad file transfers that alter newline sequences. The control-Z character in the type stops file display under MS-DOS. The final linefeed checks for the inverse of the CR-LF translation problem. The third character of the box contents has its high-bit set to catch bad file transfers that clear bit 7.

I.7.2 Profile box

The Profile box specifies information about the Recommendations | International Standards with which the file is compatible, and allows the file creator to specify the Recommendations | International Standards representing the intended purpose of the file. This box shall immediately follow the JP2 signature box. Also, all files shall contain one and only one Profile box.

The type of the Profile Box shall be 'prfl' (X'7072666C'). The contents of this box shall be as follows:

Figure I-4 — Organization of the contents of a Profile box

BR: Brand. This field specifies the governing Recommendation | International Standard on which the file is based. This field is specified by a four byte string of ASCII characters. The value of this field for files governed by this Recommendation | International Standard shall be 'jp2\040'.

This field only describes the governing Recommendation | International Standard for the file. Readers must examine the CL^i fields to determine if they can properly interpret the file.

Other values of the Brand field are reserved for ISO use.

CL^i: Compatibility list. This field specifies a code representing this Recommendation | International Standard, another standard, or a profile of another standard, to which the file conforms. This field is encoded as a four byte string of ASCII characters. A file that conforms to this Recommendation | International Standard shall have at least one CL^i field in the Profile box, and shall contain the value 'jp2\040' in one of the CL^i fields in the Profile box.

The number of CL^i fields is determined by the length of this box.

Table I-3 — Format of the contents of the Profile box

Field name	Size (bits)	Value
BR	32	$0—(2^{32}–1)$
CL^i	32	$0—(2^{32}–1)$

I.7.3 JP2 header box (superbox)

The JP2 header box contains generic information about the file, such as number of samples, colourspace, and resolution. This box is a superbox. The format of the Profile box is as follows:

Within a JP2 file (considered as a superbox), there shall be one and only one JP2 header box. The JP2 header box may be located anywhere within the file after the JP2 signature box but before the contiguous codestream box. It also must be at the same level as the JP2 signature box (it shall not be inside any other superbox within the file).

The type of the JP2 header box shall be 'jp2h' (X'6A703268').

This box contains several boxes. Other boxes may be defined in other standards and may be ignored by conforming readers. Those boxes contained within the JP2 header box that are defined within this Recommendation | International Standard are as follows:

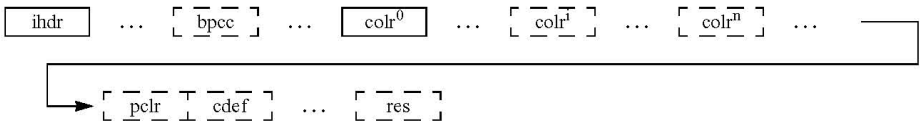


Figure I-5 — Organization of the contents of a JP2 header box

ihdr: Image Header Box. This box specifies information about the image, such as its height and width. Its structure is specified in Annex I.7.3.1. This box shall be the first box in the JP2 header box.

bpcc: BitsPerComponent box. This box specifies the bit depth of each component in the codestream after decompression. Its structure is specified in Annex I.7.3.2. This box may be found anywhere in the JP2 header box provided that it comes after the Image Header box.

colr^{i}: Colour Specification boxes. These boxes specify the colourspace of the decompressed image. Their structures are specified in Annex I.7.3.3. There shall be at least one Colour Specification box within the JP2 header box. The use of multiple Colour Specification boxes provides the ability for a decoder to be given multiple optimization or compatibility options for colour processing. These boxes may be found anywhere in the JP2 header box provided that they come after the Image Header box.

pclr: Palette box. This box defines the palette to use to create multiple components from a single component. Its structure is specified in Annex I.7.3.4. This box may be found anywhere in the JP2 header box provided that it comes after the Image Header box.

cdef: Component Definition box. This box defines the components in the codestream. Its structure is specified in Annex I.7.3.5. This box may be found anywhere in the JP2 header box provided that it comes after the Image Header box.

res: Resolution box. This box specifies the capture and default display resolutions of the image. Its structure is specified in Annex I.7.3.6. This box may be found anywhere in the JP2 header box provided that it comes after the Image Header box.

I.7.3.1 Image Header box

This box contains fixed length generic information about the image, such as the image size and number of components. The contents of the JP2 header box shall start with an Image Header box. Instances of this box in other places in the file shall be ignored. The length of the Image Header box shall be 24 bytes, including the box length and type fields. Note that much of the information within the Image Header box is redundant with information stored in the codestream itself.

The type of the Image Header box shall be 'ihdr' (X'69686472') and contents of the box shall have the following format:

Figure I-6 — Organization of the contents of an Image Header box

VERS: Version. This parameter defines the version number of this JP2 specification for which the file complies. The parameter is defined as a 2-byte big endian unsigned integer with the most significant byte containing the major version number (currently defined as 1) and the least significant byte containing a minor revision number (currently defined as 0).

The value of this field is X'0100.'

A major version number increment (if there ever is one) represents an incompatible change in JP2 files. Decoders should give up if they encounter an unrecognized major version number. Minor version

number increments represent backward compatible changes. Decoders should continue to process JP2 files even if the minor version number is unrecognized.

NC: Number of components. This parameter specifies the number of components in the image and is stored as a 2-byte big endian unsigned integer.

HEIGHT: Image height. The value of this parameter indicates the number of lines of the rendered image. If the file contains only one codestream, then this value shall be the same as the value of the Ysiz parameter in the SIZ marker segment in that codestream. Otherwise, this field specifies the height of the image into which the sequence of codestreams are rendered. This field is stored as a 4-byte big endian unsigned integer.

WIDTH: Image width. The value of this parameter indicates the number of samples per line of the rendered image. If the file contains only one codestream, then this value shall be the same as the value of the Xsiz parameter in the SIZ marker segment in that codestream. Otherwise, this field specifies the width of the image into which the sequence of codestreams are rendered. This field is stored as a 4-byte big endian unsigned integer.

BPC: Bits per component. This parameter specifies the bit depth of the components in the image and is stored as a 1-byte field.

If the bit depth is the same for all components, then this parameter specifies the actual bit depth. If the components vary in bit depth, then the value of this field shall be zero and the JP2 header box shall also contain a BitsPerComponent box defining the bit depth of each component (as defined in Annex I.7.3.2).

The low 7-bits of the value indicate the bit depth of the components. The high-bit indicates whether the components are signed or unsigned. If the high-bit is 1, then the components contain signed values. If the high-bit is 0, then the components contain unsigned values.

C: Compression type. This parameter specifies the compression algorithm used to compress the image data. The value of this field shall be 7. It is encoded as a 1-byte unsigned integer. If the value of this field is not 7, then this file is not a conforming JP2 file.

UnkC: Colourspace Unknown. This field specifies if the actual colourspace of the image data is known. This field is encoded as a 1-byte unsigned integer. Legal values for this field are 0, if the colourspace of the image is known and correctly specified the colourspace boxes within the file, or 1, if the colourspace of the image is not known. A value of 1 will be used in cases such as the transcoding of legacy images where the actual colourspace of the image data is not known. In those cases, while the colourspace interpretation methods specified in the file may not accurately reproduce the image with respect to some original, the image should be treated as if the methods do accurately reproduce the image. Values other than 0 and 1 are reserved for other use.

IPR: Intellectual Property. This parameter whether this JP2 file contains intellectual property rights information. If the value of this field is 0, this file does not contain rights information, and thus the file does not contain an IPR box. If the value is 1, then the file does contain rights information and thus does contain an IPR box as defined in Annex I.8. Other values are reserved for ISO use.

Table I-4 — Format of the contents of the Image Header box

Field name	Size (bits)	Value
VERS	16	X'0100'
NC	16	1—(2^{16}–1)
HEIGHT	32	1—(2^{32}–1)
WIDTH	32	1—(2^{32}–1)
BPC	8	-127—127

Table I-4 — Format of the contents of the Image Header box

Field name	Size (bits)	Value
C	8	7
Unk	8	0—1
IPR	8	0—1

I.7.3.2 BitsPerComponent box

The BitsPerComponent box specifies the bit depth of each component. If the bit depth is constant across all components in the codestream, then this box shall not be found. Otherwise, this box specifies the bit depth of each component. The order of bit depth values in this box is the actual order those components are enumerated within the codestream. The exact location of this box within the JP2 header box may vary provided that it follows the Image Header box.

The type of the BitsPerComponent Box shall be 'bpcc' (X'62706363'). The contents of this box shall be as follows:

Figure I-7 — Organization of the contents of a BitsPerComponent box

BPC^i: Bits per component. This parameter specifies the bit depth of component *i*, encoded as a 1-byte ones-complement integer. The ordering of the components within the BitsPerComponent Box shall be the same as the ordering of the components within the codestream. The number of BPC^i fields shall be the same as the value of the NC field from the Image Header box.

The low 7-bits of the value indicate the bit depth of this component. The high-bit indicates whether the component is signed or unsigned. If the high-bit is 1, then the component contains signed values. If the high-bit is 0, then the component contains unsigned values.

Table I-5 — Format of the contents of the BitsPerComponent box

Field name	Size (bits)	Value
BPC^i	8	-127— -1, 1—127

I.7.3.3 Colour Specification box

Each Colour Specification box defines one method by which an application can interpret the colourspace of the decompressed image data. A JP2 file may contain multiple Colour Specification boxes, specifying different methods for achieving "equivalent" results. Note that this colour specification is to be applied to the image data after it has been decompressed and after any reverse decorrelating component transform has been applied to the data. A conforming JP2 shall ignore all Colour Specification boxes after the first.

The type of a Colour Specification box shall be 'colr' (X'636F6C72'). The contents of a Colour Specification box is as follows:

Figure I-8 — Organization of the contents of a Colour Specification box

METH: Specification method. This field specifies the method used by this Colour Specification box to define the colourspace of the decompressed image. This field is encoded as a 1-byte unsigned integer. Legal values of this field are as follows:

Table I-6 — Legal METH values

Value	Meaning
1	**Enumerated colourspace**. This colourspace specification box contains the enumerated value of the colourspace of this image. The enumerated value is found in the EnumCS field in this box. If the value of the METH field is 1, then the EnumCS shall exist in this box immediately following the APPROX field, and the EnumCS field shall be the last field in this box
2	**Restricted ICC profile**.This Colour Specification box contains a Restricted ICC profile in the PROFILE field. This profile specifies the transformation needed to convert the decompressed image data into the PCS. If the value of METH is 2, then the ICC profile shall conform to the definition of either a Monochrome Input Profile or a Three-Component Matrix-Based Input Profile as defined in the ICC profile specification, version 2.2.0. In addition, the value of the Profile Connection Space field in the profile header in the embedded profile shall be 'XYZ ' (X'58595A20') indicating that the output colourspace of the profile is XYZ data. Note that the components from the codestream may have a range greater than the input range of the tone reproduction curve (TRC) of the ICC profile. Any decoded values should be clipped to the limits of the TRC before processing the image through the ICC profile. For the JP2 file format, profiles shall conform to the ICC profile definition as defined by the ICC Profile Format Specification, version 2.2.0, as well as the restrictions specified above. See Annex J.5 for a more detailed description of the legal colourspace transforms, how those transforms are stored in the file, and how to process an image using that transform without using an ICC colour management engine. If the value of METH is 2, then the PROFILE field shall immediately follow the APPROX field and the PROFILE field shall be the last field in the box.
other values	Reserved for other ISO use. If the value of METH is not 1 or 2, there may be fields in this box following the APPROX field. Those fields shall be ignored.

PREC: Precedence. This field is reserved for ISO use and the value shall be set to zero; however, conforming readers shall ignore the value of this field. This field is specified as a signed 1 byte integer.

APPROX: Colourspace approximation. This field specifies the extent to which this colour specification method approximates the "correct" definition of the colourspace. The value of this field shall be set to zero; however, conforming readers shall ignore the value of this field. Other values are reserved for other ISO use. This field is specified as 1 byte unsigned integer.

EnumCS: Enumerated colourspace. This field specifies the colourspace of the image using integer codes. To correctly interpret the colour of an image using an enumerated colourspace, the application must know the definition of that colourspace internally. This field contains a 4-byte big endian unsigned integer value indicating the colourspace of the image. If the value of the METH field is 2, then the EnumCS field shall not exist. Valid EnumCS values for the first colourspace specification box in conforming files are limited to 16 and 17 as defined in Table I-7:

PROFILE: ICC profile. This field contains a valid ICC profile, as specified by the ICC Profile Format Specification, which specifies the transformation of the decompressed image data into the PCS. This field shall not exist if the value of the METH field is 1. If the value of the METH field is 2, then the ICC

Table I-7 — Legal EnumCS values

Value	Meaning
16	sRGB as defined by IEC 61966–2–1
17	greyscale: A greyscale space where image luminance is related to code values using the sRGB non-linearity given in Eqs. (2) through (4) of IEC 61966–2–1 (sRGB) specification: $Y' = Y_{8bit}/255$ I.1 $for(Y' \leq 0.04045),\ Y_{lin} = Y'/12.92$ $for(Y' > 0.04045),\ Y_{lin} = \left(\frac{Y' + 0.055}{1.055}\right)^{2.4}$ I.2 where Y_{lin} is the linear image luminance value in the range 0.0 to 1.0. The image luminance values should be interpreted relative to the reference conditions in Section 2 of IEC 61966–2–1.
other values	Reserved for other ISO uses

profile shall conform to the Monochrome Input Profile class or the Three-Component Matrix-Based Input Profile class as defined in the ICC profile specification.

Table I-8 — Format of the contents of the Colr box

Field name	Size (bits)	Value
METH	8	1—2
PREC	8	0
APPROX	8	0
EnumCS	32 if METH=1 0 if METH=2	0—(2^{32}–1) no value
PROFILE	Varies	Varies

I.7.3.4 Palette box

The colour palette specified in this box is applied to the single colour component to convert the single value to a tuple. The colourspace of the generated tuple is then interpreted based on the values of the colour specification boxes in the JP2 Header box in the file.

The type of the palettized colour box shall be 'pclr' (X'70636C72'). The contents of this box shall be as follows:

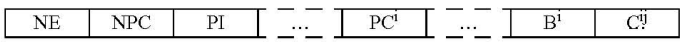


Figure I-9 — Organization of the contents of the Palette box

NE: Number of entries in the table. This value shall be in the range 1 to 1024.

NPC: Number of components created by the application of the palette. For example, if the palette turns a single index component into a three-component RGB images, then the value of this field shall be 3.

PI: Palette input. This field specifies the number of the component that should be used as the input to the palette (the index component). This field is encoded as a 2 byte unsigned integer, and the value of this field shall be less than the number of components specified by the NC field in the Image Header box

PC^i: Component number of palette created component *i*. This field specifies a number by which the component *i* of the palette table shall be referred. These values will be used by the Component Definition box to specify the individual components of the palette. This value shall be greater than the number of components specified in the Image Header Box, and shall not be the same as the value of any other PC^i field in this box. The number of PC^i fields shall be the same as the value of the NPC field.

B^i: This parameter specifies the bit depth of generated component *i*, encoded as a 8-bit integer. The low 7-bits of the value indicate the bit depth of this component. The high-bit indicates whether the component is signed or unsigned. If the high-bit is 1, then the component contains signed values. If the high-bit is 0, then the component contains unsigned values. The number of B^i values shall be the same as the value of the NPC field.

C^{ij}: The generated component value for entry *j* for component *i*. C^{ij} values are organized in component major order; all of the component values for entry *j* are grouped together, followed by all of the entries for component *j*+1. The size of C^{ij} is the value specified by field B^i. The number of components shall be the same as the NPC field. The number of C^{ij} values shall be the number of created components (the NPC field) x the number of entries in the palette (NE).

Table I-9 — Format of the contents of the Palette box

Field name	Size (bits)	Value
NE	16	1—1024
NPC	8	1—255
PI	16	0—(2^{16}–1)
PC^i	16	0—(2^{16}–1)
B^i	8	-127— -1, 1—127
C^{ij}	Varies	Varies

I.7.3.5 Component Definition box

The component definition box specifies the meaning of the data in each component in the codestream. The exact location of this box within the JP2 header box may vary provided that it follows the Image Header box.

This box contains an array of component descriptions. For each description, three values are specified: the number of the component described by that association, the type of that component, and the association of that component with particular colours. This box may specify multiple descriptions for a single component; however, the type value in each description for the same component shall be the same in all descriptions.

If the codestream contains only colour components and those components are ordered in the same order as the associated colours (for example, an RGB images with three components in the order R, G, then B), then this box shall not exist. If there are any auxiliary components or the components are not in the same order as the colour numbers, then the Component Definition box shall be found within the JP2 header box with a complete list of component definitions. However, if this file contains a Palette box, the component specified as input to the palette (in the PI field) shall not be listed in the Component Definition box.

If a multiple component transform is specified within the codestream, the component ordering box shall specify the existence of red, green and blue colours as components 0, 1 and 2 in the codestream, respectively.

ISO/IEC CD15444-1 : 1999 (V1.0, SDRA 1 March 2000)

The type of the Component Definition box shall be 'cdef' (X'63646566'). The contents of this box shall be as follows:

Figure I-10 — Organization of the contents of a Component Definition box

N: Number of component descriptions. This field specifies the number of component descriptions in this box. This field is encoded as a 2-byte big endian unsigned integer.

Cn^i: Component number. This field specifies the number of the component for this description. The value of this field represents the number of the component as defined within the codestream or created by the application of a palette to a single component codestream. The numbers of components created by the application of the palette are defined by the Palette box. This field is encoded as a 2-byte big endian unsigned integer.

Typ^i: Component type. This field specifies the type of the component for this description. The value of this field represents the type of data contained within the component. This field is encoded as a 2-byte big endian unsigned integer. Legal values of this field are as follows:

Table I-10 — Typ^i field values

Value	Meaning
0	This component is the colour component for the associated colour
1	Opacity. A sample value of 0 indicates that the sample is 100% transparent, and the maximum value of the component (related to the bit depth of the component) indicates a 100% opaque sample.
2	Premultiplied opacity. An opacity component as specified above, except that the value of the opacity component has been multiplied into the colour components for which this component is associated. Premultiplication is defined as follows: $$S_p = S \times \frac{\alpha}{\alpha_{max}} \quad \text{I.3}$$ where S is the original sample, S_p is the premultiplied sample (the sample stored in the image, α is the value of the opacity component, and α_{max} is the maximum value of the opacity component as defined by the bit depth of the opacity component.
3—(2^{16}–2)	Reserved for ISO use
2^{16}–1	The type of this component is not specified

Asoc^i: Component association. This field specifies the number of the colour for which this component is directly associated (or a special value to indicate the whole image or the lack of an association). For example, if this component is an opacity blending component for the red component in an RGB colourspace, this field would specify the number of the colour red. Table I-11 specifies legal association values. Table I-12 specifies legal colour numbers. This field is encoded as a 2-byte big endian unsigned integer.

Table I-11 — Asoc^i field values

Value	Meaning
0	This component is associated as the image as a whole (for example, a component independent opacity blending channel
1—(2^{16}–2)	This component is associated with the a particular colour as indicated by this value. This value is used to associate a particular component with a particular aspect of the specification of the colourspace of this image. For example, indicating that a component is associated with the red component of an RGB image allows the reader to associate that decoded component with the Red input to an ICC profile contained within a Colour Specification box. Colour indicators are specified in Table I-12
2^{16}–1	This component is not associated with any particular colour

Table I-12 — Colours indicated by the Asoc^i field

Class of colourspace	Colour indicated by the following value of the Asoc^i field			
	1	2	3	4
RGB	R	G	B	
Greyscale	Y			
The following colourspace classes are listed for future reference, as well as to aid in understanding of the use of the Asoc^i field				
XYZ	X	Y	Z	
Lab	L	a	b	
Luv	L	u	v	
YC_bC_r	Y	C_b	C_r	
Yxy	Y	x	y	
HSV	H	S	V	
HLS	H	L	S	
CMYK	C	M	Y	K
CMY	C	M	Y	
Jab	J	a	b	
n colour colourspaces	1	2	3	4

In this box, component numbers refer to the number of that particular component within the codestream. Colour numbers specify how that component shall be interpreted based on the specification of the colourspace of the image.

For example, the green colour in an RGB image is specified by a {Cn, Typ, Asoc} value of {*i*, 0, 2}, where *i* is the number of that component in the codestream (either directly or as generated by applying the reverse multiple component transform). Applications that are only concerned with extracting the colour components can treat the Typ/Asoc field pair

as a four-byte value where the combined value maps directly to the colour numbers (as the Typ field for a colour component shall be 0).

In another example, the codestream may contain a component *i* that specifies opacity blending data for the red and green components, and a component *j* that specifies opacity blending data for the blue component. In that file, the following {Cn, Typ, Asoc} tuples would be found in the Component Definition box: {*i*, 1, 1}, {*i*, 1, 2} and {*j*, 1, 3}.

There shall not be more than one component in a JP2 file with a the same Typ^i and $Asoc^i$ value pair, with the exception of Typ^i and $Asoc^i$ values of 2^{16}–1 (not specified). For example a JP2 file in an RGB colourspace shall only contain one green component, and a greyscale image shall contain only one grey component. There also shall not be more than one opacity component associated with a single colour component in an image.

Table I-13 — Component definition & ordering data structure values

Parameter	Size (bits)	Value
N	16	0—(2^{16}–1)
Cn^i	16	0—(2^{16}–1)
Typ^i	16	0—(2^{16}–1)
$Asoc^i$	16	0—(2^{16}–1)

I.7.3.6 Resolution box (superbox)

This box specifies the capture and default display resolution of this image. If this box exists, it shall contain either a capture display resolution box, or a default display resolution box, or both.

The type of a Resolution box shall be 'res ' (X'72657320'). The contents of the resolution box are as follows:

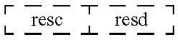


Figure I-11 — Organization of the contents of the Resolution box

resc: Capture resolution box. This box specifies the resolution at which this image was captured. The format of this box is specified in Annex I.7.3.6.1.

resd: Default display resolution box. This box specifies the default resolution at which this image should be displayed. The format of this box is specified in Annex I.7.3.6.2

I.7.3.6.1 Capture resolution box

This box specifies the resolution at which the source was digitized to create the image samples specified by the codestream. For example, this may specify the resolution of the flatbed scanner that captured a page from a book. The capture resolution could also specify the resolution of an aerial digital camera or satellite camera.

The vertical and horizontal capture resolutions are calculated using the six parameters (Table I-14) stored in this box in the following two equations, respectively:

$$VRc = \frac{VRcN}{VRcD} \times 10^{VRcE} \qquad \text{I.4}$$

$$HRc = \frac{HRcN}{HRcD} \times 10^{HRcE} \qquad \text{I.5}$$

The values *VRc* and *HRc* are always in samples/meter. If an application requires the resolution in another unit, then that application must apply the appropriate conversion.

The type of a Capture resolution box shall be 'resc' (X'72657363'). The contents of the Capture resolution box are as follows:

VRcN	VRcD	HRcN	HRcD	VRcE	HRcE

Figure I-12 — Organization of the contents of the Capture Resolution box

VRcN: Vertical Capture resolution numerator. This parameter specifies the *VRcN* value in Equation I.4, which is used to calculate the vertical capture resolution. This parameter is encoded as a 16-bit big endian unsigned integer.

VRcD: Vertical Capture resolution denominator. This parameter specifies the *VRcD* value in Equation I.4, which is used to calculate the vertical capture resolution. This parameter is encoded as a 16-bit big endian unsigned integer.

HRcN: Horizontal Capture resolution numerator. This parameter specifies the *HRcN* value in Equation I.5, which is used to calculate the horizontal capture resolution. This parameter is encoded as a 16-bit big endian unsigned integer.

HRcD: Horizontal Capture resolution denominator. This parameter specifies the *HRcD* value in Equation I.5, which is used to calculate the horizontal capture resolution. This parameter is encoded as a 16-bit big endian unsigned integer.

VRcE: Vertical Capture resolution exponent. This parameter specifies the *VRcE* value in Equation I.4, which is used to calculate the vertical capture resolution. This parameter is encoded as a twos-compliment 8-bit signed integer.

HRcE: Horizontal Capture resolution exponent. This parameter specifies the *HRcE* value in Equation I.5, which is used to calculate the horizontal capture resolution. This parameter is encoded as a twos-compliment 8-bit signed integer.

Table I-14 — Format of the contents of the Capture resolution box

Field name	Size (bits)	Value
VRcN	16	$1—(2^{16}–1)$
VRcD	16	$1—(2^{16}–1)$
HRcN	16	$1—(2^{16}–1)$
HRcD	16	$1—(2^{16}–1)$
VRcE	8	-128—127
HRcE	8	-128—127

I.7.3.6.2 Default display resolution box

This box specifies a default resolution at which the image should be displayed. For example, this may be used to determine the size of the image on a page when the image is placed in a page-layout program. Note, however, that this value is only a default. Each application must determine an appropriate display size for that application.

The vertical and horizontal display resolutions are calculated using the six parameters (Table I-15) stored in this box in the following two equations, respectively:

$$VRd = \frac{VRdN}{VRdD} \times 10^{VRdE} \quad \text{I.6}$$

$$HRd = \frac{HRdN}{HRdD} \times 10^{HRdE} \quad \text{I.7}$$

The values *VRd* and *HRd* are always in samples/meter. If an application requires the resolution in another unit, then that application must apply the appropriate conversion.

The type of a Default display resolution box shall be 'resd' (X'72657364'). The contents of the Default display resolution box are as follows:

VRdN	VRdD	HRdN	HRdD	VRdE	HRdE

Figure I-13 — Organization of the contents of the Default Display Resolution box

VRdN: Vertical Display resolution numerator. This parameter specifies the *VRdN* value in Equation I.6, which is used to calculate the vertical display resolution. This parameter is encoded as a 16-bit big endian unsigned integer.

VRdD: Vertical Display resolution denominator. This parameter specifies the *VRdD* value in Equation I.6, which is used to calculate the vertical display resolution. This parameter is encoded as a 16-bit big endian unsigned integer.

HRdN: Horizontal Display resolution numerator. This parameter specifies the *HRdN* value in Equation I.7, which is used to calculate the horizontal display resolution. This parameter is encoded as a 16-bit big endian unsigned integer.

HRdD: Horizontal Display resolution denominator. This parameter specifies the *HRdD* value in Equation I.7, which is used to calculate the horizontal display resolution. This parameter is encoded as a 16-bit big endian unsigned integer.

VRdE: Vertical Display resolution exponent. This parameter specifies the *VRdE* value in Equation I.6, which is used to calculate the vertical display resolution. This parameter is encoded as a twos-compliment 8-bit signed integer.

HRdE: Horizontal Display resolution exponent. This parameter specifies the *HRdE* value in Equation I.7, which is used to calculate the horizontal display resolution. This parameter is encoded as a twos-compliment 8-bit signed integer.

Table I-15 — Format of the contents of the Default display resolution box

Field name	Size (bits)	Value
VRdN	16	1—(2^{16}–1)
VRdD	16	1—(2^{16}–1)
HRdN	16	1—(2^{16}–1)
HRdD	16	1—(2^{16}–1)
VRdE	8	-128—127
HRdE	8	-128—127

I.7.4 Contiguous codestream box

The Contiguous codestream box contains a valid and complete JPEG 2000 codestream, as defined in Annex A of this Recommendation | International Standard. When displaying the image, a conforming reader shall ignore all codestreams after the first codestream found in the file.

The type of a contiguous codestream box shall be 'jp2c' (X'6A703263'). The contents of the box shall be as follows:

Figure I-14 — Organization of the contents of the Contiguous codestream box

Code: This field contains a valid and complete JPEG 2000 codestream as specified by Annex A of this Recommendation | International Standard.

Table I-16 — Format of the contents of the Contiguous codestream box

Field name	Size (bits)	Value
Code	Varies	Varies

I.8 Adding intellectual property rights information in JP2

This Recommendation | International Standard specifies an box type for an box which is devoted to carrying intellectual property rights information within a JP2 file. Inclusion of this information in a JP2 file is optional for conforming files. The definition of the format of the contents of this box is reserved for ISO. However, the type of this box is defined in this Recommendation | International Standard as a means to allow applications to recognize the existence of IPR information. Use and interpretation of this data is beyond the scope of this Recommendation | International Standard.

The type of the Intellectual Property Box shall be 'jp2i' (X'6A703269').

I.9 Adding vendor specific information to the JP2 file format

The following boxes provide a set of tools by which applications can add vendor specific information to the JP2 file format. All of the following boxes are optional in conforming files and may be ignored by conforming readers.

I.9.1 XML boxes

An XML box contains vendor specific data (in XML format) other than that data defined within this Recommendation | International Standard. There may be multiple XML boxes within the file, and those boxes may be found anywhere in the file except before the JP2 signature box.

The type of an XML box is 'xml\040' (X'786D6C20'). The contents of the box shall be as follows:

Figure I-15 — Organization of the contents of the XML box

DATA: This field shall be valid XML as defined by REC-xml-19980210.

The existence of any XML boxes is optional for conforming files. Also, any XML box shall not contain any information necessary for decoding the image to the extent that is defined within this part of this Recommendation | International Standard, and the correct interpretation of the data in any XML box shall not change the visual appearance of the image. All readers may ignore any XML box in the file.

I.9.2 UUID boxes

A UUID box contains vendor specific data other than that data defined within this Recommendation | International Standard. There may be multiple UUID boxes within the file, and those boxes may be found anywhere in the file except before the JP2 signature box.

The type of a UUID box shall be 'uuid' (X'75756964'). The contents of the box shall be as follows:

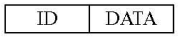


Figure I-16 — Organization of the contents of the UUID box

ID: This field contains a 16-byte UUID as specified by ISO/IEC 11578:1996. The value of this UUID specifies the format of the vendor specific data stored in the DATA field and the interpretation of that data.

DATA: This field contains the vendor specific data. The format of this data is defined outside of the scope of this standard, but is indicated by the value of the UUID field.

Table I-17 — Format of the contents of a UUID box

Field name	Size (bits)	Value
UUID	128	Varies
DATA	Varies	Varies

The existence of any UUID boxes is optional for conforming files. Also, any UUID box shall not contain any information necessary for decoding the image to the extent that is defined within this part of this Recommendation | International Standard, and the interpretation of the data in any UUID box shall not change the visual appearance of the image. All readers may ignore any UUID box.

I.9.3 UUID Info boxes (superbox)

While it is useful to allow vendors to extend JP2 files by adding binary data using UUID boxes, it is also useful to provide information in a standard form which can be used by non-extended applications to get more information about the extensions in the file. This information is contained in UUID Info boxes. A JP2 file may contain zero or more UUID Info boxes. These boxes may be found anywhere in the top level of the file (the superbox of a UUID Info box shall be the JP2 file itself) except before the signature box.

Note that these boxes, if present, may not provide a complete index for the UUID's in the file, may reference UUID's not used in the file, and possibly may provide multiple references for the same UUID.

The type of a UUID Info box shall be 'uinf' (X'75696E66'). The contents of a UUID Info box are as follows:

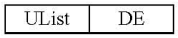


Figure I-17 — Organization of the contents of a UUID Info box

UList: UUID List box. This box contains a list of UUID's for which this UUID Info box specifies a link to more information. The format of the UUID List box is specified in Annex I.9.3.1.

DE: Data Entry URL box. This box contains a URL. An application can acquire more information about the UUID's contained in the UUID list box. The format of a Data Entry URL box is specified in Annex I.9.3.2

I.9.3.1 UUID List box

This box contains a list of UUID's. The type of a UUID List box shall be 'ulst' (X'75637374'). The contents of a UUID List box shall be as follows:

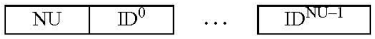


Figure I-18 — Organization of the contents of a UUID Info box

NU: Number of UUID's. This field specifies the number of UUID's found in this UUID List box. This field is encoded as a 16-bit big endian unsigned integer.

ID^i: ID. This field specifies one UUID, as specified in ISO/IEC 11578:1996, which shall be associated with the URL contained in the URL box within the same UUID Info box. The number of $UUID^i$ fields shall be the same as the value of the NU field. The value of this field shall be a 16-byte UUID.

Table I-18 — UUID List box contents data structure values

Parameter	Size (bits)	Value
NU	16	0—(2^{16}–1)
$UUID^i$	128	0—(2^{128}–1)

I.9.3.2 Data Entry URL box

This box contains a URL which can use used by an application to acquire more information about the associated vendor specific extensions. The format of the data acquired through the use of this URL is not defined in this Recommendation I International Standard. The URL type should be of a service which delivers a file (e.g. URL's of type file, http, ftp, etc.), which ideally also permits random access. Relative URL's are permissible and are relative to the file containing this data reference.

The type of a Data Entry URL box shall be 'url\040' (X'75726C20'). The contents of a Data Entry URL box shall be as follows:

Figure I-19 — Organization of the contents of a URL box

VERS: Version number. This field specifies the version number of the format of this box. The value of this field shall be 0.

FLAG: Flags. This field is reserved for other use to flag particular attributes of this box. The value of this field shall be 0.

LOC: Location. This field specifies the URL of the additional information associated with the UUID's contained in the UUID List box within the same UUID Info superbox. The URL is encoded as a null terminated string of UTF-8 characters

Table I-19 — URL box contents data structure values

Parameter	Size (bits)	Value
VERS	8	0
FLAG	24	0
LOC	varies	varies

I.10 Dealing with unknown boxes

A valid codestream may contain boxes not known to applications based solely on this Recommendation | International Standard. If a conforming reader finds an box that it does not understand, it shall skip and ignore that box.

Appendix U

U. ZIP File Format Specification

The archive zip format used in CDB Spec 3.0 is based on

```
APPNOTE.TXT - .ZIP File Format Specification

URL: http://www.pkware.com/documents/APPNOTE/APPNOTE-6.3.1.TXT

Version: 6.3.1
Revised: April 11, 2007
Copyright (c) 1989 - 2007 PKWARE Inc., All Rights Reserved.
The use of certain technological aspects disclosed in the
currentAPPNOTE is available pursuant to the below section
entitled "Incorporating PKWARE Proprietary Technology into Your
Product".
```

CDB zip compliant reader is required to support as a minimum the following features defined in APPNOTE.TXT:

- Local file header (Note: Extra field can be inserted but not required to be read)
- File data
- Data descriptor:
- Central directory structure (Note: Digital signature is supported but will not be read)
- End of central directory record: (Note: ZIP file comments are supported but will not be read)

The compression methods supported:

- No compression
- Deflate (Enhanced Deflate is not required to be supported)

The following features are not required to be supported thus are optional and left to the implementation

- Archive decryption header:
- Archive extra data record.
- Zip64 end of central directory record
- Zip64 end of central directory locator
- Splitting and Spanning ZIP files
- Encryptions of any type

Note that anything not listed in this section is by default assumed not to be supported.