DSF Usage In X-Plane

Last updated February 22, 2024

Revision History:

2/22/23	Updated for X-Plane 11, 12
7/ 6/16	Added data validation rules
12/ 8/13	Clarified DSF DEM handling
2/18/12	Updated to docuemnt 7z
2/12/12	Updated for X-Plane 1000
2/ 4/09	Updated for X-Plane 930
12/26/07	Updated For X-Plane 900
7/25/07	Initial Draft

This document outlines how the DSF file format is used by X-Plane. DSF is a container format; new features can be added to the X-Plane scenery system without changing the file format. This document lists the different legal DSF configurations that X-Plane understands.

Note: this document is a low level reference, intended for programmers who intend to create tools to edit DSF files. Authors who want to edit DSFs should simply use higher level tools like OverlayEditor, or PhotoSceneryX.

DSF Compression

Starting with X-Plane 10, X-Plane will read 7z-compressed DSF files (a single DSF compressed into a 7z archive) natively. X-Plane thus installs its DSFs without decompressing them, to save disk space. You may need to un-7z DSFs to read them. 7z compression is optional.

DSF Properties

DSF contains a series of properties, with string values. These are the properties X-Plane recognizes:

Bounding Box and Location Properties

All DSFs must contain four properties indicating the bounds of the DSF tile. These bounds are in degrees and must be integers. DSFs should contain a planet tag, which is optional, and assumed to be earth if missing.

Property Name	Default	Value
sim/north	(Required)	degrees
sim/south	(Required)	degrees
sim/east	(Required)	degrees
sim/west	(Required)	degrees
sim/planet	earth	mars

Object Density Control Properties

X-Plane only loads part of the facades and objects in a DSF, based on the "number of objects" setting in the rendering options. The "require" properties force X-Plane to load certain objects. Each require statement applies to either facades or objects, and specifies both the minimum setting where the objects are guaranteed loaded and the minimum index within the DSF of the object/facade it applies to.

More than one requirement statement can be used; all are combined together to meet all requirement constraints. Thus you can bring in objects incrementally through proper organization of the DSF object definition order.

Property Name	Value	Minimum Sim Version
sim/require_object	obj_level/first_required_index	8.0
sim/require_agp	agp_level/first_required_index	10.0
sim/require_facade	obj_level/first_required_index	8.0

Overlay and Exclusion Properties

The overlay properties control how a DSF is used as an overlay to other DSF files. The overlay property signals to X-Plane that the DSF should be loaded as an overlay and not a base mesh.

The exclude properties cause X-Plane to eliminate scenery from lower priority DSFs that are loaded underneath the overlay. Each exclusion zone is a rectangle specified in latitude and longitude. A DSF may contain multiple exclusion zones of the same type, and they may overlap.

Property Name	Value
sim/overlay	1

Property Name	Value
sim/exclude_obj	polygonal exclusion
sim/exclude_fac	polygonal exclusion
sim/exclude_for	rectangular exclusion
sim/exclude_bch	polygonal exclusion
sim/exclude_net	polygonal exclusion
sim/exclude_lin	polygonal exclusion
sim/exclude_pol	polygonal exclusion
sim/exclude_str	polygonal exclusion
sim/filter/aptid	airport ID

The quality of culling depends on the type of scenery being excluded. In some cases, culling may over-remove or under-remove scenery.

[X-PLANE 9] X-Plane 9 improves the quality of forest exclusion. While X-Plane 8 forest exclusions remove a forest polygon if any vertex is in the exclusion zone, X-Plane 9 forest exclusion zones exclude on a per-tree basis for more precise cuts.

A rectangular exclusion is four floating point coordinates lon/lat coordinates in the form of

west/south/east/north

[X-PLANE 12] A polygonal exclusion is a rectangular exclusion (for compatibility) or:

west/south/east/north;lon1,lat1,lon2,lat2,lon3,lat3,lon4,lat4

The rectangle should be a bounding box around the polygon, and the polygon should contain at least three points and no overlapping edges or holes. Concave exclusions are allowed. Like rectangular exclusions, the exclusion is a containment test with any point in the geometric primitive, so containing a single corner of an AGS will remove the entire AGS, for example.

The intent of poylgonal exclusions is to allow simple shapes to be modeled directly instead of using a large number of thin axis aligned bounding boxes. The outer AABB provides backward compatibility with X-Plane 11.

[X-PLANE 11.50] X-Plane 11.50 can exclude by airport ID – see the section on custom DSF comments for more. Each sim/filter/aptid establishes a zero-based indexing scheme to airport IDs for this purpse.

LOD Properties

[X-Plane 930:] Historically, X-Plane measures the distance of a mesh patch based on an arbitrary center point computed from the geometry. Since two different LOD mesh patches may have different vertices, their centers will be different. This cannot be predicted by authors. So for example, if a first patch has an LOD of 0 -> 20000 and the second one has 20000 -> -1, the first mesh patch may not disappear when the second one appears because they are being measured using different center points.

Setting the property sim/lod_mesh to 1 changes X-Plane's behavior in the following way: the center point for LOD calculations of a mesh patch with a non-zero LOD start value will be taken from the previous mesh patch in the command stream.

This in turn means that a series of consecutive mesh patches with increasing LODs (starting at 0) will all have the same center point, and can be switched between using consecutive LOD values. (Also note that when this option is used, the closest LOD must come first, to establish the center point!

Other Properties

For historical reasons X-Plane will only flatten a terrain mesh if these properties are present.

Property Name	Value
sim/creation_agent	X-Plane Scenery Creator 0.9
sim/internal_revision	0

Raster Layers

X-Plane interprets multiple types of raster data in the DSF:

DEM Name	Contents	Supported
elevation	MSL Elevation Data	X-Plane 10
sea_level	Bathymetric Depth	X-Plane 11
soundscape	Sound codes	X-Plane 12
spr1	Seasons: start of spring, day of year	X-Plane 12
spr2	Seasons: end of spring, day of year	X-Plane 12
sum1	Seasons: start of summer, day of year	X-Plane 12
sum2	Seasons: end of summer, day of year	X-Plane 12
fal1	Seasons: start of fall, day of year	X-Plane 12

DEM Name	Contents	Supported
fal2	Seasons: end of fall, day of year	X-Plane 12
win1	Seasons: start of winter, day of year	X-Plane 12
win2	Seasons: end of winter, day of year	X-Plane 12

Elevation Raster Data

[X-PLANE 10] Elevation should be specified via a raster DEM file. If the vertex elevation is the flag value -32768.0 then the DEM elevation is sampled. This method is recommended because X-Plane can use the elevation DEM for other purposes as well as meshing.

Bathymetric Data

[X-PLANE 11] The depth of the sea floor is specified via a bathymetric DEM; elevations are absolute MSL meters. The behavior of the depth vertex variable varies by version and data.

Sim Version	Bathymetry Present	DSF Depth Coord/Flag
X-Plane 10	No (not used)	Depth
X-Plane 11	No	Depth
X-Plane 11	Yes	0 for coastline, 1 for use bathymetry
X-Plane 12	Yes (required)	ratio for interpolation: 0 is ground elevation, 1 is bathymetry sam

Sound Raster Data

[X-PLANE 12] A sound raster file specifies codes that drive environmental sounds throughout the scenery. Codes are:

0 or invalid = barren

30 = water (we don't differentiate between water body types because we don't have the data on the DSF yet)

40 = forest

50 = rural

60 = urban low

80 = urban town

100 = urban high

120 = industrial

Airport sounds replace some of these data points based on the apt.dat data loaded at runtime.

Seasonal Data

[X-PLANE 12] X-Plane 12 supports a set of 8 seasonal raster files that define the time of year of seasons. Each of the four seasons has a start and end day; within that range, the season is selected; between the ranges (e.g. blended between end of summer and start of fall) the seasonal art is interpolated.

All eight raster layers must be present. Days are days since January 1, and seasons can “wrap aorund”, e..g winter could start on day 310 and end on day 20.

Mesh Types and Coordinate Organization

X-Plane uses .ter files to specify the way mesh patches are drawn. The coordinate organization is:

1. Longitude
2. Latitude
3. Elevation
4. Normal – X
5. Normal – Z
6. Additional Coordinates...

These .ter files may contain “border” textures–the border feature of a .ter file is only used if the overlay flag is set in the mesh patch.

Mesh normals: the normal vector is stored as the X and Z ratio of the normal vector, based on a coordinate system of Y = up and Z = north at the mesh point’s location.

Additional coordinates are ordered optionally S1, T1, then optionally S2, T2. If an odd coordinate is provided, it is treated as alpha. If an alpha is needed but not present, X-Plane generates one using seeded random numbers.

Base Texture is Projected	**Composite Texture Is Projected**	**Border+Overlay Flag**	**ST1 Controls**	**S**
no	no	no	base	
no	no	yes	base	b
no	yes	no	base	
no	yes	yes	base	b

Base Texture is Projected	Composite Texture Is Projected	Border+Overlay Flag	ST1 Controls	S
yes	no	no	base	
yes	no	yes	base	b
yes	yes	no		
yes	yes	yes	border	

X-Plane 8 does not use the alpha channel right now that a DSF may have.

Water Handling

A mesh layer with the name water or terrain_Water is treated as water data mesh triangles; unlike regular mesh triangles, no .ter file is provided.

Water mesh triangles have a number of properties that are unique to water.

Water meshes ignore the normal parameters and have two ST coordinate controls after them:

- "Fetch ratio", a ratio from 0.0 to 1.0 that controls the scaling of waves. Use 1.0 for open ocean and 0.0 for ponds.
- Depth. This is an actual depth measurement for X-Plane 10 and earlier. In X-Plane 11 and later, *if* a bathymetric DEM is present, then this is a flag: 0.0 for coastline and 1.0 for in-water. See raster data handling for more info.

In X-Plane 12, if a .ter file has the WATER_COLOR_MASK directive, then it is water provided via a .ter file. In this case, four ST coordinates are expected: fetch ratio, bathymetric depth flag (this directive should **always** be used with raster bathymetric depth) and a pair of ST coordinates defining the UV mapping for the water texture in the .ter file.

Raster Data and Meshes

If elevation raster data is present, it will be used for the elevation of a mesh point as long as:

- The patch vertex's elevation is -32768.0 or
- The patch vertex's terrain type is water.

If elevation rater data is present, all normal vectors can be left as 0.0 – X-Plane will calculate them from raster data, for all terrain types.

(By convention v10 DSFs produced with LR's scenery tools use explicit elevation for all water vertices and all coastal vertices, to ensure precise water elevation even near dams and to keep a water-tight seal between land and water. Interior land elevation points come from raster DEMs for data compression.)

Object Types and Coordinate Organization

Objects are placed with three or four coordinate values:

1. Longitude
2. Latitude
3. Heading
4. MSL height (optional in v10)

X-Plane 8 and 9 only allow AGL positioned objects (3 coordinates); X-Plane 10 allows for an optional 4th coordinate, interpreted as the MSL height of the object in meters. (See Special DSF Comments for AGL placement).

AG Points (X-Plane 10 only) may only have three coordinates (lon, lat, heading) and are always draped.

Polygon Types and Coordinate Organization

Only one beach .bch definition may be used per DSF. Subtypes within the beach are used to create variety.

X-Plane uses a number of graphic resource files for DSF polygons. The meaning of the coordinates varies based on the type.

File Type	Minimum Sim Version	Holes Allowed?	Parameter Meaning
Facade (Flat, No Wall Choice)	8.0	No	Height (meters)
Facade (Flat, With Wall Choice)	10.0	No	Height (meters)
Facade (Curved, No Wall Choice)	10.0	No	Height (meters)
Facade (Curved, Wall Choice)	10.0	No	Height (meters)
Forest	8.0 (10.0 for line and point fill, 12.0 for	Yes	Density (0-255) + Fill

File Type	Minimum Sim Version	Holes Allowed?	Parameter Meaning
	height/MSL control)		(0=solid,256=line,512=
Beach (MSL)	8.0	No	0=chain,1=ring
Beach (AGL)	10.0	No	0=chain,1=ring
Line (straight)	8.5	No	0=chain,1=ring
Line (curved)	8.5	No	0=chain,1=ring
String	8.5	No	Spacing (meters)
String (Curved)	8.5	No	Spacing (meters)
Draped Polygon (no UV)	8.5	Yes	Texture Heading
Draped Polygon (curved, no UV)	8.5	Yes	Texture Heading
Draped Polygon (with UV Map)	8.5	Yes	65535
Draped Polygon (curved, with UV Map)	8.5	Yes	65535
Autogen Block	10.0	No	Block Code + 256 * (He
Autogen String	10.0	Yes	Number of Active Side

For forests, the density is a scaling factor–255 makes the maximum number of trees, 0 makes none. This control is multiplied by the rendering settings to set a final number of trees. Tree density will not exceed the maximum possible density from the .for file.

X-Plane 10 : in X-Plan 10, a packing code is added to forest density. 0 gives the default behavior of filling the polygon with trees. 256 gives the behavior of plotting trees along every line of the polygon, treating each contour as a poly-line. 512 gives the behavior of plotting a tree on each point in every contour. Note that:

- Poly-lines are not auto-closed with line filling (to allow for U shapes) so you must duplicate the final point to make a ring.
- In point-fill mode, all points are equal, so there is no advantage to using contour rings.

For beaches, the parameter can specify a ring, which connects the end point to the beginning. This will create a correct texture transition from the end to the beginning. The dx and dz coordinates for the beaches are a normal vector, similar to a DSF’s normal vector, and are used for draping the beach. The subtype parameter is an integral subtype which describes which beach from within the .bch file is used.

X-Plane 10 : in X-Plane 10, the normal vectors and elevation of beaches can optionally be omitted, as X-Plane derives this information on the fly.

For draped lines (.lin), object strings (.str) and draped polygons (.pol) if bezier coordinates are present, then bezier curves are generated.

For object strings, the spacing of objects is controlled by the polygon parameter. For draped lines, the polygon may be treated as a ring or chain. Like beaches, best results come from using the ring feature rather than duplicating the first point. For a draped polygon, texture coordinates (ST from 0 to 1) may also be included–the parameter value 65535 indicates this.

X-Plane 12: in X-Plane 12 for draped polygons whose parameter is not 65535 (e.g. texture projected by heading, not UV mapped) if the heading exceeds 359, then the integral heading divided by 360 is used as 128th of a degree and added to the heading modulo 360, to provide sub-degree resolution. In other words:

real heading = (dsf heading % 360) + floor(dsf heading / 360) / 128.0

For both autogen blocks (.agb) and autogen points (.ags) the height of variable height elements is encoded in the upper 8 bits of the parameter (e.g. * 256) and represents the metric height divided by four. (In other words, some precision in height is lost to allow for a wider range of building height.

For autogen blocks, the lower 8 bits of the polygon represent a spelling set code – this is used to look up which “spelling set” in the autogen block to use. A typical use is to use different tile arrangements based on different block codes. X-Plane’s global scenery, for example, uses codes 0-7 to indicate whether the “back 3” walls are road adjacent or not.

Autogen strings represent the strangest polygon feature of all. Unlike other polygons, the contours in AGS are interpreted as poly-lines (*not* closed polygon rigns!); the AGS is required to be a closed polygon with holes when *all* contours are considered.

The first N contours (where N is the lower 8 bits of the polygon parameter) will spawn autogen buildings; the rest of the contours are used only to create a closed polygon-with-holes area.

A few examples may help clarify autogen strings:

- In the case of a single rectangle city block with houses on all sides, there would be one contour with 5 points (the start point must be duplicated) and the polygon parameter N=1.

- In the case of a single rectangle city block where the north side has no houses, there would be two contours: the first contour would contain the NE, SE, SW, NW points and the second woul contain the NW, NE points. N=1 because only the first contour has houses.
- In the case where only the east and west sides of the block have houses, there would be four contours: NE,SE then SW, NW, then NW, NE, then SE,SW. N=2. Note that the first two and last two contours can swap with each other.
- In the case where a city block has houses on all sides but a lake in the middle, the first contour is the block (with a dupe point to close), the second is the lake (with a dupe point to close) and N=1.

X-PLANE 12: in X-Plane 12, a forest in **point mode** (and only point mode) can optionally have a third and fourth data coordinate plane; these planes are used to control tree height on a per tree basis and vertical registration if desired. There is no mode to get randomized heights with fixed MSL locations.

Road Types and Coordinate Organization

Only one road.net definition may be used per DSF. Subtypes within the road file are used to create variety.

Road network files use 4 coordinates.

1. Longitude
2. Latitude
3. Elevation
4. Junction ID

Road segments are connected via junction IDs with the following rules:

- The DSF file's junction IDs must start at 1 and contain no gaps. 0 is reserved as the "no junction" flag.
- A road chain must start and end with a junction.
- A junction must be used any time an intersection is desired.
- A junction must be used any time a road chain changes subtype.
- All nodes that share the same junction code must share the same coordinates.

- A junction should not be used for nodes that are designed only to change the path of a road ("shape" points), because the processing overhead is higher for junctions.

X-Plane 10 : if the road.net specifies a draped road type then the elevation should be a stacking layer number (0 for the ground, then 1, 2, 3, etc.) for all junctions to specify overpasses. Within the chains, the shape point should be 0 for a vertex and 1 for a bezier curve control point. There must be no more than two consecutive control points in a road. (That is, quadratic and cubic bezier curves are allowed but no higher degere polynomials.) All bridge-crossing roads should use a junction so that the sim can ensure separation.

DSF Feature Extensions Via Comments

Airport-ID Based Exclusion/Filtering

Starting in X-Plane 10.45, sections of DSF overlays can be filtered by airport ID. The mechanism works as follows:

1. The properties table contains sim/filter/aptid properties that establish an indexing scheme to particular X-Plane airport IDs.
2. X-Plane determines whether a given airport ID is "owned" by the scenery pack that contains the DSF. If the airport is in the apt.dat of this pack (and not in a higher-priority pack) then the airport ID is owned.
3. When a filter directive sets the filter to a given index, all following network segment, polygon, and point features are excluded unless the airport is part of this pack. Filtering is done at the interpretation level; DSF command state is *not* skipped.

Filter commands are encoded via a DSF comment with the following format six-byte format:

uint16 comment type - must be 1

sint32 airport index - can be -1 to clear the filter or [0...properties)

Like all DSF command table data, these ints are little endian. Filter state is considered to be "off" by default.

The intent of this feature is to allow a DSF to place objects that are "part" of an airport and will be removed if a higher priority pack replaces the airport, *even* if the higher priority pack does not correctly provide exclusion zones.

AGL-Offset OBJ Placement

Starting in X-Plane 11.50, explicit-height OBJs could have their datum changed from MSL to AGL. The format of the comment is:

uint16 comment type - must be 2

sint32 AGL mode - 0 for MSL, 1 for AGL

The flag affects all point placements for OBJs until the mode is further changed; the default mode is MSL.

Overlay DSF Restrictions

DSF overlays have restrictions on the types of files they may use:

- **[X-Plane 8:]** Road networks are not allowed in overlays.
- Mesh patches are not allowed in overlays.

X-Plane 9 relaxes the road network rule–in X-Plane 9 a DSF overlay may contain a road network, but the one-.net-per-DSF rule still holds. When a DSF overlay has a road network and the base mesh does too, the junction IDs between the two do **not** connect.

Guidelines for DSF Extension

- Consider all properties starting with “sim/” as reserved.
- Do not add extra coordinates to vertices beyond what is in this spec.

Data Validation

These rules place limits on the kinds of data that can be specified for various art assets.

Base Mesh

- Every point within the lat-lon rectangular boundaries of the DSF must be covered by exactly one triangle that is ‘hard’ (meaning the hard flag in its parent patch is set).
- It is illegal for a triangle’s vertex to be located on the edge of another triangle; triangles must only meet vertex-to-vertex, not vertex-to-edge. (In other words, there can be no “T” junctions in the mesh.) Triangles meet at vertices if the coordinates of their vertices have the exact same lat, lon and elevation bit-values.

Objects/AGPs

- Objects must be within the DSF lat/lon boundaries.
- Object heading should be between [0 and 360).

Polygons

Area rules (these apply to facades with roofs, draped polygons, filled forests, AGS and AGB).

- All polygons must have counter-clockwise winding for exteriors and clockwise winding for holes.
- Polygons must not be self-intersecting.

Line rules (these apply to all polygons except for forests in “points” mode).

- Polygons must not have zero length sides.

Roads

- All roads chains must be made of at least one segment.
- All road segments must have positive length.
- It is illegal for a road segment to reverse direction (E.g. have a 180 degree turn).
- A junction must not have two roads entering the junction at the same heading and the same level.
- No more than two shape points within a chain can be bezier control points. (In other words, a cubic bezier is the highest degree bezier supported in curved roads.)
- While a bezier control point may be outside the DSF boundaries (typically the point pools contain some margin to allow this) the actual path of the road segment must remain within the DSF boundaries.

While it is not necessary, it is recommended that crossing roads share a junction at the crossing point with the roads on different levels; X-Plane can use this information to try to ensure that the roads are not reordered vertically due to the road draping and smoothing process.

Using Decals to Add Detail To Scenery

Last updated May 30, 2014

Decals are overlay textures that sit on top of a regular albedo (day-time) texture to add high-frequency, high-res detail to an existing texture.

A typical use of decals might be:

- To add a grassy leafy pattern to a field or airport grass texture.
- To add a gravel pattern to a runway or road.

Decals work by modulating the textures they overlay. Decals typically repeat much faster than the base texture, but may also be used at a very low frequency to add subtle changes in the overall brightness of the terrain across a very large area, helping to disguise repetition.

Decals contain high frequency pixels at high res; when viewed from far away, the entire decal blends together into a constant color/grayscale. The decal shader ensures that when a decal has become a solid tone (because it is far away) the net result is the same as your underlying albedo. In other words, as you move away from the decal, the decal blurs into "no effect". This means that you can generally apply decals and not worry about the effect on the far-away view.

Decal Texture Formats

A texture typically has two decals together:

- One RGB decal in the color components of the texture.
- One alpha decal in the alpha channel.

If you do not need the RGB decal, you can save VRAM by having a decal with only alpha (this is done using a gray-scale no alpha png on disk). If you need the RGB decal but not the alpha decal, there is no VRAM savings (the alpha channel is included in VRAM no matter what) so you might as well use the alpha channel for something.

Decal Application

Decals are applied by specifying a DECAL command in the text file for the art asset. (Decals are allowed in any file that uses the Standard Shader, including .ter, .pol, and .net files.) The DECAL command specifies the texture to use and the mixing parameters that control application.

Examples

The following examples are based on some of Albert's and Sergio's terrain textures, and attempt to show some of the possible ways decals might be applied. They are not intended to be perfect, and make use of decal textures that Alex happened to have lying around! The examples also include some suggestions for normal map usage. For the purpose of these examples, the terrain files were applied to flat, horizontal mesh – mainly for convenience!

Example 1

The first example is based on Albert's rock_cld_hill_d.png albedo (as far as we're concerned, "albedo" is just another name for "daylight texture").

Here is a small section of it, in it's basic form:

This is the basic terrain file:

```
A

800

TERRAIN


BASE_TEX   ..textures/dev/rock_cld_hill_d.png

BORDER_TEX  ..textures/border/apt.png


PROJECTED  1500  1500

NO_ALPHA

COMPOSITE_BORDERS
```

First, we will use the DECAL command to apply a grayscale detail texture. DECAL is the simplest form of X-Plane decal, and has the following characteristics:

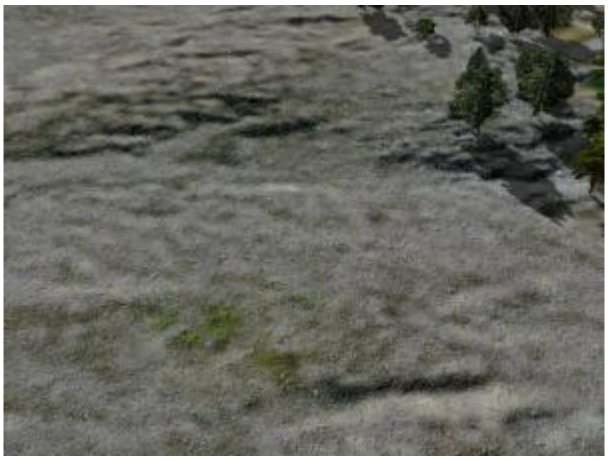
- It can only every apply a greyscale decal texture. X-Plane greyscale decals are applied using a blending mode similar to Photoshop's "hard light" mode.
- The decal is applied evenly across the entire area of the albedo.

DECAL has just two parameters: a scale ratio, and the texture file name.

DECAL 25.0 ..textures/soil/DECAL_stony_dirt.png

Adding this line to the file means that we will apply DECAL_stony_dirt.png at 25.0 x the frequency of the albedo, in other words, it will repeat every 60 metres. (Note: the decal texture does NOT need to have the prefix DECAL – I just happen to use that convention).

This is the result:



In some cases, a simple, homogenous greyscale decal like this will be enough. In our example, however, the albedo contains patches of vegetation. It would be nice to apply a different decal to the vegetation, instead of DECAL_stony_dirt.png. To do this we can use either DECAL_KEYED or DECAL_PARAMS, the other two types of X-Plane decal. There are two ideas we need to understand first:

1. Although we will be using two separate decal images, they will both be squeezed into a single texture file! By using a combination of one RGB decal and one greyscale decal, we can save VRAM by putting them both into a single RGBA file. The greyscale image goes into the alpha channel. This means, of course, that only one of our two decals will be in colour, but in practice, there are many cases where greyscale can be very effective. Both DECAL_KEYED and DECAL_PARAMS reference only a single texture *file*, therefore, even though they let us use *two* decal *images*. There is a restriction, however: both images are applied at the same *scale*. In other words, they will both cover exactly the same area of terrain.

2. We control exactly which decal gets applied to which areas of the albedo by using a ***key***. A key is a variable which tells the shader how strongly to apply a decal to a given pixel of the albedo texture. DECAL_KEYED uses a single key to control both the RGB and the greyscale decals, while DECAL_PARAMS uses two keys, one for each decal. Each key has four controls: R, G, B, and A. By tuning these controls, we can apply a decal more aggressively to some pixels than others.

Let's continue with our example by removing the simple greyscale DECAL, and replacing it with a DECAL_PARAMS. We will be using an RGBA decal texture which contains an RGB "vegetation" decal, and a greyscale "stony_dirt" decal – in fact, the greyscale decal is exactly the same image as the one we used before, the only difference is that this time it lives in the alpha channel of an RGBA texture instead of being a discrete file of its own:

```
A

800

TERRAIN


BASE_TEX   ..textures/dev/rock_cld_hill_d.png

BORDER_TEX  ..textures/border/apt.png

PROJECTED  1500  1500

NO_ALPHA

COMPOSITE_BORDERS


DECAL_PARAMS  25.0  0.5      -3.5  0.0  -3.5  3.0     2.0  -2.5  2.0  0.0
../textures/loveg/RGBA_DECAL_shrub_dirt_sdry3.png
```

What are all these parameters?

- The first parameter is the scale ratio, relative to the albedo (BASE_TEX), exactly the same as it is in the DECAL command.
- The second parameter is the "dither ratio". It controls the relative influence of the albedo and the decal on composite border dithering.
- The next four parameters are the controls for the RGB decal's key.
- The next four parameters are the controls for the greyscale decal's key.
- The last parameter is, of course, the decal texture file name.

How do the parameters get these values?

We set the scale ratio as before, at 25:1, so that the decals both repeat every 60 metres.

We set the dither ratio to 0.5. This means that the albedo and the decals will have roughly the same influence on the shape of the rendered borders. Higher values favour the decal. This means that if, like in this example, the decal is of higher frequency that the albedo, increasing the value of this parameter will make borders more "finely-grained".

This is where it gets a bit more complicated!

Looking at the albedo texture, we observe that the rocky parts are mainly grey and relatively light in colour (ignoring baked shadows for now). On the other hand, the vegetation is mainly green and rather darker than the rock. (This statement is clearly an approximation, but is accurate enough for our purposes.)

Therefore, by tuning the RGB decal's key to respond to darker, predominantly green colours, the areas of vegetation will receive more of our RGB decal, which is conveniently a sort of "shrubby" image.

In a similar fashion, by tuning the greyscale decal's key to brighter, grey tones in the albedo. those areas will receive more of the greyscale decal – our "stony dirt" image from before.

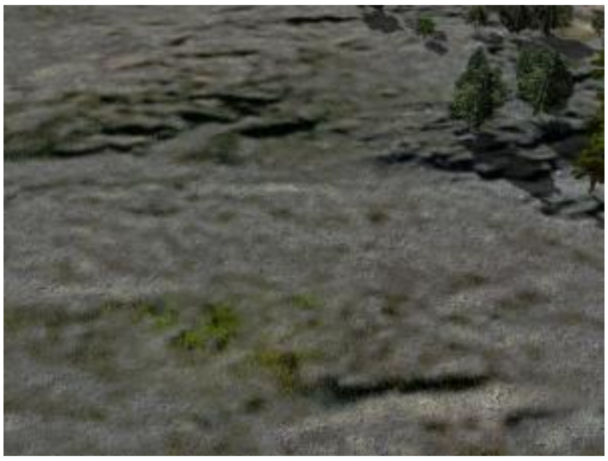
The actual values for the two keys are usually found by experiment, and do not follow any particular rules or equations. Experimentation is probably the best way of getting a feel for the controls, but this is how I arrived at the values in our example:

The second key (the one controlling the greyscale decal) is a bit more obvious that the first one, so we'll start with that. We want the rocky decal to be applied only to bright areas, so we set high values for R (=2.0), and B (=2.0). However, we *don't* want it to respond to green (we don't want rocks on our vegetation) so we set a *low* value for G (=-2.5). As mentioned at the top of this page, the exact function of control A depends on what other shader tricks we're using in the .ter file. In this case, it is simply a constant value which gets added to the R, G and B values. Basically, the greater the value of A, the more of that decal we will get. So in the case of our greyscale decal key, we set A to zero, because we *only* want to see the decal on *bright* colours. If we were to set a higher value for A, we would see the decal regardless of the colour of the albedo!

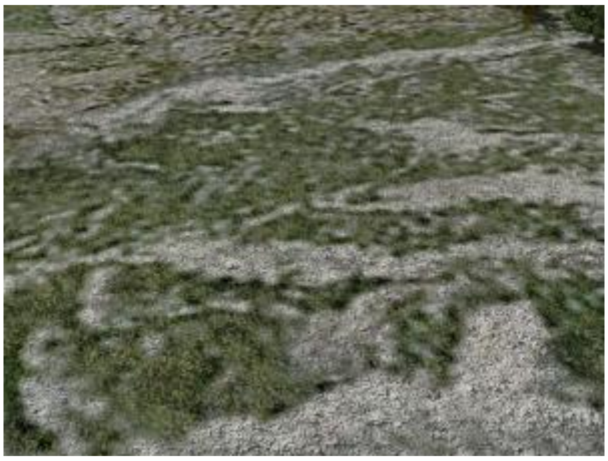
Now let's look at the first key, the one which controls our RGB "shrub" decal. In contrast to the greyscale decal, we want to see the RGB decal on both light and dark parts of the albedo – provided they are greenish. Therefore, we start by setting a high value for A. Doing this will draw shrubs over everything, so we must now tune out those areas we want to be shrub-free, by reducing the values of both R and B to -3.5.

The reason why some of these values are greater than 1.0 is that in order to get a reasonably sharp transition between regions of vegetation and regions of rock, we have to exaggerate the effect of the tuning controls. The actual difference in tone between a “veg” pixel and a “rock” pixel is generally pretty small, so we need to crank up the controls in order to get some “grip” on the albedo. The final key values are clamped between 0.0 and 1.0, so “overdriving” the key values will not cause artefacts in actual visible textures.

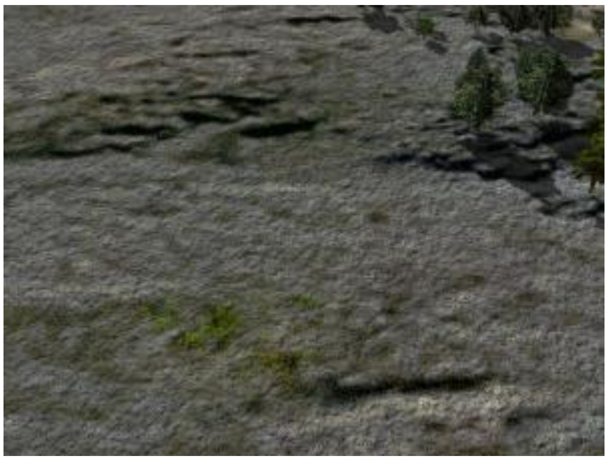
Here is the result:



Here is the same decal applied to a different albedo (sparse_cld_steep_D.png) with exactly the same parameter values:



To return to our first example, let’s now try adding a normal map. First, we’ll use a normal map that was originally designed to suggest rough, open grassland. It was made by hand-painting a greyscale height-map in Photoshop, using a soft-edged pen to create hummocks and clumps of grass, and basic noise to add “grit”. This was then baked into a normal map in Blender. Although it is doesn’t match the albedo, it still adds a nice sense of richness and depth to the terrain, and by adjusting the scale, quite a satisfying effect can be achieved:



Our .ter file now looks like this:

```
A

800

TERRAIN


BASE_TEX ..textures/dev/rock_cld_hill_d.png

BORDER_TEX ..textures/border/apt.png

PROJECTED 1500 1500


NO_ALPHA

COMPOSITE_BORDERS


DECAL_PARAMS 25.0 0.5 -3.5 0.0 -3.5 3.0 2.0 -2.5 2.0 0.0 ../textures/loveg/RGBA_DECAL_shrub_dirt_sdry3.png


NORMAL_TEX 6.0 ../textures/loveg/clumpy_turf5_NML.png
```

Changing the scale of the normal map (from 6.0 to 11.0) creates a slightly different effect:

General Observations Regarding Decals and Normal Maps in the context of terrain

The following points are based on the results of tests so far conducted by the author (Alex) and on some more general resource-management considerations.

Effective scaling

It seems that best results are obtained when the scale ratios between individual elements within a terrain definition is not “too large”. If the scale ratio between one element and the next-closest-in-scale exceeds a value in the region of 100 – 150, it is difficult to make them look integrated. Instead, they are likely to look like independent, unconnected layers, which is generally not desirable. An effective strategy for combining multiple elements in cases where the scale ratio between the largest and smallest is great seems to be one of adding an additional element of intermediate scale. For example, if a 10 x 10 m decal is applied directly to a 4000 x 4000 m albedo, the effect will probably not be good, since the scale ratio is very large: 400.0. However, adding a normal map of an intermediate size, say, 100 x 100 m, will help to bind the decal to the albedo, because the largest relative ratio between _successive_ elements is now 40.0:

Element	Size (m)	Scale Ratio	Relative Ratio
Albedo	4000 x 4000	-	
Normal map	100 x 100	40.0	40.0
Decal	10 x 10	400.0	10.0

It is this **relative** ratio which appears to be critical to the effectiveness of the terrain. The exact order or type of the elements seems less critical. For example, an alternative strategy in the above case might be to use a 100 x 100 m decal and a 10 x 10 m normal map.

Realism versus Impressionism

In my earlier experiments, I took a literal, realistic approach to design, choice, and scaling of decals – I attempted to match their size and content as closely as possible to the real world. However, results seem to suggest that, in many cases, a looser, more impressionistic approach can be as – or more – effective.

For example, it is possible to use a “park grass” decal to suggest much rougher terrain simply by reducing its scale ratio (thus increasing the area it covers). Similarly, a normal map designed to suggest a field of boulders might be used at a smaller size to represent fine scree.

Arguably, a truly accurate effect is beyond the scope of our system, and would require a disproportionate allocation of resources to make it work. However, I’m of the opinion that by adopting a more impressionistic approach, really good results can nevertheless be achieved, without obvious compromise of realism. In the previous section’s Example 1, a greyscale decal representing loose, stony ground is applied to a sheer rock albedo. Even though it is not strictly speaking realistic, it nonetheless adds interest and richness to the terrain. That particular example is probably taking this principle a bit *too* far – a more appropriate decal would make for a more realistic effect – but in terms of our visual priorities, it is suggested as a potentially acceptable method by which to add detail to our terrain reasonably quickly.

DSF File Format Specification

Last updated October 3, 2019

Revision History:

2/12/12 Updated for X-Plane 10

6/ 2/10 Updated for Wiki

11/12/06 Fixed incorrect size of counts for triangle primitives

5/08/05 Update for object prioritization

1/19/04 Initial Draft

DSF Overview

The Distribution Scenery Format (DSF) is the file format for scenery in X-Plane 8 and 9. A DSF file describes the appearance and some physical properties of a 1×1 degree section of the earth (or another planet).

This section describes the high level concepts of DSF. The DSF Specification section describes the file format precisely on a byte-by-byte basis.

This specification mentions cases where X-Plane deviates from the abstract DSF specification, or imposes further limitations.

DSF as Part of the Scenery System

Like the X-Plane 7 scenery system, X-Plane 8 and 9 scenery comes in packages consisting of a series of DSF files covering 1×1 degree surfaces, as well as a number of bitmap files and other helper text files such as objects.

Collectively the bitmap and text files that are referenced by DSF files are known as **graphic resources**.

DSF (Distribution Scenery Format) files contain compressed scenery data optimized for X-Plane.

Projection and Coordinates

DSF files represent locations on the Earth via degrees of latitude and longitude horizontally and meters relative to mean sea level vertically.

The Earth is approximated as the WGS84 ellipsoid.

All DSF coordinates are stored as sets of fixed point integers that are scaled by 32-bit floating point numbers. This math is done in 64-bit floating point precision. Points are translated to a local Cartesian coordinate system before geometry is constructed. This means that a straight line between two points in a DSF file will be straight in 3-d space, not in the geographic coordinate space of the DSF file (which is curved).

Properties

The metadata section of a DSF file contains a number of properties, consisting of a string name and a string value. Property names are hierarchical, with the '/' character reserved for hierarchy specification.

Compression and File Hashing

DSF files are not internally compressed. Client applications may choose to support reading and writing of compressed DSF files, and can distinguish between a compressed vs. uncompressed DSF file by the cookie at the beginning of the file – XPLANE vs. PK or 7Z for example.

X-PLANE NOTE: X-Plane 9 does not support reading compressed DSF files. X-Plane 10 supports uncompressed and 7Z-compressed DSF files, with the global scenery distributed in 7Z-compressed format.

DSF files contain a 128-bit MD5 hash at the end of the file describing the previous contents of the file. This allows X-Plane or other programs to quickly detect whether a DSF file has been modified and rebuild caches as necessary.

Compatibility Goals

DSF files are meant to be backward compatible but not forward compatible. New features are added via new atoms, new commands, and extensions to the definition file formats. New programs can identify different versions of DSF, and generally a program that can read the current DSF version will be able to read all older DSF versions without modification. Older programs will not be able to read newer versions of DSF.

Note: Conceptually forward compatibility doesn't make sense for scenery files. Consider a new scenery file that uses a new feature to represent a section of the world. If the old reader skips the new data then that part of the world will simply be missing or partially rendered. There is a data consistency problem with forward compatibility.

Definition Files and Building Blocks

A DSF file is essentially a 3-d model of a section of the earth, consisting of textured triangles and other polygons, with associated physical properties. However, specifying the entire Earth as triangles would make DSF files prohibitively large. Instead, DSF provides four types of building blocks from which models are built:

1. Mesh Patches. The surface of the earth is created via a series of 3-d triangles. The appearance of these triangles is defined by a terrain type

2. Objects. A 3-d model can be placed on the surface of the earth (as specified by the mesh) and rotated.
3. Polygons. A polygon can be extruded to form a building or filled to form a forest.
4. Vectors. A vector network can be extruded to form a transportation network, etc.
5. Raster Data. Starting with X-Plane 10, individual raster layers (2-d arrays of numeric values) can be saved directly into a DSF

For each type of building block, a definition file specifies how that building block will be rendered. Definition files are placed in the scenery package with the DSF files.

Mesh Patches

The DSF format does not dictate a particular configuration for the terrain mesh. Instead the mesh is a loose set of triangles that X-Plane indexes in memory. The mesh is divided into patches, or clusters of nearby triangles with the same terrain type. Each patch in turn may have multiple levels of detail (LOD), or versions of the terrain appropriate for different distances.

Mesh patches use a terrain type file to define their texture. The terrain type defines the texture(s) to be used for the texture, and may also optionally provide resolution information for projecting the texture onto the triangles or other texture processing information.

Mesh triangles may be overlay triangles (implying that they are drawn over existing mesh, with appropriate measures taken to prevent Z-buffer thrash), and they may be physical triangles, meaning they are used to calculate collisions and landings. The physical surface properties of the hard triangle come from the terrain type file.

Mesh primitives replace the 200×150 grid from .env files. Terrain types replace custom custom textures and land uses.

Objects

Objects can be placed on the surface of the scenery (as defined by the mesh) and rotated around a vertical axis. Object files (OBJ) are used to define the appearance of the object.

DSF Objects are the same as custom objects in an .ENV file. Default objects are no longer supported; however, X-Plane comes with a library of custom objects that may be used by any DSF file.

Polygons

Polygons are optionally nested, optionally closed vector paths along the surface of the earth (as defined by the mesh). Polygons are instantiated by polygon definition files.

Each polygon has one parameter value. Currently two types of polygon definition files are supported:

- Facade definition files extrude 3-d buildings along the polygonal path. The definition file describes the way the building is textured and the parameter defines the height of the building in meters.
- Forest definition files fill the area within the polygon with 3-d objects (e.g. trees). The definition file describes the type and fill pattern of tree placement, and the parameter controls density.

Polygons provide additional ways to create buildings with specific varying heights and also provide an option for creating 3-d vegetation.

Vectors

Vector primitives create pathways along and above the surface of the earth along vectors. Vector data consist of a series of junctions and a series of segments connecting the junctions. Each vector segment may be a straight line or bezier curve.

The vector definition files describe the appearance of vectors, both on their own and when they connect at junctions. Network primitives replace the various pre-defined vectors found in X-Plane 7's .env files.

In a DSF file, each junction is numbered, starting at the index 1 and increasing consecutively within the DSF file. (The index zero is reserved.) These junction numbers allow X-Plane or another reader to rapidly reconstruct the network topology of the vector structures, and clearly define when two vector chains intersect.

Raster Layers

X-Plane 10 allows for raster layers to be directly encoded into a DSF. Each raster layer has a name in the definition atoms, and meta data defining its pixel format and dimensions. How the raster data is interpreted is defined by X-Plane.

Building Block Definition Extensibility

Building Block Definition files are text files. The individual definition files can be extended to allow for more powerful representation of the various building block types.

The parameter for polygons can be interpretted as necessary by the polygon definition files.

The terrain mesh can be extended to have additional per-vertex information, which can then be used by terrain type definition files. For example, in the default DSF scenery, some vertices have a ratio of plant coverage on each vertex, and this is used to blend textures.

Chunky File Format

The outer level of DSF files is **chunky** (also sometimes called atomic), meaning each section of the file has a uniform header describing the ID of the file section and the length. This allows programs to find the data they require without parsing unknown or unimportant parts of the file. It also allows programs to store extra data in a DSF that X-Plane will ignore.

Where possible, standard encodings are used for the various atoms in a DSF file, reducing the amount of code needed to parse or write the file.

Point Pools

The 2-d, 3-d, and other coordinates in the file are stored in point pools. A point pool is a collection of N-dimensional points that are scaled and offset in a uniform matter. Point pools are always stored as fixed point fractions that are interpreted over a range that is specified once for each data plane in the point pool.

In most cases there are special versions of the primitive commands that allow a primitive to be built along consecutive points in a pool. Point pools should be constructed to keep primitive vertices in order when vertices are used only for one primitive.

Definition Tables

The definitions for primitives are stored in tables; their order defines index numbers used to reference them. The tables store the file name for the text file defining the primitive.

Atom Extensibility

New atoms may be added to DSF in future versions via new atom IDs. Atom IDs consisting entirely of capital letters and numbers are reserved for the DSF specification. Other atom IDs can be used for private data.

Command-Based File Format and State

While atoms define the sections of a DSF file, commands accomplish most of the work of defining the file’s contents. The command section consists of a series of command opcodes, each of which instructs X-Plane to add a primitive to the current scenery, or change internal state during the DSF file parsing. The various state maintained while parsing a DSF file is reset at the beginning of the DSF file and maintained continuously through all commands; this state serves primarily as a technique to compress similar data. State must be specified before utilized in a DSF file.

Current Definition and Current Road Subtype

Each building block in a DSF file is instantiated using a definition; the current definition is maintained state that indicates what definition the next command will used. The set definition commands change this state. One current definition index number is maintained for all primitives. (This means that if an object index 0 follows a mesh of terrain type 0, no set definition is needed.)

Selected Pool

Before a coordinate from a pool can be used, the pool must be selected. Selecting the pool decides which of multiple geometry pools will be used. Only one pool can be selected at a time.

Junction Offset

Unlike other primitives, vector networks use a 32-bit point pool. A 32-bit point pool is precise enough to allow all data to be encoded in one pool. Since this pool might contain more than 65535 points, a junction offset is maintained and added to indices for most junction commands.

Note: this feature exists primarily for historical reasons. However it is used by the default scenery renderer's DSF writing library because it simplfies the process of organizing vector data to not have to worry about point pools.

Terrain Patch LOD and Flags

Each terrain patch has some properties associated with it; these properties are considered state. When a new patch is created, if these properties are not changed, the new patch has the same properties as the old one.

Command Extensibility

The addition of new DSF commands will break compatibility; this represents a change to the DSF spec.

Building-Block Related Implementation Issues

Point pools may have any number of data planes; the interpretation of those planes depends on the building block. All pools start with longitude and latitude and have at least two planes. Extra planes are ignored (and allowed for future expansion). Having too few planes is an error.

Point Primitive Rotation Plane

Point primitives require at least three planes: a longitude, a latitude, and a rotation. The rotation is performed around the gravity vector, a vector pointing straight down toward the center of the earth. The object sits on the surface defined by the mesh. If the terrain is sloped, the object may end up partially under ground; negative object Y coordinates allow for underground geometry.

Rotations are stored in degrees.

Polygon Parameters and Direction

A polygon primitive defines a polygonal path or set of paths. When a polygon primitive actually contains nested polygons (to form holes), each polygon is known as a **winding**. A few restrictions on polygons and windings:

- Polygons may be concave or complex, but no edges can ever cross (except for ends touching) across any windings.
- If a winding is an outer boundary of a polygon (meaning the interior of the winding is part of the polygon, it must be counterclockwise. If a winding is an inner boundary (meaning inside the winding is outside the polygon, e.g. a hole), it must be clockwise.
- Each winding must be after every winding that contains it in the ordering of the winding list.

Every point in the polygon sits on the surface of the mesh.

Cross-Pool Vector Linkage

The vector structures for DSF are more complex than those of ENV because they provide network topology. The building blocks of the network are junctions and chains.

- A vector segment is a single straight line or bezier curve between two points consisting all of the same type of vector.
- No segments can touch each other except at their ends. (However, if segments cross above and below each other in 3-d space, this is legal.)
- A chain is a series of connected vector segments such that the entire chain is of the same type, and each segment connects to exactly one other segment except at the ends of the chain.
- The ends of the segments within the chain are shape points. They define the shape of a chain but do not represent a junction between segments.
- The two ends of the chain are junctions.

In a DSF file, each junction is assigned a unique 32-bit ID, starting at 1. Two chains form a junction if their end nodes have the same unique 32-bit ID. Only junction ID is used as a test to see if two chains meet at a junction.

In the point pool for vectors, the 4th data plane (after longitude, latitude and elevation) is used for junction IDs. Zero is used to indicate that the point is a shape point and not a junction.

There may be a complete chain whose geometry indices are not within 65536 of each other; in this case a special segment command is provided that allows for arbitrary 32-bit indices to be specified.

Vector Type Indices

Unlike other definition files, a single vector definition file contains multiple subtypes (also referenced by index). This allows a single definition file to provide specialized images for the crossing of vector subtypes that are both from the same definition file.

Mesh State

The scenery mesh is divided into patches. A patch is simply a collection of nearby triangles that have all of their attributes in common. Those attributes are:

- A level of detail range in meters indicating the range the patch can be seen at.
- The definition number defining the patches terrain type.
- Whether the patch needs to be treated in the physics model for X-Plane.
- Whether the patch is an overlay onto another patch (and must be treated for z-buffer thrash).

A patch can either be textured by providing per-vertex texture coordinates or by using a projection equation. The projection equation is used for the texture but not necessarily the mask.

The precise requirements and interpretation of planar geo data for a mesh patch are a function of both the mesh state for that patch and the terrain definition file itself. A precise description of X-Plane's mesh behavior is documented in the .TER terrain type file format specification.

Requiring Objects and Facades to Be Drawn

Starting with X-Plane 8.10, if X-Plane sees the properties sim/require_object or sim/require_facade in a DSF file, it will use the value of this property to force certain objects to be drawn even at lower settings.

The value associated with these properties is a string containing two numbers separated by a slash. The first number is the X-Plane rendering level (0-6) at which to require the object, and the second the index number of the first object or facade definition to be affected. The property may be included multiple times. For example:

Property	Value
sim/require_object	4/70
sim/require_object	2/74

In this example, all objects whose definition index is 70 or higher must be drawn when the object detail setting is 4 or higher. All objects with an index of 74 or higher must be drawn if the object detail setting is 2 or higher. (Note that the union of all rules is taken: so even though object definition index 75 is covered by both rules, the lower setting of 2 applies—these objects will be visible at rendering settings of 2 or lower.)

To take advantage of this, a DSF may need to be organized so that "high priority" objects have higher (later) indexes in the file.

DSF Specification

Data Encoding

Endian

DFS files are binary little-endian files. Floating point numbers are stored in IEEE 32-bit or 64-bit format.

DSF File Structure

A DFS file is atomic, which is also sometimes called chunky. Each 'atom' of data has a size and ID at the beginning.

Header

The header contains an 8-byte unique ASCII cookie identifying this as an x-plane DSF file. The cookie will be the ASCII characters 'XPLNEDSF'. This 8-byte unique cookie is followed by a 32-bit integer master file format version. The current version number is 1.

Atoms

Following the 12-byte header is a variable number of atoms. Each atom consists of a 32-bit atom ID followed by a 32-bit unsigned byte count for the size of the atom, including this 8-byte header. The contents of the atom depend on the atom's ID. Atom IDs may be repeated, so the order of the atoms within the file may be significant. The DSF file may also place requirements on the order of the atoms within the files.

Endian Note: since atom IDs are 32-bit integers (and not arrays of 4 characters), they are subject to endian-swapping. For example, viewing a DSF file in a windows hex editor the GEOD atom would read DOEG since 'GEOD' is really a 32-bit character constant that must be swapped.

The atom section is variably sized. The atom section ends 16-bytes before the end of the file and can be thought to be the entire file with the header and footer removed.

The format of the contents of an atom depend on the ID of the atom; different atoms may have different formats. Atoms may contain subatoms; usually the entire contents of the atom will be atoms in this case, but this is dependent on the details of the specific atoms in question.

Footer

The footer of a DSF file is a 128-bit MD5 signature of the previous contents (not including this signature). This forms a unique ID for this file that allows clients to detect revisions in content.

Common Atom Formats

Where possible, the same format is used for atoms that contain similar data; this is to reduce the amount of code necessary to read or write DSF files. This section describes some of the typical encodings of atoms.

Atom of Atoms

This format encodes a number of atoms within a single atom, forming a hierarchy. The entire content space of the super-atom contains sub-atoms end-to-end. The number of sub-atoms may be found by traversing the atom until the total number of bytes in the super-atom have been processed.

String Table Atom

A string table atom contains a series of null-terminated C-strings packed with one null byte between each one. No padding/byte alignment bytes are necessary. The number of strings is determined by traversing the entire atom. A null character on the final string **is** necessary for the final string.

Note: this is necessary to allow an empty string as the last string. It also allows the strings to be used in-place in a memory mapped file.

The orders of the strings in a string table is significant; the string table atom forms a natural numbering starting at 0 for each string. Empty strings may be encoded via a single null character.

Planar Numeric Atom

A planar numeric atom contains one or more arrays of numbers in planar format. For example, a table of 100 latitude/longitude pairs would contain all of the latitudes first and then all of the longitudes. The numbers may be either fixed point, integer or floating point. The format of the numbers and number of planes are determined by the specific atom type (e.g. these are flexible within a planar numeric atom). This metadata is **not** written into the file itself; the reader must understand what kind of planar data to expect based on the atom ID.

Note: the motivation for a planar structure is to keep similar-typed data together. In a lat/lon array, the latitude will be similar in range, as will the longitude. Metadata is not written to the file itself because the client must be able to interpret the data it reads, and therefore knows enough about a given planar numeric atom to provide the metadata as a key to reading it. It is unlikely that the numeric needs of low level atoms will change without the file format itself changing massively.

Both the number of data tuples and the number of planes are written into the planar numeric atom.

A planar numeric atom's data plane may be compressed in two ways (which are not mutually exclusive) for four possible encoding forms:

1. Raw data. Each number follows sequentially.
2. Differenced. Each number is subtracted from the previous and then written to the file. Wrapping rules apply to integer number types.
3. Run-length-encoded. A byte count and flag indicate the number of similar numbers, followed by the number, or the number of individual numbers, followed by the numbers.
4. The array may be differenced first and then run-length encoded.

Note: the goal of differencing is to decrease the range of values that appear when data is always closely spaced. The goal of run-length-encoding is to get a big savings for constant data.

The number of items in each plane of the array must be the same, and each item must have the same number of planes. The format of the atom is:

- A 32-bit item count for the number of items in the array.
- An 8-bit plane count for the number of planes in the atom.
- For each plane, a 1-byte enumeration for the encoding type. These are: raw = 0, differenced = 1, RLE = 2, RLE differenced = 3, followed by:
- Variable length encoded data depending on the compression type and numeric format.

The run-length byte for run-length encoding is an unsigned 8-bit character. The high-bit indicates a repeat of up to 127 of the following number. The high bit cleared indicates up to 127 non-repeating elements.

DSF is little endian, but differencing is done in the machine's endian format (e.g. differencing is done logically on the data, not on the file). RLE is a bit-wise operation and is not endian-sensitive.

DSF Atoms

The various file sections of a DSF file are each contained within separate atoms, with their own IDs. The header ID is listed on the header of each section of this specification.

Header Atom ('HEAD')

This is an atom of atoms containing information about the DSF file. The HEAD atom is an atom-of-atoms in the top level of the file and currently contains one subatom the PROP atom.

Properties Atom ('PROP')

The properties atom is a string table atom with an even number of strings; each consecutive pair of strings represents a property name and a property value. This allows for arbitrary metadata to be placed inside a DSF file.

Properties starting with sim/ are reserved for public X-Plane use. Properties starting with laminar/ are reserved for X-Plane private use. All other prefixes may be used for private data. When storing private data in the DSF properties metadata section, prefix the property with your company or organization name to prevent conflicts.

The following properties are currently defined:

Property	Default If Missing	Definition
sim/west	(Required)	The western edge of the DSF file in degrees longitude.
sim/east	(Required)	The eastern edge of the DSF file in degrees longitude.
sim/south	(Required)	The northern edge of the DSF file in degrees latitude.
sim/north	(Required)	The southern edge of the DSF file in degrees latitude.
sim/planet	earth	The planet this DSF belongs to, one of 'earth' or 'mars'.
sim/creation_agent	(Blank)	The name of the program that created the DSF file if know
sim/author	(Blank)	The name of the author of the DSF file if known.
sim/require_object	N/A	Requirements for displaying objects (see below).
sim/require_facade	N/A	Requirements for displaying facades (see below).

The sim/require_object and sim/require_facade properties specify that objects and facade whose definition index is greater than or equal to a certain number must be drawn by the sim. Normally the sim may draw only a fraction of the objects or facades in a DSF, based on the user’s rendering settings. By including these properties, you can force X-Plane to always draw objects.

Definitions Atom (‘DEFN’)

The definitions atom contains a series of subatoms that define the various ‘definitions’ used within the DSF file. DSF files first reference a few common definitions and then use them in a larger number of instances. Each definition comes from an external file, allowing definitions to be shared among DSF files or even between scenery packages. (This allows custom scenery packages to use a variety of x-plane-default definitions.) The various definition formats are described in other specifications.

The definition atom is an atom-of-atoms syntactically. All definition sub atoms define a series of partial file paths by being string table atoms. The forward slash (‘/’) should be used as the directory separator. All definitions are referred to by zero-based index in the rest of the file. A maximum of 65536 entries are allowed in any one table. The extension for the file path is always included.

The following four atoms sit inside the definitions atom: the TERT, OBJT, POLY and NETW atoms.

Terrain Types Atom (‘TERT’)

The terrain types atom lists a number of external .ter terrain-definition files that define the various terrains used in the DSF file. Terrain-definition files describe the set of textures to be used for the terrain (dependent on season) as well as other metadata (for example, is this terrain hard, bumpy, etc.).

You may also use .png or .bmp files directly to specify terrain types. See the .ter file specification for info on both the .ter file format and on using PNG and BMP files here.

Objects Atom (‘OBJT’)

The objects atom lists a number of external .obj object files that may be ‘placed’ repeatedly in the DSF file. With DSF files there are no default object types; radio stacks, sky scrapers, etc. are all created using either Object or Prototype (see below) files.

The Polygons Atom (‘POLY’)

The prototype atom lists a number of external polygon definition files that may be usde within the DSF file.

Polygon definitions can be either facades or forests; X-Plane determines the type of polygon definition from the filename extension.

Vector Network Atom (‘NETW’)

The network atom lists a number of external .net network definition files that may be used within the DSF file. While individual objects are placed separately in a file, roads and other ‘networks’ intersect each other. A network definition file describes the appearance and geometry not only of multiple different types of roads (or other segments), but how to build blended intersections of those segments.

X-PLANE NOTE: X-Plane 8 and 9 can only accept one network definition per DSF file; use vector subtypes to define multiple road types, etc.

Raster Definition Atom (‘DEMN’)

New to X-Plane 10: The raster definition atom defines the names for each raster layer contained in the DSF. The order of raster data in the subsequent atoms matches the order of names in the raster definition atom.

Geodata Atom (‘GEOD’)

The geodata atom defines all of the coordinates for all geometry in the DSF file. Coordinates are separated from instantiations of definitions to encourage recycling and reduce file size.

The Geodata atom is an atom-of-atoms at the root of the DSF file. The GEOD atom contains zero or more POOL, SCAL, PO32 and SC32 atoms.

16-bit Coordinate Pool atom (‘POOL’)

This atom is a planar numeric atom with a variable number of planes and 16-bit unsigned int data, establishing a coordinate pool. Multiple pool atoms sit inside the Geodata atom, so the index number of this coordinate pool is based on its order within the geodata atom, starting at 0. Points are stored in sixteen bit unsigned short format, representing values from [0-65536).

16-bit Scaling Range Atoms (‘SCAL’)

For each pool atom there is also a scaling range atom in the GeoData atom, telling how to process the point pools. Each scaling atom contains an array of 32-bit floating point numbers. There are two floats for each plane in the corresponding point pool, the first being a scaling multiplier and the second being an offset to be added to the points. These values are applied in double-precision.

POOL and SCAL atoms are applied based on their order within the file, e.g. the 5th POOL atom within the GEOD atom is scaled by the 5th SCAL atom in the GEOD atom. There must be an equal number of POOL and SCAL atoms. The data planes in the pool atom correspond to the values in the scal atom, so if there are *n* planes in a POOL atom, its corresponding SCAL atom must have *2n* 32-bit floats.

32-bit Coordinate Pool Atom (‘PO32’)

The 32-bit point pool atom is the same as the 16-bit point pool atom except that each data element is a 32-bit rather than 16-bit unsigned int. 32-bit point pool atoms are used for vectors; all other building blocks use 16-bit point pools.

32-bit Scaling Range Atom (‘SC32’)

The 32-bit scaling range atom is the same as the 16-bit scaling range atom except that it is appleid to 32-bit point pool atoms. In other words, the 3rd SC32 atom scales the 3rd PO32 atom. The atom is still formed of 32-bit floats, but like the 16-bit scaling atom, the conversion is done in double-precision floating point.

Raster Data Atom (‘DEMS’)

New to X-Plane 10: The raster data atom is an atom of atoms containing the meta data and raw data for each raster layer in the DSF.

The raster data atom contains one raster layer information (‘DEMI’) and one raster layer data atom (‘DEMD’) for each raster layer. The order of information and data atoms must match with the raster definition atoms. This is how names, meta data, and raw data are matched in the DSF.

Raster Layer Information Atom (‘DEMI’)

The raster layer information atom contains the structure information for one raster layer. The atom is a record of DEM information, encoded as follows:

Field	Encoding	Description
Version	uint8	Version of DEM record; set to 1
Bytes Per Pixel	uint8	The number of bytes for each pixel. Should be 1,2, or 4 depending on encod
Flags	uint16	Encoding Flags – see below
Width	uint32	Width of the DEM east-west in pixels
Height	uint32	Height of the DEM north-south in pixels
Scale	float32	Scaling factor to apply to DEM pixels post-load
Offset	float32	Offset factor to apply to DEM pixels post-load

Each final DEM pixel is multiplied by scale and then offset is added.

The flags field defines a number of other DEM properties:

- The low 2 bits tell the number type for the DEM data:
    - 0 = floating point (bytes per pixel must be 4)
    - 1 = signed integer (bytes per pixel must be 1, 2 or 4)

    - 2 = unsigned integer (bytes per pixel must be 1, 2 or 4)
- A flag value of 4 (bit 3) defines the data as post-centric, as opposed to area-centric.
    - In post-centric data, the pixel values at the edges of the DEM exactly lie on the geometric boundary of the DSF.
    - In point-centric data, the outer edge of the pixel rectangles lie on the geometric boundary of the DSF.

Raster Layer Data Atom (‘DEMD’)

The raster data atom contains the actual raw raster data, sitting directly in the atom’s payload, one DEMD atom per layer. The information atom above tells how to inerpret this raw data.

Commands Atom (‘CMDS’)

The commands atom contains a list of commands used to actually instantiate the scenery file by applying prototypes, objects, etc. at the coordinates available in the geodata atom.

Commands consist of a command ID and additional information in series. Command order is arbitrary but may be optimized for file size by the file writer. Command order does **not** affect display order when X-Plane renders scenery; display order is affected by internal factors in the rendering engine.

The number of bytes used by a command is known through its type; unknown commands cannot be skipped. The commands are finished when the last command in the atom is parsed. All command IDs are 8-bit. Command ID 255 is reserved for future expansion. The format of the data following the command is based on the command ID.

All commands that include a range of indices list the first index, and one more than the last index. All commands reference 16-bit point pools except for vector commands, which reference 32-bit point pools.

DSF Commands

Each command is listed in order of ID; any data that must follow the command is listed below. Data items listed are:

uint8	Unsigned 8-bit integer.
sint8	Signed 8-bit integer.
uint16	Unsigned 16-bit integer.
sint16	Signed 16-bit integer.

uint32	Unsigned 32-bit integer.
sint32	Signed 32-bit integer.
f32	32-bit IEEE floating point.
f64	64-bit IEEE floating point.

State Selection Commands

These commands change the internal state of the DSF reader, affecting the results of subsequent commands.

COORDINATE POOL SELECT (ID=1)

The coordinate pool select command changes the current coordinate pool and establishes the longitude and latitude bounds that the first two fields are interpreted within.

uint16 zero based pool index

JUNCTION OFFSET SELECT (ID=2)

The junction offset select command specifies a 32-bit number that is added to all indices when referencing coordinates for vectors. This allows the use of a 16-bit vector command for vectors whose indices are greater than 65535.

uint32 zero based index offset

SET DEFINITION 8 (ID=3)

uint8 zero based definition index

SET DEFINITION 16 (ID=4)

This is the same as above, but with a 16-bit index.

uint16 zero based definition index

SET DEFINITION 32 (ID=5)

This is the same as above, but with a 32-bit index.

uint32 zero based definition index

SET ROAD SUBTYPE 8 (ID=6)

This command sets the road subtype for the next vector-segment.

uint8 zero based road subtype

Object Placement Commands

These commands place an object on the surface of the mesh. A point pool must be selected and must have at least 3 planes, which are treated as a longitude, latitude, and rotation in degrees.

OBJECT COMMAND (ID=7)

This command places a single object based on the current definition.

uint16 coordinate index

OBJECT RANGE COMMAND (ID=8)

This command places several objects based on the current definition, using all vertices within a range.

uint16 index of first objectuint16 index of last object+1

Network Commands

The network commands instantiate complete chains and junctions for a network. Networks are formed by instantiating complete chains. The coordinate pool for a network segment must have either four planes (longitude, latitude, elevation, junction ID) or seven planes (adding on longitude, latitude, and a shape point for shaping).

Junction IDs are one-based consecutive unique integers. Junction IDs simply indicate what junctions will link to each other; if two junctions have the same ID but different spatial locations (based on the first three planes after transform), this is an error.

The junction ID zero indicates a shape point, meaning a coordinate that changes the shape of a vector but is not a junction.

All network commands except for the network-chain-32 command add the junction offset to all indices.

NETWORK CHAINS (ID=9)

This command creates one or more complete chains, using all of the vertices that are specifically enumerated. Complete chains are started or ended based on the presence of a non-zero junction ID.

uint8 number of coordinates

N x uint16 coordinate indices

NETWORK CHAINS RANGE (ID=10)

This command creates one or more complete chains, using all of the vertices within the range specified. Complete chains are started or ended based on the presence of a non-zero junction ID.

uint16 first coordinate inde

uint16 index of last coordinate+1

NETWORK CHAIN 32 (ID=11)

This command creates one or more complete chains, but rather than using 16-bit indices and the junction offset, they use explicit 32-bit indices and no offset. Use this command to create a vector when the indices span a range of more than 65536.

uint8 number of coordinates

N xuint32 coordinate indices

Polygon Commands

The polygon commands instantiate polygon primitives on the surface of the mesh. The selected plane must have at least two planes, which are interpretted as longitude and latitude. A per-polygon 16-bit parameter is interpretted based on the polygon definition.

POLYGON (ID=12)

This command instantiates a single polygon.

uint16 parameter valueuint8 number of indices

N x uint16 coordinate indices

POLYGON RANGE (ID=13)

This command instantiates a polygon through a contiguous range of vertices.

uint16 parameter valueuint16 first index

uint16 last index+1

NESTED POLYGON (ID=14)

This command instantiates a series of polygons, each with a distinct winding.

uint16 parameter

uint8 number of polygon windings

(for each of N windings)

  uint8 number of indices

  M x uint16 coordinate indicies

NESTED POLYGON RANGE (ID=15)

This command instantiates a series of polygons with distinct windings using a list of contiguous ranges. Each index starts a winding except for the last, which is one past the end of the polygon's last point.

uint16 param value

uint8 number of indices (this polygon has N-1 windings).

N x uint16 indices

Mesh Commands

The mesh commands instantiate the terrain mesh as triangles. The mesh commands take planar data with at least 5 parameters, corresponding to to longitude, latitude, elevation, and a normal. Additional parameters are used to texture the patch based on the .ter terrain type definition file.

Mesh normals are defined as a pair of coordinates from -1.0 to 1.0 that represent the X and Z components of a normalized normal vector, where the positive X axis points east and the positive Z axis points south. The Y component is derived from the X and Z components by X-Plane.

There are three ways to specify triangles in a patch: triangles, strips and fans. Strips and fans are adjacent triangles that share common vertices. This lets you specify more triangles with less vertices, saving file size by up to a factor of 3.

For each type of triangle primitive (fan, strip, or individual triangles), there are three commands. One places triangles by a series of indices (up to 255 per command).

Another uses a range of consecutive indices, saving file size. A third takes points from multiple point pools by specifying the point pool individually.

Note: why is there a cross-pool triangle command? Mesh vertices must have the exact same coordinates or else the mesh will have cracking – visible OpenGL artifacts. There may be rounding error associated with reducing the coordinate to a 16-bit integer via the scaling factors. Therefore if a coordinate V is included in two different point pools (for use in two different triangles), the two triangles may not actually line up at coordinate V. To guarantee alignment, both triangles must reference coordinate V in the same point pool, which means that triangles may have to span point pools to form the entire mesh.

TERRAIN PATCH (ID=16)

This command indicates that a new terrain patch is being created. The patch will have the same LOD range and flags that the last created patch had.

no data follows this command.

TERRAIN PATCH FLAGS (ID=17)

This command indicates that a new terrain patch is being created. The patch will have the same LOD range as the last patch, but new flags. The flags are:

Bit Value	Indication
1	Physical – if set, this patch is used for collision detectoin. If cleared, the patch is drawn but not checked.
2	Overlay – if set, this patch is drawn over another patch. Z buffering precautions are taken. The p interpretation of this flag may depend on the terrain type.

uint8 new flag values

TERRAIN PATCH FLAGS AND LOD (ID=18)

This command indicates that a new terrain patch is being created. Besides specifying flags, a new LOD range in meters is also provided. Flags are the same as above.

uint8 flagsf

32 near LOD

f32 far LOD

PATCH TRIANGLE (ID=23)

This command creates one or more specific triangle for a terrain patch. Triangles must have clockwise rotation as seen from above for all triangle primitives.

uint8 coord count

N x uint16   coordinate indices

TRIANGLE PATCH CROSS-POOL (ID=24)

This command creates one triangle from multiple terrain pools. This is the same as the command above, except that a pool index is provided per vertex.

uint8 coord count

2N x uint16 pool ID/coord index pairs

PATCH TRIANGLE RANGE (ID=25)

This command creates a number of triangles based on the inclusive range of coordinate indices. The range must be a multiple of 3. Each set of 3 adjacent vertices is treated as a triangle.

uint16 first index

uint16 last index+1

PATCH TRIANGLE STRIP (ID=26)

This command creates a triangle strip. A triangle strip is a series of adjacent triangles that share two common vertices; for a series of points 1,2,3,4,5 as a triangle strip is equivalent to the triangles 123,243,345...

uint8 coordinate count

N x uin16 coordinate indices

PATCH TRIANGLE STRIP CROSS-POOL (ID=27)

This command creates a triangle strip, except the point pool is specified per vertex rather than referencing the current coordinate pool.

uint8 coordinate count

2N x uint16 poolID/coord index pairs.

PATCH TRIANGLE STRIP RANGE (ID=28)

This command creates a triangle strip for a series of consecutive coordinates.

uint16 index of first coordinate

uint16 index of last coordinate+1

PATCH TRIANGLE FAN (ID=29)

This command creates a triangle fan. A triangle fan is a series of adjacent triangles that share two common vertices; for a series of points 1,2,3,4,5 as a triangle fan is equivalent to the triangles 123,134, 145...

uint8 coordinate count

N x uin16 coordinate indices

PATCH TRIANGLE FAN CROSS-POOL (ID=30)

This command creates a triangle fan, except the point pool is specified per vertex rather than referencing the current coordinate pool.

uint8 coordinate count

2N x uint16 poolID/coord index pairs.

PATCH TRIANGLE FAN RANGE (ID=31)

This command creates a triangle fan for a series of consecutive coordinates.

uint16 index of first coordinate

uint16 index of last coordiante+1

Comment Commands

These commands allow arbitrary data to be embedded in a DSF file. The commands are defined by the size of the length field, allowing for larger or smaller comment blocks. The length field tells the length of the following comment data not including the length field itself.

COMMENT 8 (ID=32)

uint8 length

N x uint8 comment data

COMMENT 16 (ID=33)

This defines a comment of up to 65535 bytes.

N x uint8 comment data

COMMENT 32 (ID=34)

This defines a comment of up to 4294967295 bytes.

uint32 length

N x uint8 comment data

Draped Polygon (.pol) File Format Specification

Last updated March 2, 2015

Draped polygons (.pol) define how a DSF polygon should be turned into pavement or some other ground that is draped over the existing terrain mesh. One use is to make custom taxiways, but any texture could be used.

.pol files are used via bezier polygons. Multiple windings may be used to specify holes in the pavement. The polygons should have two coordinates (lon,lat) for line segments or four coordinates (lon,lat, control lon, control lat) for bezier curves. The parameter is used as a rotation of the texture in degrees.

Note that .pol file textures are not aligned to any particular coordinate system so they are only appropriate for repeating texture, not orthophotos.

[New in 860:] Starting with X-Plane 860, a polygon's texture coordinates may be specified in the DSF file. To do this:

Set the polygon instance's parameter to 65535 in the DSF.
Pass two (or four for bezier curve) ST coordinates at the end of each vertex that will be used as ST coordinates.
X-Plane will ignore the SCALE command.
The header of a .pol file is:

A

850

DRAPED_POLYGON

The rest of the file is made of commands, or # for a comment line. The commands are:

TEXTURE <texture name>

This defines the texture to use – the texture name is relative to the .pol file.

TEXTURE_LIT <texture name>

If present, this defines a night lighting overlay to use, again relative to the .pol file.

SCALE <horizontal scale> <vertical scale>

This defines how large one iteration of the texture is, in meters.

(860: if the polygon has specific ST coords in the DSF, the scale command is ignored.)

LAYER_GROUP <group name> <offset>

This specifie which layer group (z order) to draw the polygon in. The definition is the same as the ATTR_layer_group command in the OBJ8 spec.

SURFACE <surface type>

If present, this defines the type of hard surface that the polygon will create. Surface codes are the same as for ATTR_hard in the OBJ8 spec.

TEXTURE_NOWRWAP <texture name>

[New in 860:] This command is the same as the TEXTURE command, except the texture's edge blending is set to "clamped" as opposed to "wrapped". This will cause the texture to correctly tile when it is a subsection of an orthophoto, but not when the texture repeats like a land-use.

TEXTURE_LIT_NOWRWAP <texture name>

[New in 860:] This command is the same as the TEXURE_LIT command, except the texture's edge blending is set to "clamped" as opposed to "wrapped". This will cause the texture to correctly tile when it is a subsection of an orthophoto, but not when the texture repeats like a land-use.

NO_ALPHA

[NEW in 860:] This command tells X-Plane to ignore the alpha channel of the texture. Some textures may have special "noise" alpha channels that aren't useful for basic texturing; this command can be used to strip out the alpha data in the sim, allowing the texture to be used for multiple purposes.

LOAD_CENTER

X-PLANE 920: This command estabishes that this texture will be used at a certain location, specified by the lat/lon taken to be the texture's center. By also specifying theapproximate terrain size when placed (in meters) and textures size (in pixels), X-Plane will load the texture with variable resolution based on the general distance from the user to the texture. As the user flies, X-plane will periodically reload the texture.

LOAD_CENTER affects loading of the base texture and night texture if it exists. A texture that uses LOAD_CENTER should only be referenced once by one art resource per DSF tile. For optimal performance, the texture should be in DDS format, so that reloads at lower resolution are fast.

LOAD_CENTER

LOAD_CENTER 42.70321 -72.34234 4000 1024

Example file:

A

```
850
DRAPED_POLYGON

TEXTURE concrete.png
SCALE 25 25
LAYER_GROUP taxiways 0
SURFACE concrete
```