How to Provide Track Profiles for Open Rails Dynamic Track

This "How to" guide describes track profiles for dynamic track and superelevation implemented in Open Rails as of V1349. Externally defined track profiles for dynamic track were supported in Open Rails beginning with V402. The purpose of this document is to describe the capabilities in enough detail for users to design and implement track profiles for dynamic track and superelevation.

Contents

Track Profiles

Track profiles are cross sections of track – rails, ties, ballast, etc. – used to specify the geometry of construction for dynamic track and superelevated track. Track profiles are used to generate a mesh of triangles that can be rendered to provide a visual representation of "track." The generation process involves *sweeping* (or *extruding*) the cross section along the railroad's path, whether straight or curved. Unlike predefined track shapes, the cross section can even be twisted dynamically, giving the visual effect of superelevation or cant on curves. The method is termed *generative* or *procedural*.
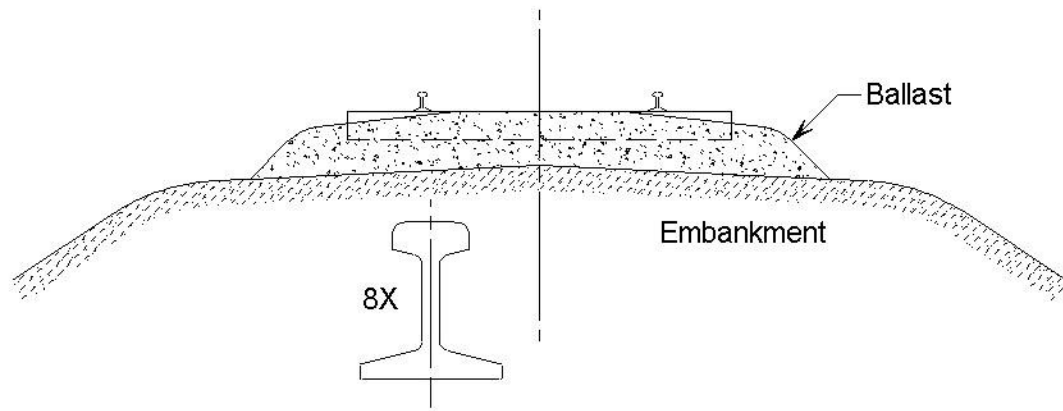
Scale 1:48 drawing of Northern Pacific track profile for single track, rock ballast roadbed. Rail section shown is 80 pound ARA-B. *Redrawn from references provided courtesy of Dave Nelson.*

A representative prototype cross section is shown above. It consists of rail, ties, ballast, and embankment. We can model whatever we want in as much detail as we want. However, we can't handle the circular arcs used for rounds and fillets; we have to break them down into line segments. And, we must trade off (experimentally) the performance impact of detail with the quality of the rendering.

The default profile that Open Rails uses for dynamic track is extremely simple:
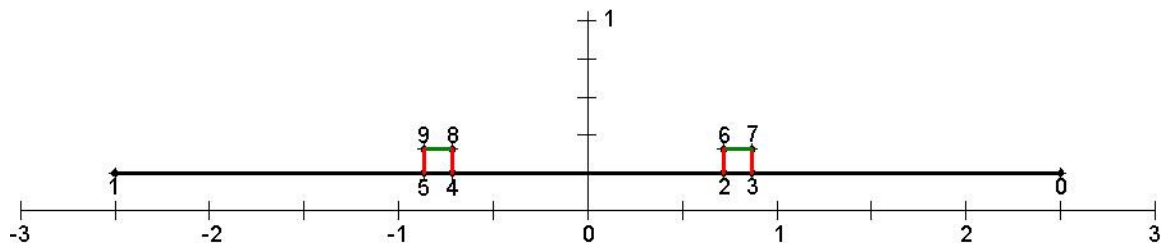
Default profile for Open Rails dynamic track. The scale is in meters.

The long horizontal line segment (black) models the ballast surface and ties. Details like rounding and tie detail are suggested by painting those effects on the texture (image) that is wrapped on the mesh generated. The two short horizontal line segments (green) model the rail tops. The four short vertical line segments (red) model the rail sides (left and right, inner and outer).

The three categories are used as levels-of-detail (LODs). Suppose that you have a camera positioned for an external view of the track, low enough so you can see rail sides and high enough so you can see rail tops. Imagine that you pull back the camera. When you get far enough back, the rail sides will disappear. (You probably won't notice.) You'll still notice the shine off the rail tops. If you continue moving back, the rail tops will disappear. And, if you continue back farther, eventually the ballast LOD will disappear. Then, if you reverse direction and move toward the track sections, the LODs will pop back in (in the reverse order). All this is designed to relieve the load on the processors for complex scenes.

The method of LOD control described above is called *Component Additive*: *Component* because each LOD is a model of a component, some number of which represent the complete track model. *Additive* because, as the camera moves in toward the model, component items are *added*. There is another method available to designers called *Complete Replacement*. There, each LOD item is a *complete* model, more or less detailed than the previous LOD item, and, as an LOD item is added, it *replaces* the previous one. Which method is more useful will depend on the track shape being recreated. This topic will be covered in more detail later.

Track Profile Hierarchy

Track profiles consist of a hierarchy in which an element named *TrProfile* is at the very top of the hierarchy. The screenshot on the next page is *XML Notepad 2007's* view of one of the file formats that can be used to define a track profile for Open Rails. Note the hierarchy in the tree view in the left-hand pane that progresses from TrProfile to LOD to LODItem to Polyline to Vertex as you descend in the tree. Those elements will be discussed in sequence in the subsections below.

Note that there are three LODs. Only the third (Ballast) is expanded to display the full hierarchy. The first two LODs are collapsed to save space.
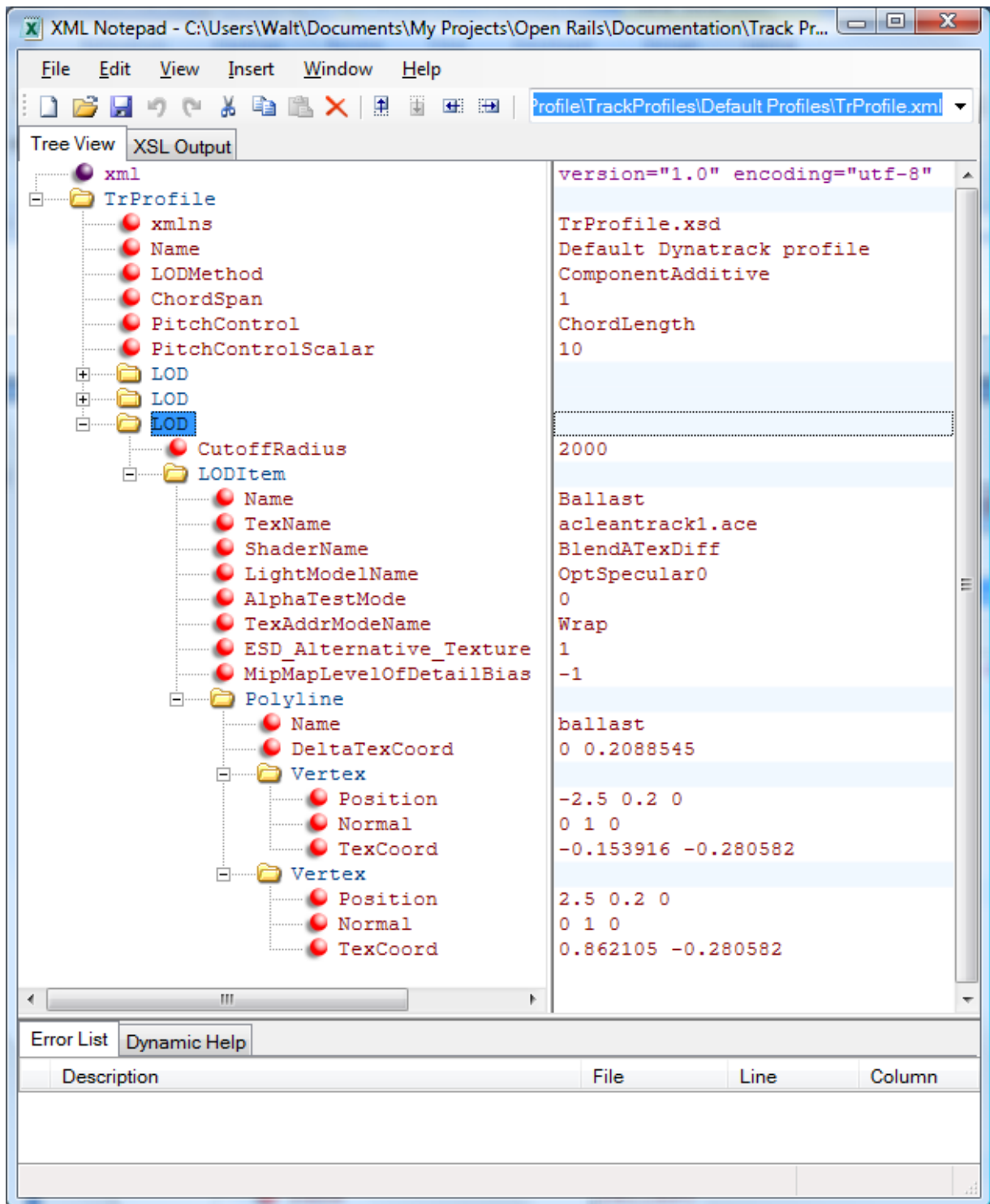
XML Notepad's view of the hierarchy of the default track profile.

TrProfile

The *TrProfile* element is at the head (or root) of the hierarchy. A set of attributes and a collection of *LODItems* comprise a TrProfile as children. The attributes are:

|	
xmlns	Standard for all track profiles that use XML for their definition. The MSTS Structured Text File (STF) format may alternatively be used, in which case this is not required.
Name	A descriptive name for the track profile.

ChordSpan	Specifies the angle (degrees) desired for the pitch with which the profile should be replicated for curves. The profile plus all its replications form a set of cross sections for the track shape.
LODMethod	Identifies the method of LOD control desired – *ComponentAdditive* or *CompleteReplacement*. See LOD Control below for a detailed explanation.
TrackGauge	Determines the gauge (distance between rail heads) of track represented by this track profile, used when generating superelevated track. Default units are meters.
SuperElevationMethod	Decides the style of superelevation used by this track profile when used to generate superelevated track. Valid options are: *None* — This track profile will not generate any visual superelevation. Superelevation will still be calculated for train physics. *Both* — Both rails will be elevated to achieve superelevation; the inside rail will be lowered, and the outside rail raised. *Inside* — Superelevation will only lower the inside rail, the outside rail will remain at the same height. *Outside* — Superelevation will only raise the outside rail, the inside rail will remain at the same height. This is the default setting.
PitchControl	Specifies the method of pitch refinement for curves. It may be: *None* — No refinement desired. *ChordLength* — If the angle specified by *ChordSpan* results in a chord length greater than *PitchControlScalar*, the chord length will be refined to *PitchControlScalar*. *ChordDisplacement* — If the angle specified by *Chordspan* results in a chord displacement (distance between the curve and chord at the point of maximum deviation) greater than *PitchControlScalar*, the chord displacement will be refined to *PitchControlScalar*. See Pitch Control below for a detailed explanation.
PitchControlScalar	Holds the value for ChordLength or ChordDisplacement.
IncludedShapes	(Optional) List of comma separated track shape file names which this track profile should be allowed to replace when generating superelevated track sections. Useful when considering multiple track profiles.
ExcludedShapes	(Optional) Same as *IncludedShapes*, but defines a list of shape file names which should NOT be replaced.
IncludedTextures	(Optional) List of comma separated texture names which will be

	compared to the textures used by the track shape. If the textures match, this track profile will be allowed to replace the track shape when generating superelevated track sections. Useful when considering multiple track profiles.
ExcludedTextures	(Optional) Same as *IncludedTextures*, but defines a list of texture file names a track shape must NOT include to be replaced.

LOD Control

The purpose of LOD control is to handle the way LODs are switched in and out of the rendering queue as a function of the distance between the camera and the object. Two control schemes are implemented: ComponentAdditive and CompleteReplacement.

ComponentAdditive – To see how this scheme works, consider the illustration below.
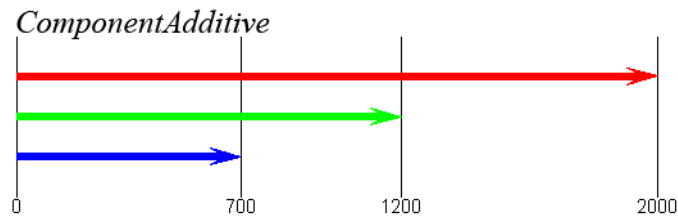
In this example, the blue LOD has a cutoff radius of 700 meters; the green and blue LODs, 1200 and 2000, respectively. That is, when those distances are exceeded, the respective LOD is not added to the rendering queue (i.e., is not seen). Each LOD models a *Component* of the object, and the effect of an observer moving in closer to the camera (from the right end of the distance scale here) is *Additive*. That is, as the observer moves closer, each time they hit a cutoff radius boundary, another component is *added*.

A good example of this scheme is the default track profile where the red, green, and blue arrows correspond to ballast and ties (red), rail tops (green), and rail sides (blue). Starting with the camera close to the object, all three components are seen. As the camera moves away from the object and the 700-meter range is crossed, the rail sides are no longer rendered. Then, as the camera crosses the 1200-meter range, the rail tops leave the render list. Finally, after the camera crosses the 2000-meter range, no component of this object is rendered. Reversing the camera and moving it closer to the object, the components pop into the scene in reverse order, that is, ballast (red) at 2000 meters, rail tops (green) at 1200 meters, and rail sides (blue) at 700 meters.

CompleteReplacement – For the alternative scheme, take a look at the following diagram:
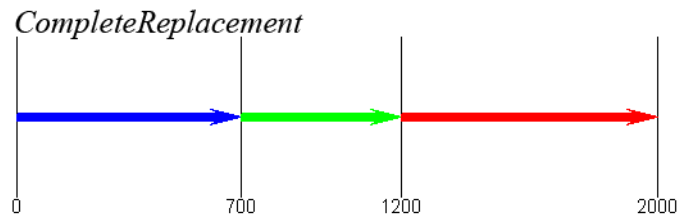
Here, 700, 1200, and 2000 again divide the camera range space into four regions: 0-700 (blue), 700-1200 (green), 1200-2000 (red), and beyond 2000 (nothing). Here, the rules are different: Only one LOD is displayed for a given camera position. And that is the LOD that has a *CutoffRadius* corresponding to the range that the camera is in, that is, 700 (blue), 1200 (green), and 2000 (red). Each LOD must constitute an appropriate *Complete* model for that range (hence the *Complete* in *CompleteReplacement*).

Note that, as the camera crosses a region boundary while moving away from the object, an LOD leaves the scene and is *replaced* by another (except for the last – beyond-2000). This accounts for the *Replacement* in *CompleteReplacement*.

There are certain advantages to each approach. *ComponentAdditive* can reduce the file size and complexity as each component of the track profile only needs to be defined once. *CompleteReplacement* is more powerful as components of the track profile can be given entirely different appearances at each distance level. Ultimately, which of these methods should be used is strictly a designer preference. Note that, with either of these two schemes, <u>it is required that the LOD items be arranged in **_ascending_** order of *CutoffRadius*</u>.

Pitch Control

ChordSpan (described above) is the first level of control of the *pitch* with which profiles are generated to define a mesh for a model of a curved section of dynamic track. *Pitch* refers to the angle subtended by two adjacent profiles, which is a constant for any particular circular-arc section.

ChordSpan pitch control leaves something to be desired for large-radius curves. Hence, two alternative methods have been introduced for refinement of *ChordSpan* control – *ChordLength* and *ChordDisplacement*. The following illustrations define these quantities:
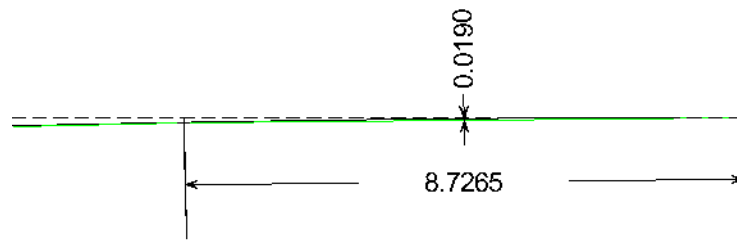

The diagram above is an "eye test." It depicts a curve, which is supposed to be a solid black arc, but it is broken up by interference from the green straight line segments which approximate the circular arc. The dashed black line is tangent to the arc at its point of origin (right). A tiny cross marks the origin of the arc (right). The curve is of 500-meter radius. The diagram depicts one pitch unit of one degree. There is a second tiny cross at the top of the left-hand extension line for the 8.7265 dimension. The second cross marks the point where the curve has deflected one degree (one pitch unit). In fact, the left-hand extension line has a severe case of the "jaggies" because, superimposed with it is a one-degree radial line to the center of the arc.

The green line segments (two of them) are chords spanning one-degree segments. The 8.7265 (meters) dimension is the length of the chord. It is this length that is tested for *ChordLength* control.

<u>ChordLength Control</u> – Under *ChordLength* control, when *ChordSpan* control (one degree in this example) yields a chord length that is greater than *PitchControlScalar* , then the pitch will be refined such that the chord length is exactly *PitchControlScalar*.

One suggested "rule of thumb" is that target chord lengths should be held to a maximum in the 10.0-12.5 meter range.

<u>ChordDisplacement Control</u> – Referring again to the diagram above, the other dimension illustrated (the 0.0190-meter dimension) refers to the miniscule distance between the arc and chord at its point of maximum excursion, which is at the center of the chord. Just to be sure that this is clear, here is a "blow-up" of the dimension. Note that there is, in fact, a miniscule red line segment that identifies the maximum excursion.
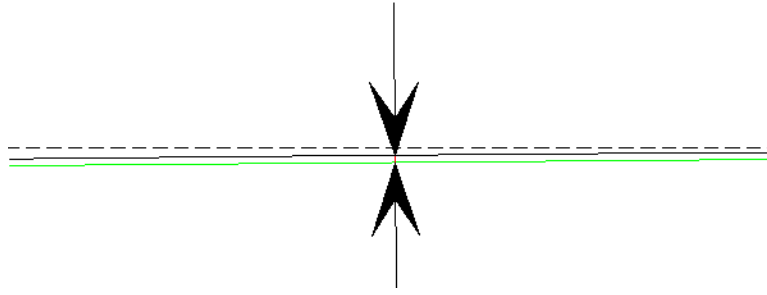
Under *ChordDisplacement* control, when *ChordSpan* control fails to produce a chord displacement less than *PitchControlScalar*, then the pitch will be refined such that the chord displacement is exactly *PitchControlScalar*.

One suggested guideline for profile designers is to consider a target chord displacement as a fraction of railhead width. For example, the 19 mm chord displacement in this one-degree pitch example is 28 percent of a 68 mm railhead width.

For more on pitch control, see the Appendix.

Multiple Track Profiles

Some routes in Open Rails use multiple track systems with different cross sections and/or textures. In this case, a singular track profile will be insufficient for the superelevation[*] feature, resulting in some curves either taking on an appearance that is inconsistent with the tangent track attached to the curve, or simply being rendered without any superelevation at all. To solve this problem, it is required to employ more than one track profile. Ideally, each visually distinct type of track present on the route should be given its own profile, each following the same process outlined in this document. For details on configuring multiple track profile files, see the track profile files section.
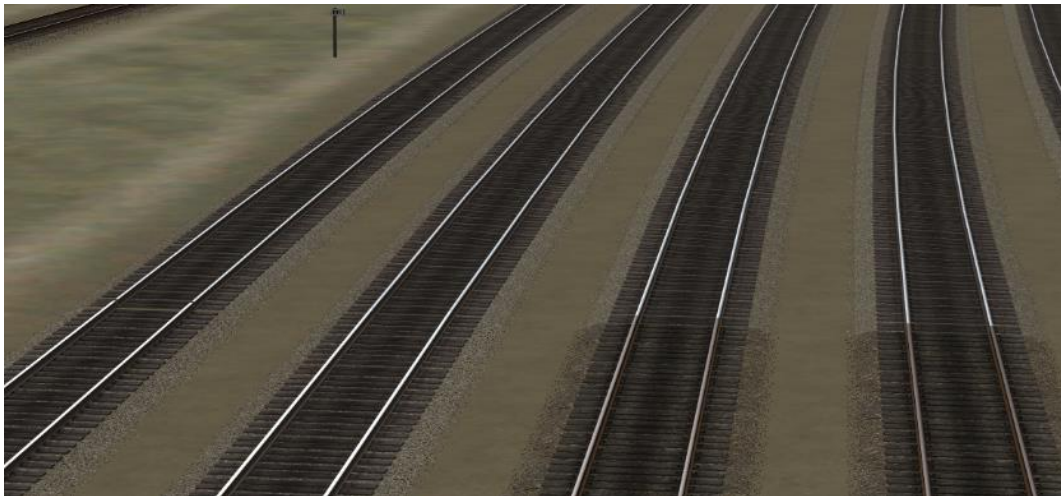
The singular track profile on this route looks great for the mainline (left) but is unsightly when used on the yard track (right).

Once multiple track profile files have been created, the next challenge is determining which track profile should be used by each track segment. By default, Open Rails will decide which track profile to use based on the texture file names *(.ace or .dds does not matter, the file extension is ignored)* used by the original track shape. Whichever track profile uses the *TexName*s that most closely match the original textures will be used to generate superelevation. If the track profile is missing textures used by the shape, or uses textures the shape doesn't use, then the track profile is less likely to be selected. If no

---

[*] Due to limitations in the way dynamic track is implemented, multiple track profiles are only supported when replacing static track shapes with superelevation. Dynamic track will always use the default "TrProfile" file, regardless of the number or configuration of secondary track profiles.

track profile appears to be suitable, then the original shape will be used without any superelevation applied.

While the texture approach is adequate for most situations as very few visually distinct track systems use the same textures, it may be desirable to gain more control over the process for selecting track profiles to replace track shapes. If you observe track sections on your route being replaced with the visually inconsistent profiles during superelevation despite adding multiple track profiles, this is a sign that manual intervention is required. As such, some additional (optional) attributes can be added to a track profile to enforce custom selection rules.

Included and Excluded Filters

To allow a track profile to replace only certain track sections and to prevent it from replacing other track sections, some inclusion and exclusion filters can be added to the track profile file as attributes of *TrProfile*. If any of these filters are present in a track profile, the previously mentioned texture matching behavior is disabled entirely, so only the filters will determine the track profile used. Additionally, track profiles with filters defined are inherently preferred over those profiles using the default matching behavior.

All filter attributes can accept a list of multiple filters, where the entire set of filters is contained in quotation marks and each individual filter is separated by a comma. For example, in STF: *IncludedTextures ("acleantrack1, acleantrack2")* is valid syntax, but *IncludedTextures ("acleantrack1" "acleantrack2")* is not. Similarly, in XML *IncludedTextures="acleantrack1, acleantrack2"* would work but *IncludedTextures="acleantrack1" "acleantrack2"* would fail.

For flexibility, filters can also include the wildcard characters '*' and '?'. As in other programs, the * wildcard represents any number of characters, while the ? wildcard represents a single character only. For example, *IncludedTextures ("acleantrack?")* would match to "acleantrack1", but would not match "acleantrack" or "acleantrack37". Meanwhile, *IncludedTextures ("acleantrack*")* would match all 3 examples.

Currently, there are two types of included and excluded filters: *textures* and *shapes*. The *IncludedTextures* and *ExcludedTextures* attributes will compare their filters against the *texture names* used by the original track shape. Conversely, *IncludedShapes* and *ExcludedShapes* will compare their filters to the *shape file name* of the original track shape. Note that <u>all filters ignore case and don't include file extensions</u>. If file extensions are included in a filter, it will never produce a match.

The *Included* filters will attempt to match at least one filter to the shape/texture names. If none of the filters match, then the track profile will not be used by the shape. If multiple filters match, then the track profile is more likely to be selected. On the other hand, if the shape/texture names match just one of the *Excluded* filters, the track profile won't be used by the shape.

With the included and excluded features combined, it is possible to highly fine-tune which track profiles will be used with which track shapes, though care must be taken to include all the intended shapes/textures, and not to exclude any important shapes/textures.

Much better! With the addition of a second track profile using the yard track texture, now both sets of track look just like we want.

Returning to the description of the structure of a track profile, after the root item, *TrProfile*, next in the hierarchy is a list of the one or more LODs that comprise the track profile, arranged in order of increasing *CutoffRadius* (see below).

LOD

The only purpose for an LOD is to define the *component* model for a *ComponentAdditive* model or the *complete* model for a *CompleteReplacement* model. Each LOD has an associated *CutoffRadius* that defines the range of camera positions where the LOD is seen. An LOD has only one attribute:

CutoffRadius	The end-of-range distance between the center of this track section and the camera. This LOD will not be rendered when this distance is exceeded. (The start-of-range will be the next previous LOD with a *CutoffRadius* less than this one.)

LODs give control over model complexity as a function of distance of the object from the camera. As the distance increases, the model can become less complex, which economizes hardware/software performance.

Each LOD consists of one or more *LODItems*.

LODItem

Each *LODItem* that follows as a child of an LOD serves a single purpose: It serves as a (geometry, material) unit. All material properties (including texture) are introduced at this level.

Again, attributes and a collection comprise a LODItem. The attributes are:

Name	A descriptive name for the LOD.
TexName	The filename (with file extension) for the texture to be used for this LOD.
ShaderName	A material property that controls how light reflects off the surface of the LOD item:
	Diffuse Diffuse light reflected.
	TexDiff Diffuse light reflected.
	BlendATexDiff Diffuse light reflected with alpha-testing

|		
		and blending.
	AddATexDiff	Diffuse light reflected with alpha-testing and blending.
	Tex	Diffuse light not reflected.
	BlendATex	Diffuse light not reflected with alpha-testing and blending.
	AddATex	Diffuse light not reflected with alpha-testing and blending.
LightModelName	A material property modeling how an LOD item is lit.	
	DarkShade	Darkly shaded.
	OptHalfBright	Half lit.
	OptFullBright	Fully lit.
	Cruciform	Simulates lighting on vegetation.
	CruciformLong	Simulates lighting on vegetation.
	OptSpecular0	Dull finish.
	OptSpecular25	Shiny.
	OptSpecular750	Very shiny.
AlphaTestMode	A material property that controls whether or not alpha-testing is done.	
	0	No alpha-testing.
	1	Alpha-testing is performed, and, if the texture's alpha channel is less than *ReferenceAlpha,* the texture is opaque at a pixel; otherwise it is transparent. *ReferenceAlpha* is 10 for all of the shaders with "ATex" in their names and 200 otherwise.
TexAddrModeName	A material property that determines how the texture gets applied to the surface of the LOD item.	
	Wrap	Tile the texture at every integer value of a texture coordinate (u, v). For example, for u values between 0 and 3, the texture is repeated three times.
	Mirror	Similar to *Wrap*, except that the texture is flipped at every integer value of (u, v).
	Clamp	Texture coordinates less than 0 are set to the texture color at 0, and texture coordinates greater than 1 are set to the texture color at 1.
	Border	Texture coordinates outside the range 0-1 are set to the border color.
ESD_Alternative_Texture	Equivalent to the MSTS parameter of the same name. This integer indicates where OpenRails should look to find alternative textures, depending on weather and time-of-day:	

	0: The LODItem has no alternative texture. The base texture is used all of the time.
	1: The LODItem has two textures, the base texture and one in the Snow subdirectory.
	252: The LODItem has alternative textures to cover seasonal variations with and without snow. These should go in Summer, SummerSnow, Autumn, AutumnSnow, Winter, and WinterSnow subdirectories.
	256: The LODItem has one alternative texture, in the Night subdirectory.
MipMapLevelOfDetailBias	A number that can be "tweaked" to sharpen or blur textures. Negative changes sharpen; positive changes blur. (0 means no bias.)

The collection that follows the attributes of an LODItem is a collection of Polylines.

The last two LODItems are not expanded in the XML Notepad screenshot to save space. In XML Notepad, the LODItems can be expanded to show their Polyline children simply by clicking the '+' sign.

Polyline

A *Polyline* element is a contiguous[*] set of straight line segments that connect a set of points. For example, consider the illustration to the right. The part labeled "0" is not a polyline; it's a point. "1–2" is a polyline, a single-segment polyline. "3 – 4 – 5" is a two-segment polyline. "6 – 7 – 8 – 9 – 6" is a four-segment closed polyline, rightfully called a *polygon*. "10 – 11   12 – 13" is not a polyline; it is two polylines.
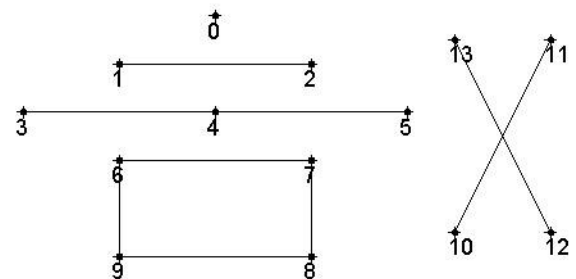
Each Polyline element is defined in the track profile file by a set of attributes and a collection of child elements. The attributes follow:

Name	A descriptive name for the polyline.
DeltaTexCoord	The change in texture coordinates (Δu, Δv) per meter along the path of the track centerline.

It is obvious from the examples above that a polyline is defined by a collection of points, in addition to its attributes. We term these vertices because they describe more than just the location. The collection that constitutes the child elements of a polyline is a collection of vertices.

Vertex

A *Vertex*, by definition, is at the bottom of the hierarchy tree because it has no children. It has only attributes, which are as follows:

Position	The position vector (x, y, z) at the point relative to the center (root) of the track section when no superelevation is applied. The position will be automatically adjusted to 'twist' the track for superelevation at runtime.

[*]   Contiguous – arranged head-to-tail.

Normal	A unit normal vector (n_x, n_y, n_z) at the point, which is used for lighting calculations.
TexCoord	A texture coordinate (*u, v*) at the point, which is used for texture wrapping.
PositionControl	Optional attribute to determine how this vertex is allowed to move when twisted by superelevation. This can be useful for avoiding undesired twisting and stretching of certain elements (eg: the embankment the track is built on) of the track profile.

	None	The position of this vertex won't be adjusted for superelevation, preventing twisting of this vertex.
	All	The default setting, the vertex position will be adjusted by superelevation as needed.
	Inside	Vertex position will only be adjusted for superelevation if on the inside of a curve.
	Outside	Vertex position will only be adjusted for superelevation if on the outside of a curve.

Track Profile Hierarchy Summary

Before leaving the topic of the track profile hierarchy, it is important to emphasize quantities:

- There is one and only one *TrProfile*, which serves as the root of the hierarchy tree.
- There may be one or more *LODs*; there must be at least one. The *LODs* must be sorted in *TrProfile* in order of increasing *CutoffRadius*. Multiple *LODs* with the same *CutoffRadius* are not allowed.
- There may be one or more *LODItems* for each *LOD*; there must be at least one.
- There may be one or more *Polylines* for each *LODItem*; there must be at least one.
- There may be <u>two</u> or more *Vertices* for each *Polyline*; there must be at least <u>two</u>.

Track Profile Files

Track profiles are defined by external files placed in the "TrackProfiles" subfolder inside a route folder. Note that this means each route requires its own TrackProfiles folder containing a copy of track profile files.

All track profiles must have a file name beginning with "TrProfile", though the file extension changes from style to style. While multiple track profiles are allowed, special care must be given for the file named "TrProfile" as this will be used as the profile for every section of dynamic track. However, it is not required to have a "TrProfile" file; if Open Rails finds no TrProfile.*xxx* in the TrackProfiles folder for a route (as is the case for unmodified MSTS routes), where *xxx* is a supported file extension, then it builds a "canned" track profile in memory. The "canned" profile looks identical to the one you're used to for Kuju dynamic track. If your route does not use Kuju track shapes, it is recommended to make a custom "TrProfile" file based on the most common track shape used by the route, or else the appearance of dynamic track will not match static track shapes.

If multiple track profiles are to be used, remember to begin the file name for each additional track profile with "TrProfile". For example, "TrProfileConcrete" will be recognized as a secondary track profile by Open Rails, but "ConcreteProfile" will not be. The file name of additional track profiles should reflect the track the profile has been modeled after to reduce confusion. There is no hard limit on the number of track profiles, but best practice is to only include track profiles for track types used on the route as each additional profile requires additional computing time. Each time a track shape is converted

to superelevation, Open Rails will iterate through all in *alphabetical order* with the default "TrProfile" processed first. As a result, if two track profiles are equally suited to replace a static track shape, Open Rails will select whichever profile comes first in the alphabet.

Currently, two track profile file styles (or formats) are supported – MSTS STF-style and XML-style.[*] If both an .XML and .STF track profile file with the same name exist, Open Rails will load the XML profile in preference to the STF.

STF-Style

MSTS veterans probably have experience with the complete parenthesis syntax of ENG, WAG, etc. files. Take a look at what an STF (structured text format)-style track profile file looks like for the default profile (page 16).

It's relatively clean when formatted this way, and you may be used to it. The downside is that validation is minimal, and you may have problems figuring out where you made an error. (But MSTS veterans don't make errors.[†])

The MSTS-style track profile files have a file extension of ".stf".

All STF files must have a SIMIS-style header on Line 1, which is specific to profiles. It must be: "`SIMISA@@@@@@@@@@JINX0p0t_____`" (without the quotation marks).

XML-Style

You've already seen what XML (extended markup language)-style track profile files look like when viewed with XML Notepad 2007. To see what an XML track profile really looks like, see page 17.

There's not a lot of difference. The principal difference is that XML uses tags (< … >) to bound elements and *name = value* syntax to define attributes. Also, XML bounds all values with quotes. So, there's a little bit more typing required by the XML style. But then you'll probably be using an XML editor (e.g., XML Notepad 2007).

But wait. There is one additional file required by the XML style – TrProfile.xsd. But there is one of these provided for you, and you had better not modify it. Modify it, and the code breaks!

TrProfile.xsd defines (in XML) the syntax of a track profile. It drives the validation code in Open Rails used to check the syntax of your track profile (TrProfile.xml).

All you must remember is to put your TrProfile.xml and the standard TrProfile.xsd in the TrackProfiles folder in the root folder of the desired route. (If there is sufficient interest in support for XML, TrProfile.xsd will be treated as Open Rails "content," and you won't have to worry about it.)

---

[*] Hopefully, experience will reveal the superior style, and one will be deprecated in the future.

[†] They'll blame it on the programmer.

Installation

Installation is simple:

1. Select a style: STF or XML.
2. If there is no TrackProfiles folder in the root folder of the desired route, make one.
3.
    a. If STF, copy all required TrProfile*.stf files to the route's TrackProfiles folder. If there are any TrProfile*.xml files with the same in the folder, delete them, rename them, or move them elsewhere. (TrProfile*.xml takes precedence over TrProfile*.stf if both in the folder have the same name.)
    b. If XML, copy all TrProfile*.xml and TrProfile.xsd to the route's TrackProfiles folder.
4. Start the game. When the game loads, you'll notice an additional item logged on the "Loading …" line in the log file. It is "TRP," which stands for TrackProfile. As the track profile(s) are loaded, additional strings will be logged to indicate the type of each track profile loaded: "Default," "STF," or "XML" is output in parenthesis following the "TRP."
5. Observe the visual appearance of track in the 3D world. If installed and configured correctly, all sections of dynamic track or superelevated track should fit in naturally.
    a. If a section of dynamic track or superelevated track does not look correct, this may indicate the wrong track profile has been used or the track profile is configured incorrectly.

```
SIMISA@@@@@@@@@@JINX0p0t_____
TrProfile ( Name ( "Default Dynatrack Profile" ) LODMethod ( "ComponentAdditive" ) ChordSpan ( 1 )
            PitchControl ( "ChordLength" ) PitchControlScalar ( 10 ) TrackGauge ( 1.435 ) SuperElevationMethod ( "Outside" )
        LOD ( CutoffRadius ( 700 )
            LODItem ( Name ( "Railsides" ) TexName ( "acleantrack2.ace" ) ShaderName ( "TexDiff" )
                    LightModelName ( "OptSpecular0" ) AlphaTestMode ( 0 ) TexAddrModeName ( "Wrap" )
                    ESD_Alternative_Texture ( 0 ) MipMapLevelOfDetailBias ( 0 )
                    Polyline ( Name ( "left_outer" ) DeltaTexCoord ( 0.1673372 0 )
                            Vertex ( Position ( -0.8675 0.200 ) Normal ( -1 0 0 ) TexCoord ( -0.139362  0.101563 ) )
                            Vertex ( Position ( -0.8675 0.325 ) Normal ( -1 0 0 ) TexCoord ( -0.139363  0.003906 ) )
                    )
                    Polyline ( Name ( "left_inner" ) DeltaTexCoord ( 0.1673372 0 )
                            Vertex ( Position ( -0.7175 0.325 ) Normal (  1 0 0 ) TexCoord ( -0.139363  0.003906 ) )
                            Vertex ( Position ( -0.7175 0.200 ) Normal (  1 0 0 ) TexCoord ( -0.139362  0.101563 ) )
                    )
                    Polyline ( Name ( "right_inner" ) DeltaTexCoord ( 0.1673372 0 )
                            Vertex ( Position (  0.7175 0.200 ) Normal ( -1 0 0 ) TexCoord ( -0.139362  0.101563 ) )
                            Vertex ( Position (  0.7175 0.325 ) Normal ( -1 0 0 ) TexCoord ( -0.139363  0.003906 ) )
                    )
                    Polyline ( Name ( "right_outer" ) DeltaTexCoord ( 0.1673372 0 )
                            Vertex ( Position (  0.8675 0.325 ) Normal (  1 0 0 ) TexCoord ( -0.139363  0.003906 ) )
                            Vertex ( Position (  0.8675 0.200 ) Normal (  1 0 0 ) TexCoord ( -0.139362  0.101563 ) )
                    )
            )
        )
        LOD ( CutoffRadius ( 1200 )
            LODItem ( Name ( "Railtops" ) TexName ( "acleantrack2.ace" ) ShaderName ( "TexDiff" )
                    LightModelName ( "OptSpecular25" ) AlphaTestMode ( 0 ) TexAddrModeName ( "Wrap" )
                    ESD_Alternative_Texture ( 0 ) MipMapLevelOfDetailBias ( 0 )
                    Polyline ( Name ( "right" ) DeltaTexCoord ( 0.0744726 0)
                            Vertex ( Position ( -0.8675 0.325 ) Normal ( 0 1 0 ) TexCoord (  0.232067  0.126953 ) )
                            Vertex ( Position ( -0.7175 0.325 ) Normal ( 0 1 0 ) TexCoord (  0.232067  0.224609 ) )
                    )
                    Polyline ( Name ( "left" ) DeltaTexCoord ( 0.0744726 0 )
                            Vertex ( Position (  0.7175 0.325 ) Normal ( 0 1 0 ) TexCoord (  0.232067  0.126953 ) )
                            Vertex ( Position (  0.8675 0.325 ) Normal ( 0 1 0 ) TexCoord (  0.232067  0.224609 ) )
                    )
            )
        )
        LOD ( CutoffRadius ( 2000 )
            LODItem ( Name ( "Ballast" ) TexName ( "acleantrack1.ace" ) ShaderName ( "BlendATexDiff" )
                    LightModelName ( "OptSpecular0" ) AlphaTestMode ( 0 ) TexAddrModeName ( "Wrap" )
                    ESD_Alternative_Texture ( 1 ) MipMapLevelOfDetailBias ( -1 )
                    Polyline  ( Name ( "ballast" ) DeltaTexCoord ( 0 0.2088545 )
                            Vertex ( Position ( -2.5000 0.200 ) Normal ( 0 1 0 ) TexCoord ( -0.153916 -0.280582 ) )
                            Vertex ( Position (  2.5000 0.200 ) Normal ( 0 1 0 ) TexCoord (  0.862105 -0.280582 ) )
                    )
            )
        )
)
```

MSTS-Style Track Profile (TrProfile.stf)

```xml
<?xml version="1.0" encoding="utf-8"?>
<TrProfile xmlns="TrProfile.xsd" Name="Default Dynatrack profile" LODMethod="ComponentAdditive" ChordSpan="1"
        PitchControl="ChordLength" PitchControlScalar="10" TrackGauge="1.435" SuperElevationMethod="Outside">
  <LOD CutoffRadius="700">
    <LODItem Name="Railsides" TexName="acleantrack2.ace" ShaderName="TexDiff" LightModelName="OptSpecular0"
            AlphaTestMode="0" TexAddrModeName="Wrap" ESD_Alternative_Texture="0" MipMapLevelOfDetailBias="0">
      <Polyline Name="left_outer" DeltaTexCoord="0.1673372 0">
        <Vertex Position="-0.8675 0.2 0" Normal="-1 0 0" TexCoord="-0.139362 0.101563" />
        <Vertex Position="-0.8675 0.325 0" Normal="-1 0 0" TexCoord="-0.139363 0.003906" />
      </Polyline>
      <Polyline Name="left_inner" DeltaTexCoord="0.1673372 0">
        <Vertex Position="-0.7175 0.325 0" Normal="1 0 0" TexCoord="-0.139363 0.003906" />
        <Vertex Position="-0.7175 0.2 0" Normal="1 0 0" TexCoord="-0.139362 0.101563" />
      </Polyline>
      <Polyline Name="right_inner" DeltaTexCoord="0.1673372 0">
        <Vertex Position="0.7175 0.2 0" Normal="-1 0 0" TexCoord="-0.139362 0.101563" />
        <Vertex Position="0.7175 0.325 0" Normal="-1 0 0" TexCoord="-0.139363 0.003906" />
      </Polyline>
      <Polyline Name="right_outer" DeltaTexCoord="0.1673372 0">
        <Vertex Position="0.8675 0.325 0" Normal="1 0 0" TexCoord="-0.139363 0.003906" />
        <Vertex Position="0.8675 0.2 0" Normal="1 0 0" TexCoord="-0.139362 0.101563" />
      </Polyline>
    </LODItem>
  </LOD>
  <LOD CutoffRadius="1200">
    <LODItem Name="Railtops" TexName="acleantrack2.ace" ShaderName="TexDiff" LightModelName="OptSpecular25"
            AlphaTestMode="0" TexAddrModeName="Wrap" ESD_Alternative_Texture="0" MipMapLevelOfDetailBias="0">
      <Polyline Name="right" DeltaTexCoord="0.0744726 0">
        <Vertex Position="-0.8675 0.325 0" Normal="0 1 0" TexCoord="0.232067 0.126953" />
        <Vertex Position="-0.7175 0.325 0" Normal="0 1 0" TexCoord="0.232067 0.224609" />
      </Polyline>
      <Polyline Name="left" DeltaTexCoord="0.0744726 0">
        <Vertex Position="0.7175 0.325 0" Normal="0 1 0" TexCoord="0.232067 0.126953" />
        <Vertex Position="0.8675 0.325 0" Normal="0 1 0" TexCoord="0.232067 0.224609" />
      </Polyline>
    </LODItem>
  </LOD>
  <LOD CutoffRadius="2000">
    <LODItem Name="Ballast" TexName="acleantrack1.ace" ShaderName="BlendATexDiff" LightModelName="OptSpecular0"
            AlphaTestMode="0" TexAddrModeName="Wrap" ESD_Alternative_Texture="1" MipMapLevelOfDetailBias="-1">
      <Polyline Name="ballast" DeltaTexCoord="0 0.2088545">
        <Vertex Position="-2.5 0.2 0" Normal="0 1 0" TexCoord="-0.153916 -0.280582" />
        <Vertex Position="2.5 0.2 0" Normal="0 1 0" TexCoord="0.862105 -0.280582" />
      </Polyline>
    </LODItem>
  </LOD>
</TrProfile>
```
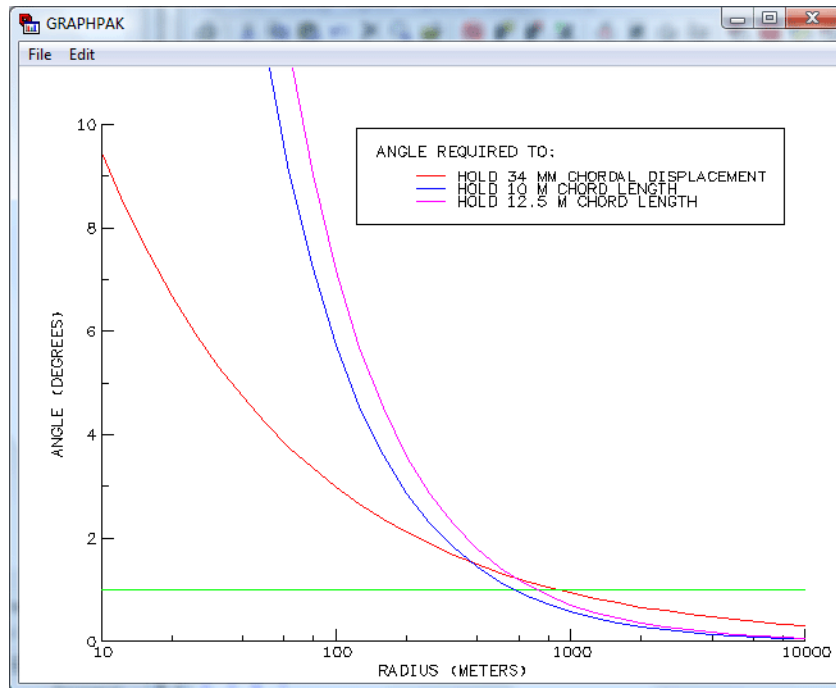
XML Track Profile (TrProfile.xml)

Appendix – More on Pitch Control

For more insight into pitch control, consider the following plot:

The vertical axis of the plot represents the pitch angle of a series of straight-line segment used as an approximation to a circular-arc curve; the horizontal axis represents the radius of the curve. The green line represents a constant one-degree pitch angle, independent of radius – what you get with a *ChordSpan* control scheme.

Look at the blue curve, which is a plot of the pitch angle required to hold a 10-meter chord length as a function of curve radius. Note that it crosses the one-degree line at about 600 meters radius. (Note that the radius axis has a logarithmic scale.) This tells you that, above 600 meters radius, you will be missing a target of 10-meter chord length with a one-degree chord span. Similarly, you will be missing a target of 12.5-meter chord length (magenta curve) for radii above about 700 meters. If you seek a target of 34 mm chord displacement (half a rail width), you will miss it with a one-degree chord span for radii above about 900 meters.

The *ChordLength* and *ChordDisplacement* scheme are, in fact, algebraically related. If L is the chord length and d is the chord displacement, then:

$$L \left(1 - \cos\frac{\theta}{2}\right) = 2\,d\,\sin\frac{\theta}{2}$$

for any given chord span θ. Thus, L and d are linearly related for any given chord span, θ. For example, for a one-degree chord span:

$$d = .00218\,L$$

Thus, for a one-degree chord span, a 10-meter chord length would yield a chord displacement of 21.8 mm.