Panda3DdevManual Documentation

Release 0.0.1

frainfreeze

Aug 15, 2018

Table of Contents

1 Preface **3**

2 Source tree **5**

2.1 contrib . . . 5
2.2 direct . . . 5
2.3 dmodels . . . 5
2.4 dtool . . . 5
2.5 makepanda . . . 6
2.6 models . . . 6
2.7 panda . . . 6
2.7.1 android . . . 6
2.7.2 androiddisplay . . . 6
2.7.3 audio . . . 6
2.7.4 audiotraits . . . 6
2.7.5 awesomium . . . 6
2.7.6 bullet . . . 6
2.7.7 cftalk . . . 7
2.7.8 chan . . . 7
2.7.9 char . . . 7
2.7.10 cocoadisplay . . . 7
2.7.11 collada . . . 7
2.7.12 collide . . . 7
2.7.13 configfiles . . . 7
2.7.14 cull . . . 7
2.7.15 device . . . 7
2.7.16 dgraph . . . 7
2.7.17 display . . . 8
2.7.18 distort . . . 8
2.7.19 doc . . . 8
2.7.20 downloader . . . 8
2.7.21 downloadertools . . . 8
2.7.22 dxgsg9 . . . 8
2.7.23 dxml . . . 8
2.7.24 egg . . . 8
2.7.25 egg2pg . . . 8
2.7.26 egldisplay . . . 9

2.7.27 event . . . 9
2.7.28 express . . . 9
2.7.29 ffmpeg . . . 9
2.7.30 framework . . . 9
2.7.31 gles2gsg . . . 9
2.7.32 glesgsg . . . 9
2.7.33 glgsg . . . 9
2.7.34 glstuff . . . 9
2.7.35 glxdisplay . . . 9
2.7.36 gobj . . . 9
2.7.37 grutil . . . 10
2.7.38 gsgbase . . . 10
2.7.39 iphone . . . 10
2.7.40 iphonedisplay . . . 10
2.7.41 linmath . . . 10
2.7.42 mathutil . . . 10
2.7.43 movies . . . 10
2.7.44 nativenet . . . 10
2.7.45 net . . . 10
2.7.46 ode . . . 10
2.7.47 osxdisplay . . . 10
2.7.48 pandabase . . . 11
2.7.49 parametrics . . . 11
2.7.50 particlesystem . . . 11
2.7.51 pgraph . . . 11
2.7.52 pgraphnodes . . . 11
2.7.53 pgui . . . 11
2.7.54 physics . . . 11
2.7.55 physx . . . 11
2.7.56 pipeline . . . 11
2.7.57 pnmimage . . . 11
2.7.58 pnmimagetypes . . . 12
2.7.59 pnmtext . . . 12
2.7.60 pstatclient . . . 12
2.7.61 putil . . . 12
2.7.62 recorder . . . 12
2.7.63 rocket . . . 12
2.7.64 skel . . . 12
2.7.65 speedtree . . . 12
2.7.66 testbed . . . 12
2.7.67 text . . . 12
2.7.68 tform . . . 12
2.7.69 tinydisplay . . . 13
2.7.70 vision . . . 13
2.7.71 vrpn . . . 13
2.7.72 wgldisplay . . . 13
2.7.73 windisplay . . . 13
2.7.74 x11display . . . 13
2.8 pandatool . . . 13
2.9 samples . . . 13

3 File formats, specifications and similar **15**
3.1 The GraphicsEngine . . . 15
3.1.1 GraphicsPipe . . . 15

3.1.2 GraphicsWindow and GraphicsBuffer . . . 16
3.1.3 GraphicsStateGuardian . . . 16
3.1.4 Renderinga frame . . . 16
3.1.5 Using a GraphicsEngine to create windows and buffers . . . 17
3.1.6 Sharinggraphics contexts . . . 18
3.1.7 Closingwindows . . . 18
3.2 ppython . . . 18
3.3 Panda Audio Documenation . . . 18
3.3.1 AudioManagerand AudioSound . . . 18
3.3.2 ExampleUsage . . . 19
3.4 Codingstyle . . . 19
3.5 COLLISIONFLAGS . . . 22
3.6 eggpalettize . . . 23
3.6.1 HOWTO USE EGG_PALETTIZE . . . 23
3.6.2 GROUPINGEGG FILES . . . 24
3.6.3 CONTROLLINGTEXTURE PARAMETERS . . . 25
3.6.4 RUNNINGEGG-PALETTIZE . . . 26
3.6.5 WHENTHINGS GO WRONG . . . 26
3.7 THEPHILOSOPHY OF EGG FILES . . . 27
3.7.1 GENERALEGG SYNTAX . . . 28
3.7.2 LOCALINFORMATION ENTRIES . . . 29
3.7.3 GLOBALINFORMATION ENTRIES . . . 29
3.7.4 GEOMETRYENTRIES . . . 44
3.7.5 PARAMETRICDESCRIPTION ENTRIES . . . 49
3.7.6 MORPHDESCRIPTION ENTRIES . . . 53
3.7.7 GROUPINGENTRIES . . . 54
3.7.8 GROUPBINARY ATTRIBUTES . . . 54
3.7.9 GROUPSCALARS . . . 55
3.7.10 OTHERGROUP ATTRIBUTES . . . 57
3.7.11 name { type [flags] } . . . 58
3.7.12 {type} . . . 60
3.7.13 MISCELLANEOUS . . . 66
3.7.14 ANIMATIONSTRUCTURE . . . 67
3.8 HOWTO CONTROL RENDER ORDER . . . 71
3.8.1 CULLBINS . . . 72
3.9 Howto make multipart actor . . . 74
3.9.1 MULTIPARTACTORS vs. HALF-BODY ANIMATION . . . 74
3.9.2 BROADOVERVIEW . . . 75
3.9.3 MOREDETAILS . . . 75
3.9.4 NUTS AND BOLTS . . . 75
3.10 MULTIGENMODEL FLAGS . . . 77
3.10.1 HANDLES . . . 78
3.10.2 BEHAVIORS . . . 78
3.10.3 PROPERTIES . . . 79
3.11 Multi-Texturing in Maya . . . 80
3.12 Config . . . 81
3.12.1 Using the prc files . . . 81
3.12.2 Defining config variables . . . 83
3.12.3 Directly assigning config variables . . . 85
3.12.4 Querying config variables . . . 86
3.12.5 Re-reading prc files . . . 87
3.12.6 Runtime prc file management . . . 88
3.12.7 Compile-time options for finding prc files . . . 89
3.12.8 Executable prc files . . . 91

3.12.9 Signed prc files 91

4 Miscellaneous and F.A.Q. **97**

5 Appendix **99**

Contents:

CHAPTER 1

Preface

Book is organized in tree structure that follows panda 1.9 source tree. It might also contain doc strings and comments from the source code.

Manual is built from several parts:

1. Source tree structured.
2. File formats, specifications and similar
3. Miscellaneous and F.A.Q.
4. Appendix

Note: Maunal may be from 2014-2016 however most of the content is much older. Some parts date from 2002 and lots of information might be deprecated.

CHAPTER 2

Source tree

2.1 contrib

Code contributed by the community. This code is usually not maintained by the developers but by the respective community contributors.

2.2 direct

Mid-level tools/subsystems which supports show development, and scene-composition. It contains pretty much *all* of Panda's Python code with some C++.Includes code which sets up and initializes PANDA (using Python wrapper functions which call low-level C++ counterparts).Includes python modules for mid-level show coding systems: actors, directdevices (high-level wrappers around low-level input devices such as joysticks, magnetic trackers, etc.), finite state machines, 2D gui elements, intervals, tasks, and the DIRECT tk widget classes and panels.

2.3 dmodels

Similar to /models but processed by makepanda, models that still need to be converted to .egg at build time

Their origin is probably Disney

2.4 dtool

This tree contains base classes and core functionality that the other Panda libraries rely on, such as basic threading constructs, file reading/writing constructs and the configuration system. It also contains interrogate, which is used to generate Python bindings for Panda3D.

2.5 makepanda

Panda3D building system.

2.6 models

Contains some free models for use in samples. They origin is probably CMU

2.7 panda

Low-level 3D graphics engine code. Primarily C++.

Includes code for graphics/scene graph setup/manipulation/rendering, graphic state guardians (interfaces to OpenGL, Direct X, tinypanda(based on TinyGL)), and source code for many PANDA systems: animation, audio, gui, input devices, particles, physics, shaders, etc.

2.7.1 android

-update me-

2.7.2 androiddisplay

-update me-

2.7.3 audio

-update me-

2.7.4 audiotraits

-update me-

2.7.5 awesomium

-update me-

2.7.6 bullet

Panda has classes that represent underlying bullet objects, that basically wrap around it and integrate it with Panda classes and structures.

For instance, there's BulletRigidBodyNode, which is a class that extends a PandaNode and as such can be placed inside the panda scene graph. However, it stores a btRigidBody object from bullet, and exposes methods that are wrappers around that underlying Bullet object.

2.7.7 cftalk

connected frame protocol

2.7.8 chan

Animation channels. This defines the various kinds of AnimChannels that may be defined, as well as the MovingPart class which binds to the channels and plays the animation. This is a support library for char, as well as any other libraries that want to define objects whose values change over time.

2.7.9 char

-update me-

2.7.10 cocoadisplay

-update me-

2.7.11 collada

-update me-

2.7.12 collide

This package contains the classes that control and detect collisions

2.7.13 configfiles

This package contains the housekeeping and configuration files needed by things like attach, and emacs.

2.7.14 cull

This package contains the Cull Traverser. The cull traversal collects all state changes specified, and removes unneccesary state change requests.It also does all the depth sorting for proper alphaing.

2.7.15 device

Device drivers, such as mouse and keyboard, trackers, etc... The base class for using various device APIs is here.

2.7.16 dgraph

Defines and manages the data graph, which is the hierarchy of devices, tforms, and any other things which might have an input or an output and need to execute every frame.

2.7.17 display

Abstract display classes, including pipes, windows, channels, and display regions.

2.7.18 distort

-update me-

2.7.19 doc

Documentation Panda3D developers considered that doesn't fit in any of the packages. For contents please see the part 2 ("other") of this manual.

2.7.20 downloader

Tool to allow automatic download of files in the background.

2.7.21 downloadertools

-update me-

2.7.22 dxgsg9

Handles all communication with the DirectX backend, and manages state to minimize redundant state changes.

2.7.23 dxml

-update me-

2.7.24 egg

A.k.a. the "egg library", this reads, writes, and manipulates egg files. It knows nothing about the scene graph structure in the rest of the player; it lives in its own little egg world.

2.7.25 egg2pg

A.k.a. the "egg loader", this converts the egg structure read from the egg library, above, to a scene graph structure, suitable for rendering.

egg2pg reads egg file and converts it to a Panda scene graph. ie. in-memory structure of PandaNode etc.

I'm assuming "pg" stands for "panda graph".

Also, technically, the "egg" tree reads the .egg file into in-memory EggData structures, and egg2pg converts those to scene graph structures. When egg2pg converts that into scene graph structures egg files from memory get deleted. If you want to keep them around, you can use the lower-level interfaces yourself.

2.7.26 egldisplay

-update me-

2.7.27 event

Tools for throwing, handling and receiving events.

2.7.28 express

-update me-

2.7.29 ffmpeg

-update me-

2.7.30 framework

A simple, stupid framework around which to write a simple, stupid demo program. Handy for quickly writing programs that can open a window and display the OmniTriangle.

2.7.31 gles2gsg

-update me-

2.7.32 glesgsg

-update me-

2.7.33 glgsg

Handles all communication with the GL backend, and manages state to minimize redundant state changes.

2.7.34 glstuff

-update me-

2.7.35 glxdisplay

X windows display classes that replace Glut functionality.

2.7.36 gobj

Graphical non-scene-graph objects, such as textures and geometry primitives.

2.7.37 grutil

-update me-

2.7.38 gsgbase

Base GSG class defined to avoid cyclical dependency build.

2.7.39 iphone

-update me-

2.7.40 iphonedisplay

-update me-

2.7.41 linmath

Linear algebra library.

2.7.42 mathutil

Math utility functions, such as frustum and plane

2.7.43 movies

-update me-

2.7.44 nativenet

-update me-

2.7.45 net

Net connection classes

2.7.46 ode

-update me-

2.7.47 osxdisplay

-update me-

2.7.48 pandabase

-update me-

2.7.49 parametrics

-update me-

2.7.50 particlesystem

Tool for doing particle systems. Contains various kinds of particles, emiters, factories and renderers.

2.7.51 pgraph

-update me-

2.7.52 pgraphnodes

-update me-

2.7.53 pgui

-update me-

2.7.54 physics

Base classes for physical objects and forces. Also contains the physics manager class.

2.7.55 physx

-update me-

2.7.56 pipeline

-update me-

2.7.57 pnmimage

Reads and writes image files in various formats, by using the pnm and tiff libraries. PNMImage class manages reading and writing image files from disk.

One of the properties of PNMImage is that all images are laid out (almost) the same way in memory, regardless of their properties. This makes it very easy to write a class like PNMPainter, which can paint equally well on grayscale, grayscale/alpha, 24-bit, 32-bit, or 64-bit images.

2.7.58 pnmimagetypes

-update me-

2.7.59 pnmtext

-update me-

2.7.60 pstatclient

-update me-

2.7.61 putil

-update me-

2.7.62 recorder

-update me-

2.7.63 rocket

-update me-

2.7.64 skel

-update me-

2.7.65 speedtree

-update me-

2.7.66 testbed

C test programs, that primarily link with framework.

2.7.67 text

Package for generating renderable text using textured polygons.

2.7.68 tform

Data transforming objects that live in the data graph and convert raw data (as read from an input device, for instance) to something more useful.

2.7.69 tinydisplay

-update me-

2.7.70 vision

-update me-

2.7.71 vrpn

Defines the specific client code for interfacing to the VRPN API.

2.7.72 wgldisplay

Windows OpenGL specific display classes.

2.7.73 windisplay

-update me-

2.7.74 x11display

-update me-

2.8 pandatool

This tree contains various utility tools that are used to manipulate model files and convert models from other formats to Panda3D's .egg format (and vice versa).

2.9 samples

Contains samples to demonstrate how Panda3D works.

CHAPTER 3

File formats, specifications and similar

3.1 The GraphicsEngine

The GraphicsEngine is where it all begins. Thereis only one, global, GraphicsEngine in an application, and its job isto keep all of the pointers to your open windows and buffers, andalso to manage the task of doing the rendering, for all of the openwindows and buffers. Panda normally creates a GraphicsEngine for youat startup, which is available as `base.graphicsEngine`.There is usually no reason to create a second GraphicsEngine.

Note also that the following interfaces arestrictly for the advanced user. Normally, if you want

to create a new window or an offscreen buffer forrendering, you would just use the

base.openWindow()

or

window.makeTextureBuffer()

interfaces, which handle all of the details foryou automatically.

However, please continue reading if you want tounderstand in detail how Panda manages

windows and buffers, or if you have special needsthat are not addressed by the above

convenience methods.

3.1.1 GraphicsPipe

Each application will also need at least oneGraphicsPipe. The GraphicsPipe encapsulates the particular API usedto do rendering. For instance, there is one GraphicsPipe class forOpenGL rendering, and a different GraphicsPipe for DirectX. Althoughit is possible to create a GraphicsPipe of a specific type directly,normally Panda will create a default GraphicsPipe for you at startup,which is available in `base.pipe`.

The GraphicsPipe object isn't often used directly,except to create the individual GraphicsWindow and GraphicsBuffer-objects.

3.1.2 GraphicsWindow and GraphicsBuffer

The GraphicsWindow class is the class thatrepresents a single onscreen window for rendering. Panda normallyopens a default window for you at startup, which is available in`base.win`. You can create as manyadditional windows as you like. (Note, however, that some graphicsdrivers incur a performance penalty when multiple windows are opensimultaneously.)

Similarly, GraphicsBuffer is the class thatrepresents a hidden, offscreen buffer for rendering special offscreeneffects, such as render-to-texture. It is common for an applicationto have many offscreen buffers open at once.

Both classes inherit from the base classGraphicsOutput, which contains all of the code common to rendering toa window or offscreen buffer.

3.1.3 GraphicsStateGuardian

The GraphicsStateGuardian, or GSG for short,represents the actual graphics context. This class manages the actualnuts-and-bolts of drawing to a window; it manages the loading oftextures and vertex buffers into graphics memory, and has thefunctions for actually drawing triangles to the screen. (During theprocess of rendering the frame, the "graphics state"changes several times; the GSG gets its name from the fact that mostof its time is spent managing this graphics state.)

You would normally never call any methods on theGSG directly; Panda handles all of this for you, via theGraphicsEngine. This is important, because in some modes, the GSG mayoperate almost entirely in a separate thread from all of yourapplication code, and it is important not to interrupt that threadwhile it might be in the middle of drawing.

Each GraphicsOutput object keeps a pointer to theGSG that will be used to render that window or buffer. It is possiblefor each GraphicsOutput to have its own GSG, or it is possible toshare the same GSG between multiple different GraphicsOutputs.Normally, it is preferable to share GSG's, because this tends to bemore efficient for managing graphics resources.

Consider the following diagram to illustrate therelationship between these classes. This shows a typical applicationwith one window and two offscreen buffers:

| | GraphicsPipe | | | ————-: | :————: | ————— | | / | | | \ | | GraphicsOutput | GraphicsOutput | GraphicsOutput | | | | | | | | | GSG | GSG | GSG |

The GraphicsPipe was used to create each of thethree GraphicsOutputs, of which one is a GraphicsWindow, and theremaining two are GraphicsBuffers. Each GraphicsOutput has a pointerto the GSG that will be used for rendering. Finally, theGraphicsEngine is responsible for managing all of these objects.

In the above illustration, each window and bufferhas its own GSG, which is legal, although it's usually better toshare the same GSG across all open windows and buffers.

3.1.4 Renderinga frame

There is one key interface to rendering each frameof the graphics simulation:

base.graphicsEngine.renderFrame()

This method causes all open GraphicsWindows andGraphicsBuffers to render their contents for

the current frame. In order for Panda3D to renderanything, this method must be called once per frame. Normally, thisis done automatically by the task "igloop", which iscreated when you start

Panda.

3.1.5 Using a GraphicsEngine to create windows and buffers

In order to render in Panda3D, you need aGraphicsStateGuardian , and either a GraphicsWindow

(for rendering into a window) or a GraphicsBuffer(for rendering offscreen). You cannot create or destroy these objectsdirectly; instead, you must use interfaces on the GraphicsEngine tocreate them. Before you can create either of the above, you need tohave a GraphicsPipe, which specifies

the particular graphics API you want to use (e.g.OpenGL or DirectX). The default GraphicsPipe

specified in your Config.prc file has already beencreated at startup, and can be accessed by

base.pipe.

Now that you have a GraphicsPipe and aGraphicsEngine, you can create a

GraphicsStateGuardian object. This objectcorresponds to a single graphics context on the

graphics API, e.g. a single OpenGL context. (Thecontext owns all of the OpenGL or DirectX

objects like display lists, vertex buffers, andtexture objects.) You need to have at least one

GraphicsStateGuardian before you can create aGraphicsWindow:

myGsg=base.graphicsEngine.makeGsg(base.pipe)

Now that you have a GraphicsStateGuardian, you canuse it to create an onscreen

GraphicsWindow or an offscreen GraphicsBuffer:

base.graphicsEngine.makeWindow(gsg, name,sort)

base.graphicsEngine.makeBuffer(gsg, name,sort, xSize, ySize, wantTexture)

gsg is the GraphicsStateGuardian, name Is anarbitrary name you want to assign to the window/

buffer, and sort is an integer that determines theorder in which the windows/buffers will be

rendered. The buffer specific arguments xSize andySize decide the dimensions of the buffer, and

wantTexture should be set to True if you want toretrieve a texture from this buffer later on.

You can also use

graphicsEngine.makeParasite(host,name,sort,xSize,ySize)

where host is a GraphicsOutput object. It createsa buffer but it does not allocate room for itself. Instead it rendersto the framebuffer of host. It effectively has wantTexture set toTrue so you can

retrieve a texture from it later on. See TheGraphicsOutput class and Graphics Buffers and Windows

for more information.

myWindow=base.graphicsEngine.makeWindow(myGsg,”HelloWorld”,0)

myBuffer=base.graphicsEngine.makeBuffer(myGsg,”HiWorld”,0,800,600,True)

myParasite=base.graphicsEngine.makeBuffer(myBuffer,”Ima leech”,0,800,600)

Note: if youwant the buffers to be visible add show-buffers true to yourconfiguration file.

This causes the buffers to be opened as windowsinstead, which is useful while debugging.

3.1.6 Sharinggraphics contexts

It is possible to share the sameGraphicsStateGuardian among multiple different GraphicsWindows and/or GraphicsBuffers; if you dothis, then the graphics context will be used to render into each window one at a time. This isparticularly useful if the different windows will be rendering many of the same objects, sincethen the same texture objects and vertex buffers can be shared between different windows.

It is also possible to use a differentGraphicsStateGuardian for each different window. This means that if a particular texture is to berendered in each window, it will have to be loaded into graphics memory twice, once in each context,which may be wasteful. However, there are times when this may be what you want to do, forinstance if you have multiple graphics cards and you want to to render to both of themsimultaneously. (Note that the actual support for simultaneously rendering to multiple graphicscards is currently unfinished in Panda at the time of this writing, but the API has beendesigned with this future path in mind.)

3.1.7 Closingwindows

To close a specific window or buffer you useremoveWindow(window). To close all windows

removeAllWindows()

base.graphicsEngine.removeWindow(myWindow)

base.graphicsEngine.removeAllWindows()

More about GraphicsEngine

Here is some other useful functionality of theGraphicsEngine class.

GetNumWindows() Returns the number of windows andbuffers that this GraphicsEngine

object is managing. IsEmpty() Returns True if thisGraphicsEngine is not managing any windows or buffers. See API foradvanced functionality of GraphicsEngine and GraphicsStateGuardianclass.

3.2 ppython

ppython.exe is used for startingPanda3D. Basicly it is only duplicated copy of python.exe renamed soyou don't mix Panda's python with other python on your PATH

3.3 Panda Audio Documenation

3.3.1 AudioManagerand AudioSound

The AudioManager is a combination of a file cacheand a category

of sounds (e.g. sound effects, battle sounds, ormusic).

The first step is to decide which AudioManager touse and load it.

Once you have an AudioManager (e.g.effectsManager), a call to
get_sound() on that manager should getyou an AudioSound
(e.g. mySound = effectsManager.getSound(“bang”)).
After getting a sound from an AudioManager, youcan tell the sound
change its volume, loop, start time, play, stop,etc. There is no
need to involve the AudioManager explicitly inthese operations.
Simply delete the sound when you’re done with it. (The AudioSound
knows which AudioManager it is associated with,and will do the right
thing).
The audio system, provides an API for the rest ofPanda; and leaves a
lot of leaway to the low level sound system. Thisis good and bad.
On the good side: it’s easier to understand, andit allows for widely
varrying low level systems. On the bad side: itmay be harder to keep
the behavior consistent accross implementations(please try to keep
them consistent, when adding an implementation).

3.3.2 ExampleUsage

PythonExample:
effects=AudioManager.createAudioManager()
music=AudioManager.createAudioManager()
bang=effects.load(“bang”)
background=music.load(“background_music”)
background.play()
bang.play()
C++Example:
AudioManagereffects=AudioManager::create_AudioManager();
AudioManagermusic=AudioManager::create_AudioManager();
bang=effects.get_sound(“bang”);
background=music.get_sound(“background_music”);
background.play();
bang.play();

3.4 Codingstyle

Almostany programming language gives a considerable amount of freedom
tothe programmer in style conventions. Most programmers eventually

developa personal style and use it as they develop code.
Whenmultiple programmers are working together on one project, this canlead to multiple competing styles appearing throughout the code. Thisis not the end of the world, but it does tend to make the code moredifficult to read and maintain if common style conventions are notfollowed throughout.
Itis much better if all programmers can agree to use the same style whenworking together on the same body of work. It makes reading, understanding,and extending the existing code much easier and faster foreveryone involved. This is akin to all of the animators on a featurefilm training themselves to draw in one consistent style throughoutthe film.
Often,there is no strong reason to prefer one style over another, exceptthat at the end of the day just one must be chosen.
Thefollowing lays out the conventions that we have agreed to use withinPanda. Most of these conventions originated from an amalgamationof the different styles of the first three programmers to domajor development in Panda. The decisions were often arbitrary, andsome may object to the particular choices that were made. Althoughdiscussions about the ideal style for future work are still welcome,considerable code has already been written using these existingconventions, and the most important goal of this effort is consistency. Thus, changing the style at this point would require changingall of the existing code as well.
Notethat not all existing Panda code follows these conventions. This isunfortunate, but it in no way constitutes an argument in favor of abandoningthe conventions. Rather, it means we should make an effort tobring the older code into compliance as we have the opportunity.
Naturally,these conventions only apply to C and C++ code; a completelydifferent set of conventions has been established for Pythoncode for the project, and those conventions will not be discussedhere.
SPACING:
Notab characters should ever appear in a C++ file; we use only space charactersto achieve the appropriate indentation. Most editors can beconfigured to use spaces instead of tabs.

Weuse two-character indentation. That is, each nested level of
indentationis two characters further to the right than the enclosing
level.
Spacesshould generally surround operators, e.g. i + 1 instead of i+1.
Spacesfollow commas in a parameter list, and semicolons in a for
statement. Spaces are not placed immediately within parentheses;
e.g.foo(a, b) rather than foo(a,b).
Resistwriting lines of code that extend beyond 80 columns; instead,
folda long line when possible. Occasionally a line cannot be easily
foldedand remain readable, so this should be taken as more of a
suggestionthan a fixed rule, but most lines can easily be made to fit
within80 columns.
Commentsshould never extend beyond 80 columns, especially sentence or
paragraphcomments that appear on a line or lines by themselves.
Theseshould generally be wordwrapped within 72 columns. Any smart
editorcan do this easily.
CURLYBRACES:
Ingeneral, the opening curly brace for a block of text trails the
linethat introduces it, and the matching curly brace is on a line by
itself,lined up with the start of the introducing line, e.g.:
for(inti=0;i<10;i++){
...
}
Commandslike if, while, and for should always use curly braces, even
ifthey only enclose one command. That is, do this:
if(foo){
bar();
}
insteadof this:
if(foo)
bar();
NAMING:
Classnames are mixed case with an initial capital, e.g. MyNewClass.
Eachdifferent class (except nested classes, of course) is defined in
itsown header file named the same as the class itself, but with the
firstletter lowercase, e.g. myNewClass.h.

Typedefnames and other type names follow the same convention as class
names:mixed case with an initial capital. These need not be defined
intheir own header file, but usually typedef names will be scoped
withinsome enclosing class.
Localvariable names are lowercase with an underscore delimiting
words:my_value. Class data members, including static data members,
arethe same, but with a leading underscore: _my_data_member. We do
notuse Hungarian notation.
Classmethod names, as well as standalone function names, are
lowercasewith a delimiting underscore, just like local variable
names:my_function().
LANGUAGECONSTRUCTS:
PreferC++ constructs over equivalent C constructs when writing C++
code. For instance, use:
staticconstintbuffer_size=1024;
insteadof:
#defineBUFFER_SIZE1024
Resistusing brand-new C++ features that are not broadly supported by
compilers. One of our goals in Panda is ease of distribution to a
widerange of platforms; this goal is thwarted if only a few compilers
maybe used.
Moreexamples of the agreed coding style may be found in
panda/src/doc/sampleClass.* fileshould be also in appendix of this manual.

3.5 COLLISIONFLAGS

floor:for things that avatars can stand on
barrier:for things that avatars should collide against that are not floors
camera-collide:for things that the camera should avoid
trigger:for things (usually not barriers or floors) that should trigger an
eventwhen avatars intersect with them
sphere:for things that should have a collision sphere around them
tube:for things that should have a collision tube (cylinder) around them
NOTES
Thebarrier & camera-collide flags are typically used together.
Currently,the camera automatically pulls itself in front of anything

markedwith the camera-collide flag, so that the view of the avatar isn't blocked.
Thetrigger flag implies that avatars will not collide with the object; theycan move freely through it.
Thesphere & tube flags create a collision object that is as small as possiblewhile completely containing the original flagged geometry.

3.6 eggpalettize

3.6.1 HOWTO USE EGG_PALETTIZE

Theprogram egg-palettize is used when building models to optimize textureusage on all models before loading them into the show. It is capableof collecting together several different small texture images fromdifferent models and assembling them together onto the same image file,potentially reducing the total number of different texture imagesthat must be loaded and displayed at runtime from several thousandto several hundred or fewer.
Italso can be used to group together related textures that will be renderedat the same time (for instance, textures related to one neighborhood),and if nothing else, it can resize textures at build timeso that they may be painted at any arbitrary resolution according tothe artist's convenience, and then reduced to a suitable size for texturememory management (and to meet hardware requirements of having dimensionsthat are always a power of two).
Itis suggested that textures always be painted at high resolution and reducedusing egg-palettize, since this allows the show designer the greatestflexibility; if a decision is later made to increase the resolutionof a texture, this may be done by changing an option with egg-palettize,and does not require intervention of the artist.
Thebehavior of egg-palettize is largely controlled through a source filecalled textures.txa, which is usually found in the src/maps directorywithin the model tree. For a complete description of the syntaxof the textures.txa file, invoke the command egg-palettize -H.

3.6.2 GROUPINGEGG FILES

Muchof the contents of textures.txa involves assigning egg files to variousgroups; assigning two egg files to the same group indicates thatthey are associated in some way, and their texture images may be copiedtogether into the same palettes.

Thegroups are arbitrary and should be defined at the beginning of the eggfile with the syntax:

:groupgroupname

Wheregroupname is the name of the group. It is also possible to assigna directory name to a group. This is optional, but if done, it indicatesthat all of the textures for this group should be installed withinthe named subdirectory. The syntax is:

:groupgroupname dir dirname

Wheredirname is the name of the subdirectory. If you are generatinga phased download, the dirname should be one of phase_1, phase_2,etc., corresponding to the PHASE variable in the install_egg rule(see ppremake-models.txt).

Finally,it is possible to relate the different groups to each other hierachically. Doing this allows egg-palettize to assign textures to theminimal common subset between egg files that share the textures. Forinstance, if group beta and group gamma both depend on group alpha,a texture that is assigned to both groups beta and gamma can actuallybe placed on group alpha, to maximize sharing and minimize duplicationof palette space.

Yourelate two groups with the syntax:

:groupgroupname with basegroupname

Onceall the groups are defined, you can assign egg files to the variousgroups with a syntax like this:

model.egg: groupname

wheremodel.egg is the name of some egg model file built within the tree. You can explicitly group each egg file in this way, or you can usewildcards to group several at once, e.g.:

dog*.egg: dogs

Assigningan egg file to a group assigns all of the textures used by thategg file to that same group. If no other egg files reference the sametextures, those textures will be placed in one or more palette

imagesnamed after the group. If another egg file in a different groupalso references the textures, they will be assigned to the lowestgroup that both groups have in common (see relating the groups hierarchically,above), or copied into both palette images if the two groupshaving nothing in common.

3.6.3 CONTROLLINGTEXTURE PARAMETERS

Mostof the contents of the textures.txa is usually devoted to scaling thetexture images appropriately. This is usually done with a line somethinglike this:
texture.rgb: 64 64
wheretexture.rgb is the name of some texture image, and 64 64 is the sizein pixels it should be scaled to. It is also possible to specify thetarget size as a factor of the source size, e.g.:
bigtexture.rgb: 50%
specifiesthat the indicated texture should be scaled to 50% in each dimension(for a total reduction to 0.5 * 0.5 = 25% of the original area).
Asabove, you may group together multiple textures on the same line usingwildcards, e.g.:
wall*.rgb: 25%
Finally,you may include one or more optional keywords on the end of thetexture scaling line that indicate additional properties to apply tothe named textures. See egg-palettize -H for a complete list.
Someof the more common keywords are:
mipmap- Enables mipmaps for the texture.
linear- Disables mipmaps for the texture.
omit- Omits the texture from any palettes. The texture will still bescaled and installed, but it will not be combined with other textures. Normally you need to do this only when the texture will beapplied to some geometry at runtime. (Since palettizing a texturerequires adjusting the UV's of all the geometry that referencesit, a texture that is applied to geometry at runtime cannotbe palettized.)

3.6.4 RUNNINGEGG-PALETTIZE

Normally,egg-palettize is run automatically just by typing:
makeinstall
inthe model tree. It automatically reads the textures.txa file and generatesand installs the appropriate palette image files, as part of thewhole build process, and requires no further intervention from the user. See ppremake-models.txt for more information on setting up the modeltree.
Whenegg-palettize runs in the normal mode, it generates suboptimal palettes. Sometimes, for instance, a palette image is created with onlyone small texture in the corner, and the rest of it unused. This happensbecause egg-palettize is reserving space for future textures, andis ideal for development; but it is not suitable for shipping a finishedproduct. When you are ready to repack all of the palettes as optimallyas possible, run the command:
makeopt-pal
Thiscauses egg-palettize to reorganize all of the palette images to makethe best usage of texture memory. It will force a regeneration ofmost of the egg files in the model tree, so it can be a fairly involvedoperation.
Itis sometimes useful to analyze the results of egg-palettize. You cantype:
makepi >pi.txt
towrite a detailed report of every egg file, texture image, and generatedpalette image to the file pi.txt.
Finally,the command:
makepal-stats >stats.txt
willwrite a report to stats.txt of the estimated texture memory usage forall textures, broken down by group.

3.6.5 WHENTHINGS GO WRONG

Thewhole palettizing process is fairly complex; it's necessary for egg-palettizeto keep a record of the complete state of all egg files andall textures ever built in a particular model tree. It generally doesa good job of figuring out when things change and correctly

regeneratingthe necessary egg files and textures when needed, but sometimesit gets confused.
Thisis particularly likely to happen when you have reassigned some eggfiles from one group to another, or redefined the relationship betweendifferent groups. Sometimes egg-palettize appears to run correctly,but does not generate correct palettes. Other times egg-palettizewill fail with an assertion failure, or even a segment fault(general protection fault) when running egg-palettize, due to thiskind of confusion. This behavior should not happen, but it does happenevery once and a while.
Whenthis sort of thing happens, often the best thing to do is to invokethe command:
makeundo-pal
followedby:
makeinstall
Thisremoves all of the old palettization information, including the stateinformation cached from previous runs, and rebuilds a new set of palettesfrom scratch. It is a fairly heavy hammer, and may take some timeto complete, depending on the size of your model tree, but it almostalways clears up any problems related to egg-palettize.

3.7 THEPHILOSOPHY OF EGG FILES

THEPHILOSOPHY OF EGG FILES (vs. bam files)
Eggfiles are used by Panda3D to describe many properties of a scene: simplegeometry, including special effects and collision surfaces, charactersincluding skeletons, morphs, and multiple-joint assignments,and character animation tables.
Eggfiles are designed to be the lingua franca of model manipulation forPanda tools. A number of utilities are provided that read and writeegg files, for instance to convert to or from some other modelingformat, or to apply a transform or optimize vertices. The eggfile philosophy is to describe objects in an abstract way that facilitateseasy manipulation; thus, the format doesn't (usually) includeinformation such as polygon connectivity or triangle meshes.
Eggfiles are furthermore designed to be human-readable to help a

developerdiagnose (and sometimes repair) problems. Also, the egg syntaxis always intended to be backward compatible with previous versions,so that as the egg syntax is extended, old egg files will continueto remain valid.
Thisis a different philosophy than Panda's bam file format, which is abinary representation of a model and/or animation that is designed tobe loaded quickly and efficiently, and is strictly tied to a particularversion of Panda. The data in a bam file closely mirrors theactual Panda structures that are used for rendering. Although an effortis made to keep bam files backward compatible, occasionally thisis not possible and we must introduce a new bam file major version.
Whereegg files are used for model conversion and manipulation of models,bam files are strictly used for loading models into Panda. Althoughyou can load an egg file directly, a bam file will be loaded muchmore quickly.
Eggfiles might be generated by outside sources, and thus it makes senseto document its syntax here. Bam files, on the other hand, shouldonly be generated by Panda3D, usually by the program egg2bam. Theexact specification of the bam file format, if you should need it, isdocumented within the Panda3D code itself.

3.7.1 GENERALEGG SYNTAX

Eggfiles consist of a series of sequential and hierarchically-nested entries. In general, the syntax of each entry is:
<Entry-type>name{contents}
Wherethe name is optional (and in many cases, ignored anyway) and the syntaxof the contents is determined by the entry-type. The name (and stringsin general) may be either quoted with double quotes or unquoted. Newlines are treated like any other whitespace, and case is notsignificant. The angle brackets are literally a part of the entry keyword. (Square brackets and ellipses in this document are used to indicateoptional pieces, and are not literally part of the syntax.)
Thename field is always syntactically allowed between an entry keywordand its opening brace, even if it will be ignored. In the syntaxlines given below, the name is not shown if it will be ignored.

Commentsmay be delimited using either the C++-style // . . . or the
C-style/* . . . */. C comments do not nest. There is also a
entrytype, of the form:
{text}
entries are slightly different, in that tools which read and
writeegg files will preserve the text within entries, but
theymay not preserve comments delimited by // or /* */. Special
charactersand keywords within a entry should be quoted;
it'ssafest to quote the entire comment.

3.7.2 LOCALINFORMATION ENTRIES

Thesenodes contain information relevant to the current level of
nestingonly.
name{value}
<Char*>name{value}
Scalarscan appear in various contexts. They are always optional,
andspecify some attribute value relevant to the current context.
Thescalar name is the name of the attribute; different attribute
namesare meaningful in different contexts. The value is either a
numericor a (quoted or unquoted) string value; the interpretation
asa number or as a string depends on the nature of the named
attribute. Because of a syntactic accident with the way the egg
syntaxevolved, and <Char*> are lexically the same andboth
canrepresent either a string or a number. <Char*> is being phased
out;it is suggested that new egg files use only .

3.7.3 GLOBALINFORMATION ENTRIES

Thesenodes contain information relevant to the file as a whole. They
canbe nested along with geometry nodes, but this nesting is
irrelevantand the only significant placement rule is that they should
appearbefore they are referenced.
{string}
Thisentry indicates the coordinate system used in the egg file; the
eggloader will automatically make a conversion if necessary. The
followingstrings are valid: Y-up, Z-up, Y-up-right, Z-up-right,
Y-up-left,or Z-up-left. (Y-up is the same as Y-up-right, and Z-up

isthe same as Z-up-right.)
Byconvention, this entry should only appear at the beginning of the
file,although it is technically allowed anywhere. It is an error
toinclude more than one coordinate system entry in the same file.
Ifit is omitted, Y-up is assumed.
name{filename[scalars]}
Thisdescribes a texture file that can be referenced later with
{ name }. It is not necessary to make a entry for
eachtexture to be used; a texture may also be referenced directly
bythe geometry via an abbreviated inline entry, but a
separate entry is the only way to specify anything other
thanthe default texture attributes.
Ifthe filename is a relative path, the current egg file's directory
issearched first, and then the texture-path and model-path are
searched.
Thefollowing attributes are presently implemented for textures:
alpha-file{alpha-filename}
Ifthis scalar is present, the texture file's alpha channel is
readin from the named image file (which should contain a
grayscaleimage), and the two images are combined into a single
two-or four-channel image internally. This is useful for loading
alphachannels along with image file formats like JPEG that don't
traditionallysupport alpha channels.
alpha-file-channel{channel}
Thisdefines the channel that should be extracted from the file
namedby alpha-file to determine the alpha channel for the
resultingchannel. The default is 0, which means the grayscale
combinationof r, g, b. Otherwise, this should be the 1-based
channelnumber, for instance 1, 2, or 3 for r, g, or b,
respectively,or 4 for the alpha channel of a four-component
image.
format{format-definition}
Thisdefines the load format of the image file. The
format-definitionis one of:
RGBA,RGBM, RGBA12, RGBA8, RGBA4,
RGB,RGB12, RGB8, RGB5, RGB332,

LUMINANCE_ALPHA,
RED,GREEN, BLUE, ALPHA, LUMINANCE
Theformats whose names end in digits specifically request a particulartexel width. RGB12 and RGBA12 specify 48-bit texels withor without alpha; RGB8 and RGBA8 specify 32-bit texels, and RGB5and RGBA4 specify 16-bit texels. RGB332 specifies 8-bit texels.
Theremaining formats are generic and specify only the semantic meaningof the channels. The size of the texels is determined by thewidth of the components in the image file. RGBA is the most general;RGB is the same, but without any alpha channel. RGBM is likeRGBA, except that it requests only one bit of alpha, if the graphicscard can provide that, to leave more room for the RGB components,which is especially important for older 16-bit graphicscards (the "M" stands for "mask", as in acutout).
Thenumber of components of the image file should match the format specified;if it does not, the egg loader will attempt to provide theclosest match that does.
compression{compression-mode}
Definesan explicit control over the real-time compression mode appliedto the texture. The various options are:
DEFAULTOFF ON
FXT1DXT1 DXT2 DXT3 DXT4 DXT5
Thiscontrols the compression of the texture when it is loaded intographics memory, and has nothing to do with on-disk compressionsuch as JPEG. If this option is omitted or "DEFAULT", thenthe texture compression is controlled by the compressed-texturesconfig variable. If it is "OFF", texture compressionis explicitly off for this texture regardless of the settingof the config variable; if it is "ON", texture compression isexplicitly on, and a default compression algorithm supported by thedriver is selected. If any of the other options, it names the specificcompression algorithm to be used.
wrap{repeat-definition}
wrapu{repeat-definition}
wrapv{repeat-definition}

wrapw{repeat-definition}
Thisdefines the behavior of the texture image outside of the
normal(u,v) range 0.0 - 1.0. It is “REPEAT” to repeat the
textureto infinity, “CLAMP” not to. The wrapping behavior may be
specifiedindependently for each axis via “wrapu” and “wrapv”,or
itmay be specified for both simultaneously via “wrap”.
Althoughless often used, for 3-d textures wrapw may also be
specified,and it behaves similarly to wrapu and wrapv.
Thereare other legal values in addtional to REPEAT and CLAMP.
Thefull list is:
CLAMP
REPEAT
MIRROR
MIRROR_ONCE
BORDER_COLOR
borderr{red-value}
borderg{green-value}
borderb{blue-value}
bordera{alpha-value}
Thesedefine the “border color” of the texture, which is
particularlyimportant when one of the wrap modes, above, is
BORDER_COLOR.
type{texture-type}
Thismay be one of the following attributes:
1D
2D
3D
CUBE_MAP
Thedefault is “2D”, which specifies a normal, 2-d texture. If
anyof the other types is specified instead, a texture image of
thecorresponding type is loaded.
If3D or CUBE_MAP is specified, then a series of texture images
mustbe loaded to make up the complete texture; in this case, the
texturefilename is expected to include a sequence of one or more
hashmark (“#”) characters, which will be filled in with the
sequencenumber. The first image in the sequence must be numbered

0,and there must be no gaps in the sequence. In this case, a separatealpha-file designation is ignored; the alpha channel, if present,must be included in the same image with the color channel(s).

multiview { flag }

Ifthis flag is nonzero, the texture is loaded as a multiview texture. In this case, the filename must contain a hash mark ("#")as in the 3D or CUBE_MAP case, above, and the different imagesare loaded into the different views of the multiview textures. If the texture is already a cube map texture, the samehash sequence is used for both purposes: the first six images definethe first view, the next six images define the second view, andso on. If the texture is a 3-D texture, you must also specify num-views,below, to tell the loader how many images are loaded forviews, and how many are loaded for levels.

Amultiview texture is most often used to load stereo textures, wherea different image is presented to each eye viewing the texture,but other uses are possible, such as for texture animation.

num-views { count }

Thisis used only when loading a 3-D multiview texture. It specifieshow many different views the texture holds; the z height ofthe texture is then implicitly determined as (number of images) /(number of views).

read-mipmaps { flag }

Ifthis flag is nonzero, then pre-generated mipmap levels will be loadedalong with the texture. In this case, the filename should containa sequence of one or more hash mark ("#") characters, whichwill be filled in with the mipmap level number; the texture filenamethus determines a series of images, one for each mipmap level. The base texture image is mipmap level 0.

Ifthis flag is specified in conjunction with a 3D or cube map texture(as specified above), then the filename should contain two hashmark sequences, separated by a character such as an underscore,hyphen, or dot. The first sequence will be filled in withthe mipmap level index, and the second sequence will be

filledin with the 3D sequence or cube map face.

minfilter { filter-type }

magfilter { filter-type }

magfilteralpha { filter-type }

magfiltercolor { filter-type }

Thisspecifies the type of filter applied when minimizing or maximizing. Filter-type may be one of:

NEAREST

LINEAR

NEAREST_MIPMAP_NEAREST

LINEAR_MIPMAP_NEAREST

NEAREST_MIPMAP_LINEAR

LINEAR_MIPMAP_LINEAR

Thereare also some additional filter types that are supported for historicalreasons, but each of those additional types maps to one ofthe above. New egg files should use only the above filter types.

anisotropic-degree { degree }

Enablesanisotropic filtering for the texture, and specifies the degreeof filtering. If the degree is 0 or 1, anisotropic filteringis disabled. The default is disabled.

envtype { environment-type }

Thisspecifies the type of texture environment to create; i.e. it controlsthe way in which textures apply to models.

Environment-typemay be one of:

MODULATE

DECAL

BLEND

REPLACE

ADD

BLEND_COLOR_SCALE

MODULATE_GLOW

MODULATE_GLOSS

*NORMAL

*NORMAL_HEIGHT

*GLOW

*GLOSS

*HEIGHT

*SELECTOR

Thedefault environment type is MODULATE, which means the texture coloris multiplied with the base polygon (or vertex) color. This isthe most common texture environment by far. Other environment typesare more esoteric and are especially useful in the presence ofmultitexture. In particular, the types prefixed by an asterisk (*)require enabling Panda's automatic ShaderGenerator.

combine-rgb{combine-mode}

combine-alpha{combine-mode}

combine-rgb-source0{combine-source}

combine-rgb-operand0{combine-operand}

combine-rgb-source1{combine-source}

combine-rgb-operand1{combine-operand}

combine-rgb-source2{combine-source}

combine-rgb-operand2{combine-operand}

combine-alpha-source0{combine-source}

combine-alpha-operand0{combine-operand}

combine-alpha-source1{combine-source}

combine-alpha-operand1{combine-operand}

combine-alpha-source2{combine-source}

combine-alpha-operand2{combine-operand}

Theseoptions replace the envtype and specify the texture combiner mode,which is usually used for multitexturing. This specifies howthe texture combines with the base color and/or the other texturesapplied previously. You must specify both an rgb and an alphacombine mode. Some combine-modes use one source/operand pair,and some use all three; most use just two.

combine-modemay be one of:

REPLACE

MODULATE

ADD

ADD-SIGNED

INTERPOLATE

SUBTRACT

DOT3-RGB

DOT3-RGBA

combine-sourcemay be one of:

TEXTURE

CONSTANT

PRIMARY-COLOR

PREVIOUS

CONSTANT_COLOR_SCALE

LAST_SAVED_RESULT

combine-operandmay be one of:

SRC-COLOR

ONE-MINUS-SRC-COLOR

SRC-ALPHA

ONE-MINUS-SRC-ALPHA

Thedefault values if any of these are omitted are:

combine-rgb{modulate}

combine-alpha{modulate}

combine-rgb-source0{previous}

combine-rgb-operand0{src-color}

combine-rgb-source1{texture}

combine-rgb-operand1{src-color}

combine-rgb-source2{constant}

combine-rgb-operand2{src-alpha}

combine-alpha-source0{previous}

combine-alpha-operand0{src-alpha}

combine-alpha-source1{texture}

combine-alpha-operand1{src-alpha}

combine-alpha-source2{constant}

combine-alpha-operand2{src-alpha}

saved-result{flag}

Ifflag is nonzero, then it indicates that this particular texture

stagewill be supplied as the “last_saved_result” source for any

futuretexture stages.

tex-gen{mode}

Thisspecifies that texture coordinates for the primitives that

referencethis texture should be dynamically computed at runtime,

forinstance to apply a reflection map or some other effect. The validvalues for mode are:

EYE_SPHERE_MAP(or SPHERE_MAP)

WORLD_CUBE_MAP

EYE_CUBE_MAP(or CUBE_MAP)

WORLD_NORMAL

EYE_NORMAL

WORLD_POSITION

EYE_POSITION

POINT_SPRITE

stage-name { name }

Specifiesthe name of the TextureStage object that is created to renderthis texture. If this is omitted, a custom TextureStage is createdfor this texture if it is required (e.g. because some othermultitexturing parameter has been specified), or the system defaultTextureStage is used if multitexturing is not required.

priority { priority-value }

Specifiesan integer sort value to rank this texture in priority amongother textures that are applied to the same geometry. This isonly used to eliminate low-priority textures in case more texturesare requested for a particular piece of geometry than the graphicshardware can render.

blendr { red-value }

blendg { green-value }

blendb { blue-value }

blenda { alpha-value }

Specifiesa four-component color that is applied with the color in casethe envtype, above, is “blend”, or one of thecombine-sources is”constant”.

uv-name { name }

Specifiesthe name of the texture coordinates that are to be associatedwith this texture. If this is omitted, the default texturecoordinates are used.

rgb-scale { scale }

alpha-scale { scale }

Specifiesan additional scale factor that will scale the r, g, b

(ora) components after the texture has been applied. This is
onlyused when a combine mode is in effect. The only legal values
are1, 2, or 4.
alpha { alpha-type }
Thisspecifies whether and what type of transparency will be
performed. Alpha-type may be one of:
OFF
ON
BLEND
BLEND_NO_OCCLUDE
MS
MS_MASK
BINARY
DUAL
Ifalpha-type is OFF, it means not to enable transparency, even if
theimage contains an alpha channel or the format is RGBA. If
alpha-typeis ON, it means to enable the default transparency,
evenif the image filename does not contain an alpha channel. If
alpha-typeis any of the other options, it specifies the type of
transparencyto be enabled.
bin { bin-name }
Thisspecifies the bin name order of all polygons with this
textureapplied, in the absence of a bin name specified on the
polygonitself. See the description for bin under polygon
attributes.
draw-order { number }
Thisspecifies the fixed drawing order of all polygons with this
textureapplied, in the absence of a drawing order specified on
thepolygon itself. See the description for draw-order under
polygonattributes.
depth-offset { number }
depth-write { mode }
depth-test { mode }
Specifiesspecial depth buffer properties of all polygons with this
textureapplied. See the descriptions for the individual
attributesunder polygon attributes.

quality-level { quality }
Setsa hint to the renderer about the desired performance /
qualitytradeoff for this particular texture. This is most useful
forthe tinydisplay software renderer; for normal,
hardware-acceleratedrenderers, this may have little or no effect.
Thismay be one of:
DEFAULT
FASTEST
NORMAL
BEST
“Default”means to use whatever quality level is specified by the
globaltexture-quality-level config variable.
{ transform-definition }
Thisspecifies a 2-d or 3-d transformation that is applied to the
UV’sof a surface to generate the texture coordinates.
Thetransform syntax is similar to that for groups, except it may
defineeither a 2-d 3x3 matrix or a 3-d 4x4 matrix. (You should
usethe two-dimensional forms if the UV’s are two-dimensional, and
thethree-dimensional forms if the UV’s are three-dimensional.)
Atwo-dimensional transform may be any sequence of zero or more of
thefollowing. Transformations are post multiplied in the order
theyare encountered to produce a net transformation matrix.
Rotationsare counterclockwise about the origin in degrees.
Matrices,when specified explicitly, are row-major.
{ x y }
{ degrees }
{ x y }
{ s }
{
0001 02
1011 12
2021 22
}
Athree-dimensional transform may be any sequence of zero or more
ofthe following. See the description under , below, for
moreinformation.

{ x y z }

{ degrees }

{ degrees }

{ degrees }

{ degrees x y z }

{ x y z }

{ s }

{

0001 02 03

1011 12 13

2021 22 23

3031 32 33

}

name { [scalars] }

Thisdefines a set of material attributes that may later be

referencedwith { name }.

Thefollowing attributes may appear within the material block:

diffr { number }

diffg { number }

diffb { number }

diffa { number }

ambr { number }

ambg { number }

ambb { number }

amba { number }

emitr { number }

emitg { number }

emitb { number }

emita { number }

specr { number }

specg { number }

specb { number }

speca { number }

shininess { number }

local { flag }

Theseproperties collectively define a “material” that controlsthe

lightingeffects that are applied to a surface; a material is only ineffect in the presence of lighting.

Thefour color groups, diff*, amb*, emit*, and spec* specify the diffuse,ambient, emission, and specular components of the lighting equation,respectively. Any of them may be omitted; the omitted component(s)take their color from the native color of the primitive,otherwise the primitive color is replaced with the materialcolor.

Theshininess property controls the size of the specular highlight, andthe value ranges from 0 to 128. A larger value creates a smallerhighlight (creating the appearance of a shinier surface).

name { vertices }

Avertex pool is a set of vertices. All geometry is created by referringto vertices by number in a particular vertex pool. There maybe one or several vertex pools in an egg file, but all vertices thatmake up a single polygon must come from the same vertex pool.

Thebody of a entry is simply a list of one or more entries, as follows:

number { x [y [z [w]]] [attributes] }

A entry is only valid within a vertex pool definition.

Thenumber is the index by which this vertex will be referenced. Itis optional; if it is omitted, the vertices are implicitly numberedconsecutively beginning at one. If the number is supplied,the vertices need not be consecutive.

Normally,vertices are three-dimensional (with coordinates x, y, andz); however, in certain cases vertices may have fewer or more dimensions,up to four. This is particularly true of vertices usedas control vertices of NURBS curves and surfaces. If more coordinatesare supplied than needed, the extra coordinates are ignored;if fewer are supplied than needed, the missing coordinatesare assumed to be 0.

Thevertex's coordinates are always given in world space, regardlessof any transforms before the vertex pool or before the referencinggeometry. If the vertex is referenced by geometry undera transform, the egg loader will do an inverse transform to movethe vertex into the proper coordinate space without changing

itsposition in world space. One exception is geometry under an
node; in this case the vertex coordinates are given in
thespace of the node. (Another exception is a
;see below.)
Inneither case does it make a difference whether the vertex pool
isitself declared under a transform or an node. The
onlydeciding factor is whether the geometry that *uses* the
vertexpool appears under an node. It is possible for
asingle vertex to be interpreted in different coordinate spaces
bydifferent polygons.
Whileeach vertex must at least have a position, it may also have
acolor, normal, pair of UV coordinates, and/or a set of morph
offsets. Furthermore, the color, normal, and UV coordinates may
themselveshave morph offsets. Thus, the [attributes] in the
syntaxline above may be replaced with zero or more of the
followingentries:
target { x y z }
Thisspecifies the offset of this vertex for the named morph
target. See the "MORPH DESCRIPTION ENTRIES" header, below.
{ x y z [morph-list] }
Thisspecifies the surface normal of the vertex. If omitted, the
vertexwill have no normal. Normals may also be morphed;
morph-listhere is thus an optional list of entries,
similarto the above.
{ r g b a [morph-list] }
Thisspecifies the four-valued color of the vertex. Each
componentis in the range 0.0 to 1.0. A vertex color, if
specifiedfor all vertices of the polygon, overrides the polygon's
color. If neither color is given, the default is white
(11 1 1). The morph-list is an optional list of entries.
[name] { u v [w] [tangent] [binormal] [morph-list] }
Thisgives the texture coordinates of the vertex. This must be
specifiedif a texture is to be mapped onto this geometry.
Thetexture coordinates are usually two-dimensional, with two
componentvalues (u v), but they may also be three-dimensional,
withthree component values (u v w). (Arguably, it should be

called instead of in the three-dimensional case, but
it'snot.)
Asbefore, morph-list is an optional list of entries.
Unlikethe other kinds of attributes, there may be multiple sets
ofUV's on each vertex, each with a unique name; this provides
supportfor multitexturing. The name may be omitted to specify
thedefault UV's.
TheUV's also support an optional tangent and binormal. These
valuesare based on the vertex normal and the UV coordinates of
connectedvertices, and are used to render normal maps and similar
lightingeffects. They are defined within the entry because
theremay be a different set of tangents and binormals for each
differentUV coordinate set. If present, they have the expected
syntax:
[name] { u v [w] { x y z } { x y z }}
name { x y z w }
Thisspecifies some named per-vertex auxiliary data which is
importedfrom the egg file without further interpretation by
Panda. The auxiliary data is copied to the vertex data under a
columnwith the specified name. Presumably the data will have
meaningto custom code or a custom shader. Like named UV's, there
maybe multiple Aux entries for a given vertex, each with a
differentname.
name { vertices }
Adynamic vertex pool is similar to a vertex pool in most respects,
exceptthat each vertex might be animated by substituting in values
froma table. Also, the vertices defined within a
dynamicvertex pool are always given in local coordinates, instead
ofworld coordinates.
Thepresence of a dynamic vertex pool makes sense only within a
charactermodel, and a single dynamic vertex pool may not span
multiplecharacters. Each dynamic vertex pool creates a DynVerts
objectwithin the character by the same name; this name is used
laterwhen matching up the corresponding .
Atthe present time, the DynamicVertexPool is not implemented in
Panda3D.

3.7.4 GEOMETRYENTRIES

name {
[attributes]
{
indices
{ pool-name }
}
}
Apolygon consists of a sequence of vertices from a single vertex pool. Vertices are identified by pool-name and index number within thepool; indices is a list of vertex numbers within the given vertexpool. Vertices are listed in counterclockwise order.
Althoughthe vertices must all come from the same vertex pool, they mayhave been assigned to arbitrarily many different joints regardlessof joint connectivity (there is no "straddle-polygon" limitation). See Joints, below.
Thepolygon syntax is quite verbose, and there isn't any way to specifya set of attributes that applies to a group of polygons–the attributeslist must be repeated for each polygon. This is why egg filestend to be very large.
Thefollowing attributes may be specified for polygons:
{ texture-name }
Thisrefers to a named entry given earlier. It applies thegiven texture to the polygon. This requires that all the polygon'svertices have been assigned texture coordinates.
Thisattribute may be repeated multiple times to specify multitexture. In this case, each named texture is applied to the polygon,in the order specified.
{ filename }
Thisis another way to apply a texture to a polygon. The entry is defined "inline" to the polygon, instead of referringto a entry given earlier. There is no way to specifytexture attributes given this form.
There'sno advantage to this syntax for texture mapping. It's supportedonly because it's required by some older egg files.
{ material-name }

Thisapplies the material properties defined in the earlier
entry to the polygon.
{ x y z [morph-list] }
Thisdefines a polygon surface normal. The polygon normal will be
usedunless all vertices also have a normal. If no normal is
defined,none will be supplied. The polygon normal, like the
vertexnormal, may be morphed by specifying a series of
entries.
Thepolygon normal is used only for lighting and environment
mappingcalculations, and is not related to the implicit normal
calculatedfor CollisionPolygons.
{ r g b a [morph-list] }
Thisdefines the polygon's color, which will be used unless all
verticesalso have a color. If no color is defined, the default
iswhite (1 1 1 1). The color may be morphed with a series of
entries.
{ boolean-value }
Thisdefines whether the polygon will be rendered double-sided
(i.e.its back face will be visible). By default, this option is
disabled,and polygons are one-sided; specifying a nonzero value
disablesbackface culling for this particular polygon and allows
itto be viewed from either side.
bin { bin-name }
Itis sometimes important to control the order in which objects
arerendered, particularly when transparency is in use. In Panda,
thisis achieved via the use of named bins and, within certain
kindsof bins, sometimes an explicit draw-order is also used (see
below).
Inthe normal (state-sorting) mode, Panda renders its geometry by
firstgrouping into one or more named bins, and then rendering the
binsin a specified order. The programmer is free to define any
numberof bins, named whatever he/she desires.
Thisscalar specifies which bin this particular polygon is to be
renderedwithin. If no bin scalar is given, or if the name given
doesnot match any of the known bins, the polygon will be assigned
tothe default bin, which renders all opaque geometry sorted by

state,followed by all transparent geometry sorted back-to-front.
Seealso draw-order, below.
draw-order { number }
Thisworks in conjunction with bin, above, to further refine the
orderin which this polygon is drawn, relative to other geometry
inthe same bin. If (and only if) the bin type named in the bin
scalaris a CullBinFixed, this draw-order is used to define the
fixedorder that all geometry in the same will be rendered, from
smallernumbers to larger numbers.
Ifthe draw-order scalar is specified but no bin scalar is
specified,the default is a bin named “fixed”, which is a
CullBinFixedobject that always exists by default.
depth-offset { number }
Specifiesa special depth offset to be applied to the polygon.
Thismust be an integer value between 0 and 16 or so. The default
valueis 0; values larger than 0 will cause the polygon to appear
closerto the camera for purposes of evaluating the depth buffer.
Thiscan be a simple way to resolve Z-fighting between coplanar
polygons:with two or more coplanar polygons, the polygon with the
highestdepth-offset value will appear to be visible on top. Note
thatthis effect doesn’t necessarily work well when the polygons
areviewed from a steep angle.
depth-write { mode }
Specifiesthe mode for writing to the depth buffer. This may be
ONor OFF. The default is ON.
depth-test { mode }
Specifiesthe mode for testing against the depth buffer. This may
beON or OFF. The default is ON.
visibility { hidden | normal }
Ifthe visibility of a primitive is set to “hidden”, theprimitive
isnot generated as a normally visible primitive. If the
Config.prcvariable egg-suppress-hidden is set to true, the
primitiveis not converted at all; otherwise, it is converted as a
“stashed”node.
This,like the other rendering flags alpha, draw-order, and bin,
maybe specified at the group level, within the primitive level,

oreven within a texture.
name {
[attributes]
{
indices
{ pool-name }
}
}
Apatch is similar to a polygon, but it is a special primitive that
canonly be rendered with the use of a tessellation shader. Each
patchconsists of an arbitrary number of vertices; all patches with
thesame number of vertices are collected together into the same
GeomPatchesobject to be delivered to the shader in a single batch.
Itis then up to the shader to create the correct set of triangles
fromthe patch data.
Allof the attributes that are valid for Polygon, above, may also be
specifiedfor Patch.
name {
[attributes]
{
indices
{ pool-name }
}
}
APointLight is a set of single points. One point is drawn for each
vertexlisted in the . Normals, textures, and colors may
bespecified for PointLights, as well as draw-order, plus one
additionalattribute valid only for PointLights and Lines:
thick { number }
Thisspecifies the size of the PointLight (or the width of a
line),in pixels, when it is rendered. This may be a
floating-pointnumber, but the fractional part is meaningful only
whenantialiasing is in effect. The default is 1.0.
perspective { boolean-value }
Ifthis is specified, then the thickness, above, is to interpreted
asa size in 3-d spatial units, rather than a size in pixels, and

thepoint should be scaled according to its distance from the
viewernormally.
name {
[attributes]
{
indices
{ pool-name }
}
[componentattributes]
}
ALine is a connected set of line segments. The listed N vertices
definea series of N-1 line segments, drawn between vertex 0 and
vertex1, vertex 1 and vertex 2, etc. The line is not implicitly
closed;if you wish to represent a loop, you must repeat vertex 0 at
theend. As with a PointLight, normals, textures, colors,
draw-order,and the “thick” attribute are all valid (but not
“perspective”). Also, since a Line (with more than two vertices) is
madeup of multiple line segments, it may contain a number of
entries, to set a different color and/or normal for each
linesegment, as in TriangleStrip, below.
name {
[attributes]
{
indices
{ pool-name }
}
[componentattributes]
}
Atriangle strip is only rarely encountered in an egg file; it is
normallygenerated automatically only during load time, when
connectedtriangles are automatically meshed for loading, and even
thenit exists only momentarily. Since a triangle strip is a
renderingoptimization only and adds no useful scene information
overa loose collection of triangles, its usage is contrary to the
generalegg philosophy of representing a scene in the abstract.
Nevertheless,the syntax exists, primarily to allow inspection of

themeshing results when needed. You can also add custom
TriangleStripentries to force a particular mesh arrangement.
Atriangle strip is defined as a series of connected triangles.
Afterthe first three vertices, which define the first triangle,
eachnew vertex defines one additional triangle, by alternating up
anddown.
Itis possible for the individual triangles of a triangle strip to
havea separate normal and/or color. If so, a entry
shouldbe given for each so-modified triangle:
index {
{ r g b a [morph-list] }
{ x y z [morph-list] }
}
Whereindex ranges from 0 to the number of components defined by the
trianglestrip (less 1). Note that the component attribute list
mustalways follow the vertex list.
name {
[attributes]
{
indices
{ pool-name }
}
[componentattributes]
}
Atriangle fan is similar to a triangle strip, except all of the
connectedtriangles share the same vertex, which is the first
vertex. See , above.

3.7.5 PARAMETRICDESCRIPTION ENTRIES

Thefollowing entries define parametric curves and surfaces.
Generally,Panda supports these only in the abstract; they're not
geometryin the true sense but do exist in the scene graph and may
havespecific meaning to the application. However, Panda can create
visiblerepresentations of these parametrics to aid visualization.
Theseentries might also have meaning to external tools outside of an
interactivePanda session, such as egg-qtess, which can be used to

convertNURBS surfaces to polygons at different levels of resolution.
Ingeneral, dynamic attributes such as morphs and joint assignment are
legalfor the control vertices of the following parametrics, but Panda
itselfdoesn't support them and will always create static curves and
surfaces. External tools like egg-qtess, however, may respect them.
{
[attributes]
{order}
{knot-list}
{indices{pool-name}}
}
ANURBS curve is a general parametric curve. It is often used to
representa motion path, e.g. for a camera or an object.
Theorder is equal to the degree of the polynomial basis plus 1. It
mustbe an integer in the range [1,4].
Thenumber of vertices must be equal to the number of knots minus the
order.
Eachcontrol vertex of a NURBS is defined in homogeneous space with
fourcoordinates x y z w (to convert to 3-space, divide x, y, and z
byw). The last coordinate is always the homogeneous coordinate; if
onlythree coordinates are given, it specifies a curve in two
dimensionsplus a homogeneous coordinate (x y w).
Thefollowing attributes may be defined:
type{curve-type}
Thisdefines the semanting meaning of this curve, either XYZ, HPR,
orT. If the type is XYZ, the curve will automatically be
transformedbetween Y-up and Z-up if necessary; otherwise, it will
beleft alone.
subdiv{num-segments}
Ifthis scalar is given and nonzero, Panda will create a visible
representationof the curve when the scene is loaded. The number
representsthe number of line segments to draw to approximate the
curve.
{rgba[morph-list]}
Thisspecifies the color of the overall curve.
NURBScontrol vertices may also be given color and/or morph

attributes,but and entries do not apply to NURBS
vertices.
name{
[attributes]
{u-orderv-order}
<U-knots>{u-knot-list}
<V-knots>{v-knot-list}
{
indices
{pool-name}
}
}
ANURBS surface is an extension of a NURBS curve into two parametric
dimensions,u and v. NURBS surfaces may be given the same set of
attributesassigned to polygons, except for normals: ,
„ , and draw-order are all valid attributes
forNURBS. NURBS vertices, similarly, may be colored or morphed,
but and entries do not apply to NURBS vertices. The
attributesmay also include and entries; see
below.
Tohave Panda create a visualization of a NURBS surface, the
followingtwo attributes should be defined as well:
U-subdiv{u-num-segments}
V-subdiv{v-num-segments}
Thesedefine the number of subdivisions to make in the U and V
directionsto represent the surface. A uniform subdivision is
alwaysmade, and trim curves are not respected (though they will
bedrawn in if the trim curves themselves also have a subiv
parameter). This is only intended as a cheesy visualization.
Thesame sort of restrictions on order and knots applies to NURBS
surfacesas do to NURBS curves. The order and knot description may
bedifferent in each dimension.
Thesurface must have u-num * v-num vertices, where u-num is the
numberof u-knots minus the u-order, and v-num is the number of
v-knotsminus the v-order. All vertices must come from the same
vertexpool. The nth (zero-based) index number defines control

vertex(u, v) of the surface, where n = (v * u-num) + u. Thus, it
isthe u coordinate which changes faster.
Aswith the NURBS curve, each control vertex is defined in
homogeneousspace with four coordinates x y z w.
ANURBS may also contain curves on its surface. These are one or
morenested entries included with the attributes; these
curvesare defined in the two-dimensional parametric space of the
surface. Thus, these curve vertices should have only two dimensions
plusthe homogeneous coordinate: u v w. A curve-on-surface has no
intrinsicmeaning to the surface, unless it is defined within a
entry, below.
Finally,a NURBS may be trimmed by one or more trim curves. These
arespecial curves on the surface which exclude certain areas from
theNURBS surface definition. The inside is specified using two
rules:an odd winding rule that states that the inside consists of
allregions for which an infinite ray from any point in the region
willintersect the trim curve an odd number of times, and a curve
orientationrule that states that the inside consists of the regions
tothe left as the curve is traced.
Eachtrim curve contains one or more loops, and each loop contains
oneor more NURBS curves. The curves of a loop connect in a
head-to-tailfashion and must be explicitly closed.
Thetrim curve syntax is as follows:
{
{
{
{order}
{knot-list}
{indices{pool-name}}
}
[{...}...]
}
[{...}...]
}
Althoughthe egg syntax supports trim curves, there are at present
noegg processing tools that respect them. For instance, egg-qtess

ignorestrim curves and always tesselates the entire NURBS surface.

3.7.6 MORPHDESCRIPTION ENTRIES

Morphsare linear interpolations of attribute values at run time, accordingto values read from an animation table. In general, vertex positions,surface normals, texture coordinates, and colors may be morphed.

Amorph target is defined by giving a net morph offset for a series of vertexor polygon attributes; this offset is the value that will be addedto the attribute when the morph target has the value 1.0. At runtime, the morph target's value may be animated to any scalar value (butgenerally between 0.0 and 1.0); the corresponding fraction of the offsetis added to the attribute each frame.

Thereis no explicit morph target definition; a morph target exists solelyas the set of all offsets that share the same target name. The targetname may be any arbitrary string; like any name in an egg file, itshould be quoted if it contains special characters.

Thefollowing types of morph offsets may be defined, within their correspondingattribute entries:

target{xyz}

Aposition delta, valid within a entry or a entry. Thegiven offset vector, scaled by the morph target's value, is addedto the vertex or CV position each frame.

target{xyz}

Anormal delta, similar to the position delta, valid within a entry (for vertex or polygon normals). The given offset vector,scaled by the morph target's value, is added to the normal vectoreach frame. The resulting vector may not be automatically normalizedto unit length.

target{uv[w]}

Atexture-coordinate delta, valid within a entry (within a entry). The offset vector should be 2-valued if the enclosingUV is 2-valued, or 3-valued if the enclosing UV is 3-valued. The given offset vector, scaled by the morph target's value,is added to the vertex's texture coordinates each frame.

target{rgba}

Acolor delta, valid within an entry (for vertex orpolygon colors). The given 4-valued offset vector, scaled by themorph target'svalue, is added to the color value each frame.

3.7.7 GROUPINGENTRIES

name{group-body}
A node is the primary means of providing structure to the
eggfile. Groups can contain vertex pools and polygons, as well as
othergroups. The egg loader translates nodes directly into
PandaNodesin the scene graph (although the egg loader reserves the
rightto arbitrarily remove nodes that it deems unimportant–see the
flag, below to avoid this). In addition, the following
entriescan be given specifically within a node to specify
attributesof the group:

3.7.8 GROUPBINARY ATTRIBUTES

Theseattributes may be either on or off; they are off by default.
Theyare turned on by specifying a non-zero "boolean-value".
{boolean-value}
DCSstands for Dynamic Coordinate System. This indicates that
showcode will expect to be able to read the transform set on this
nodeat run time, and may need to modify the transform further.
Thisis a special case of , below.
{dcs-type}
Thisis another syntax for the flag. The dcs-type string
shouldbe one of either "local" or "net", whichspecifies the kind
ofpreserve_transform flag that will be set on the corresponding
ModelNode. If the string is "local", it indicates that the local
transformon this node (as well as the net transform) will not be
affectedby any flattening operation and will be preserved through
theentire model loading process. If the string is "net", then
onlythe net transform will be preserved; the local transform may
beadjusted in the event of a flatten operation.
{boolean-value}
Thisindicates that the show code might need a pointer to this
particulargroup. This creates a ModelNode at the corresponding
level,which is guaranteed not to be removed by any flatten

operation. However, its transform might still be changed, but see alsothe flag, above.

{boolean-value}

Thisindicates that this group begins an animated character. A Characternode, which is the fundamental animatable object of Panda'shigh-level Actor class, will be created for this group. Thisflag should always be present within the entry at the topof any hierarchy of 's and/or geometry with morphed vertices;joints and morphs appearing outside of a hierarchy identifiedwith a flag are undefined.

{boolean-value}

Thisattribute indicates that the child nodes of this group representa series of animation frames that should be consecutivelydisplayed. In the absence of an "fps" scalar for thegroup (see below), the egg loader creates a SwitchNode, and it theresponsibility of the show code to perform the switching. If anfps scalar is defined and is nonzero, the egg loader creates a SequenceNodeinstead, which automatically cycles through its children.

3.7.9 GROUPSCALARS

fps{frame-rate}

Thisspecifies the rate of animation for a SequenceNode (created whenthe Switch flag is specified, see above). A value of zero indicatesa SwitchNode should be created instead.

bin{bin-name}

Thisspecifies the bin name for all polygons at or below this node thatdo not explicitly set their own bin. See the description of binfor geometry attributes, above.

draw-order{number}

Thisspecifies the drawing order for all polygons at or below this nodethat do not explicitly set their own drawing order. See the descriptionof draw-order for geometry attributes, above.

depth-offset{number}

depth-write{mode}

depth-test{mode}

Specifiesspecial depth buffer properties of all polygons at or belowthis node that do not override this. See the descriptions forthe individual attributes under polygon attributes.

visibility{hidden|normal}

Ifthe visibility of a group is set to "hidden", theprimitives nestedwithin that group are not generated as a normally visible primitive. If the Config.prc variable egg-suppress-hidden is set totrue, the primitives are not converted at all; otherwise, they areconverted as a "stashed" node.

decal{boolean-value}

Ifthis is present and boolean-value is non-zero, it indicates thatthe geometry *below* this level is coplanar with the geometry *at*this level, and the geometry below is to be drawn as a decal ontothe geometry at this level. This means the geometry below thislevel will be rendered "on top of" this geometry, butwithout theZ-fighting artifacts one might expect without the use of the decalflag.

decalbase{boolean-value}

Thiscan optionally be used with the "decal" scalar, above. If present,it should be applied to a sibling of one or more nodes withthe "decal" scalar on. It indicates which of the sibling nodesshould be treated as the base of the decal. In the absence ofthis scalar, the parent of all decal nodes is used as the decal base. This scalar is useful when the modeling package is unable toparent geometry nodes to other geometry nodes.

collide-mask{value}

from-collide-mask{value}

into-collide-mask{value}

Setsthe CollideMasks on the collision nodes and geometry nodes createdat or below this group to the indicated values. These arebits that indicate which objects can collide with which otherobjects. Setting "collide-mask" is equivalent to setting both"from-collide-mask" and "into-collide-mask" tothe same value.

Thevalue may be an ordinary decimal integer, or a hex number in theform 0x000, or a binary number in the form 0b000.

blend{mode}
Specifiesthat a special blend mode should be applied geometry at thislevel and below. The available options are none, add, subtract,inv-subtract, min, and max. See ColorBlendAttrib.
blendop-a{mode}
blendop-b{mode}
Ifblend mode, above, is not none, this specifies the A and B operandsto the blend equation. Common options are zero, one, incoming-color,one-minus-incoming-color. See ColorBlendAttrib forthe complete list of available options. The default is “one”.
blendr{red-value}
blendg{green-value}
blendb{blue-value}
blenda{alpha-value}
Ifblend mode, above, is not none, and one of the blend operands isconstant-color or a related option, this defines the constant colorthat will be used.
occluder{boolean-value}
Thismakes the first (or only) polygon within this group node into anoccluder. The polygon must have exactly four vertices. An occluderpolygon is invisible. When the occluder is activated withmodel.set_occluder(occluder), objects that are behind the occluderwill not be drawn. This can be a useful rendering optimizationfor complex scenes, but should not be overused or performancecan suffer.

3.7.10 OTHERGROUP ATTRIBUTES

{type}
Thisentry indicates that all geometry defined at or below this grouplevel is part of a billboard that will rotate to face the camera. Type is either “axis” or “point”, describing thetype of rotation.
Billboardsrotate about their local axis. In the case of a Y-up file,the billboards rotate about the Y axis; in a Z-up file, they rotateabout the Z axis. Point-rotation billboards rotate about theorigin.

Thereis an implicit around billboard geometry. This meansthat the geometry within a billboard is not specified in worldcoordinates, but in the local billboard space. Thus, a vertexdrawn at point 0,0,0 will appear to be at the pivot point ofthe billboard, not at the origin of the scene.

{

{

inout[fade]{xyz}

}

}

Thesubtree beginning at this node and below represents a single levelof detail for a particular model. Sibling nodes represent theadditional levels of detail. The geometry at this node will bevisible when the point (x, y, z) is closer than "in" units,but furtherthan "out" units, from the camera. "fade" ispresently ignored.

key{value}

Thisattribute defines the indicated tag (as a key/value pair), retrievablevia NodePath::get_tag() and related interfaces, on thisnode.

3.7.11 name { type [flags] }

Thisentry indicates that geometry defined at this group level is actuallyan invisible collision surface, and is not true geometry. Thegeometry is used to define the extents of the collision surface. If there is no geometry defined at this level, then a childis searched for with the same collision type specified, and itsgeometry is used to define the extent of the collision surface(unless the "descend" flag is given; see below).

Validtypes so far are:

Plane

Thegeometry represents an infinite plane. The first polygon foundin the group will define the plane.

Polygon

Thegeometry represents a single polygon. The first polygon is used.

Polyset

Thegeometry represents a complex shape made up of several polygons. This collision type should not be overused, as it providesthe least optimization benefit.

Sphere

Thegeometry represents a sphere. The vertices in the group are averagedtogether to determine the sphere's center and radius.

Box

Thegeometry represents a box. The smalles axis-alligned box thatwill fit around the vertices is used.

InvSphere

Thegeometry represents an inverse sphere. This is the same as Sphere,with the normal inverted, so that the solid part of an inversesphere is the entire world outside of it. Note that an inversesphere is in infinitely large solid with a finite hole cutinto it.

Tube

Thegeometry represents a tube. This is a cylinder-like shape withhemispherical endcaps; it is sometimes called a capsule or alozenge in other packages. The smallest tube shape that will fitaround the vertices is used.

Theflags may be any zero or more of:

event

Throwsthe name of the entry, or the name of the surfaceif the entry has no name, as an event whenever anavatar strikes the solid. This is the default if the entry has a name.

intangible

Ratherthan being a solid collision surface, the defined surface representsa boundary. The name of the surface will be thrown asan event when an avatar crosses into the interior, and name-outwill be thrown when an avater exits.

descend

Insteadof creating only one collision object of the given type, eachgroup descended from this node that contains geometry will definea new collision object of the given type. The event

name,if any, will also be inherited from the top node and sharedamong all the collision objects.

keep

Don'tdiscard the visible geometry after using it to define a collisionsurface; create both an invisible collision surface andthe visible geometry.

level

Storesa special effective normal with the collision solid that pointsup, regardless of the actual shape or orientation of the solid. This can be used to allow an avatar to stand on a slopingsurface without having a tendency to slide downward.

3.7.12 {type}

Thisis a short form to indicate one of several pre-canned sets of attributes. Type may be any word, and a Config definition will be searchedfor by the name "egg-object-type-word", where "word"is thetype word. This definition may contain any arbitrary egg syntaxto be parsed in at this group level.

Anumber of predefined ObjectType definitions are provided:

barrier

Thisis equivalent to { Polyset descend }. The geometrydefined at this root and below defines an invisible collisionsolid.

trigger

Thisis equivalent to { Polyset descend intangible }. Thegeometry defined at this root and below defines an invisible triggersurface.

sphere

Equivalentto { Sphere descend }. The geometry is replacedwith the smallest collision sphere that will enclose it. Typically you model a sphere in polygons and put this flag onit to create a collision sphere of the same size.

tube

Equivalentto { Tube descend }. As in sphere, above, butthe geometry is replaced with a collision tube (a capsule). Typicallyyou will model a capsule or a cylinder in polygons.

bubble

Equivalentto { Sphere keep descend }. A collision bubbleis placed around the geometry, which is otherwise unchanged.

ghost

Equivalentto collide-mask { 0 }. It means that the geometrybeginning at this node and below should never be collidedwith–characters will pass through it.

backstage

Thishas no equivalent; it is treated as a special case. It meansthat the geometry at this node and below should not be translated. This will normally be used on scale references and othermodeling tools.

Theremay also be additional predefined egg object types not listedhere; see the *.pp files that are installed into the etc directoryfor a complete list.

{transform-definition}

Thisspecifies a matrix transform at this group level. This definesa local coordinate space for this group and its descendents. Vertices are still specified in world coordinates (ina vertex pool), but any geometry assigned to this group will beinverse transformed to move its vertices to the local space. Thetransform definition may be any sequence of zero or more of thefollowing. Transformations are post multiplied in the order theyare encountered to produce a net transformation matrix. Rotationsare defined as a counterclockwise angle in degrees about aparticular axis, either implicit (about the x, y, or z axis), or arbitrary. Matrices, when specified explicitly, are row-major.

{xyz}

{degrees}

{degrees}

{degrees}

{degreesxyz}

{xyz}

{s}

{

00010203

10111213

20212223

30313233

}

Notethat the block should always define a 3-d transformwhen it appears within the body of a , while it maydefine either a 2-d or a 3-d transform when it appears within thebody of a . See , above.

{transform-definition}

Thisdefines an optional default pose transform, which might be a differenttransform from that defined by the entry, above. This makes sense only for a . See the description,below.

Thedefault pose transform defines the transform the joint will maintainin the absence of any animation being applied. This is differentfrom the entry, which defines the coordinate spacethe joint must have in order to keep its vertices in their (globalspace) position as given in the egg file. If this is differentfrom the entry, the joint's vertices will *not*be in their egg file position at initial load. If there is no entry for a particular joint, the implicit default-posetransform is the same as the entry.

Normally,the entry, if any, is created by the egg-optchar-defpose option. Most other software has little reasonto specify an explicit .

{indices{pool-name}}

Thismoves geometry created from the named vertices into the currentgroup, regardless of the group in which the geometry is actuallydefined. See the description, below.

{

fps{float-value}

num-frames{integer-value}

}

Oneor more AnimPreload entries may appear within the that containsa entry, indicating an animated character (see

above). These AnimPreload entries record the minimal preloaded animationdata required in order to support asynchronous animation binding. These entries are typically generated by the egg-optchar programwith the -preload option, and are used by the Actor code whenallow-async-bind is True (the default).

name{group-body}

An node is exactly like a node, exceptthat verticesreferenced by geometry created under the node arenot assumed to be given in world coordinates, but are instead givenin the local space of the node itself (including anytransforms given to the node).

Inother words, geometry under an node is defined in localcoordinates. In principle, similar geometry can be created underseveral different nodes, and thus can be positioned ina different place in the scene each instance. This doesn't necessarilyimply the use of shared geometry in the Panda3D scene graph,but see the syntax, below.

Thisis particularly useful in conjunction with a entry, to loadexternal file references at places other than the origin.

Aspecial syntax of entries does actually createshared geometryin the scene graph. The syntax is:

name{
{group-name}
[{group-name}...]
}

Inthis case, the referenced group name will appear as a duplicate instancein this part of the tree. Local transforms can be applied andare relative to the referencing group's transform. The referencedgroup must appear preceding this point in the egg file, andit will also be a part of the scene in the point at which it firstappears. The referenced group may be either a or an of its own; usually, it is a nested within an earlier entry.

name{[transform][ref-list][joint-list]}

Ajoint is a highly specialized kind of grouping node. A tree of jointsis used to specify the skeletal structure of an animated

character.
Ajoint may only contain one of three things. It may contain a
entry, as above, which defines the joint's unanimated
(rest)position; it may contain lists of assigned vertices or CV's;
andit may contain other joints.
Atree of nodes only makes sense within a character
definition,which is created by applying the flag to a group.
See, above.
Thevertex assignment is crucial. This is how the geometry of a
characteris made to move with the joints. The character's geometry
isactually defined outside the joint tree, and each vertex must be
assignedto one or more joints within the tree.
Thisis done with zero or more entries per joint, as the
following:
{indices[membership{m}]{pool-name}}
Thisis syntactically similar to the way vertices are assigned to
polygons. Each entry can assign vertices from only one
vertexpool (but there may be many entries per joint).
Indicesis a list of vertex numbers from the specied vertex pool, in
anarbitrary order.
Themembership scalar is optional. If specified, it is a value
between0.0 and 1.0 that indicates the fraction of dominance this
jointhas over the vertices. This is used to implement
soft-skinning,so that each vertex may have partial ownership in
severaljoints.
The entry may also be given to ordinary nodes.
Inthis case, it treats the geometry as if it was parented under the
groupin the first place. Non-total membership assignments are
meaningless.
name{table-list}
Atable is a set of animated values for joints. A tree of tables
withthe same structure as the corresponding tree of joints must be
definedfor each character to be animated. Such a tree is placed
undera node, which provides a handle within Panda to the
treeas a whole.
Bundlesmay only contain tables; tables may contain more tables,

bundles,or any one of the following (entries are optional,
anddefault as shown):
<S$Anim>name{
fps{24}
{values}
}
Thisis a table of scalar values, one per frame. This may be
appliedto a morph slider, for instance.
<Xfm$Anim>name{
fps{24}
order{srpht}
contents{ijkabcrphxyz}
{values}
}
Thisis a table of matrix transforms, one per frame, such as may
beapplied to a joint. The “contents” string consists of asubset
ofthe letters “ijkabcrphxyz”, where each letter correspondsto a
columnof the table; is a list of numbers of length(contents)
*num_frames. Each letter of the contents string corresponds to a
typeof transformation:
i,j, k - scale in x, y, z directions, respectively
a,b, c - shear in xy, xz, and yz planes, respectively
r,p, h - rotate by roll, pitch, heading
x,y, z - translate in x, y, z directions
Thenet transformation matrix specified by each row of the table
isdefined as the net effect of each of the individual columns’
transform,according to the corresponding letter in the contents
string. The order the transforms are applied is defined by the
orderstring:
s - all scale and shear transforms
r,p, h - individual rotate transforms
t - all translation transforms
<Xfm$Anim_S$>name{
fps{24}
order{srpht}
<S$Anim>i{. . . }

<S$Anim>j{ . . . }

. . .

}

Thisis a variant on the <Xfm$Anim> entry, where each column of thetable is entered as a separate <S$Anim> table. This syntax reflectsan attempt to simplify the description by not requiring repetitionof values for columns that did not change value during ananimation sequence.

name{

width{table-width}

fps{24}

{values}

}

Thisis a table of vertex positions, normals, texture coordinates, orcolors. These values will be subsituted at runtime for the correspondingvalues in a . The name of the tableshould be “coords”, “norms”, “texCoords”,or “colors”, accordingto the type of values defined. The number table-width isthe number of floats in each row of the table. In the case of acoords or norms table, this must be 3 times the number of verticesin the corresponding dynamic vertex pool. (For texCoords andcolors, this number must be 2 times and 4 times, respectively.)

3.7.13 MISCELLANEOUS

{filename}

Thisincludes a copy of the referenced egg file at the current point. This is usually placed under an node, so that the currenttransform will apply to the geometry in the external file. Theextension “.egg” is implied if it is omitted.

Aswith texture filenames, the filename may be a relative path, in whichcase the current egg file’s directory is searched first, and thenthe model-path is searched.

3.7.14 ANIMATIONSTRUCTURE

Unanimatedmodels may be defined in egg files without much regard to anyparticular structure, so long as named entries like VertexPools andTextures appear before they are referenced.

However,a certain rigid structural convention must be followed in orderto properly define an animated skeleton-morph model and its associatedanimation data.

Thestructure for an animated model should resemble the following:

CHARACTER_NAME{
{1}
JOINT_A{
{...}
{...}
{...}
JOINT_B{
{...}
{...}
{...}
}
JOINT_C{
{...}
{...}
{...}
}
...
}
}

The flag is necessary to indicate that this group begins an animatedmodel description. Without the flag, joints will be treatedas ordinary groups, and morphs will be ignored.

Inthe above, UPPERCASE NAMES represent an arbitrary name that you maychoose. The name of the enclosing group, CHARACTER_NAME, is takenas the name of the animated model. It should generally match thebundle name in the associated animation tables.

Withinthe group, you may define an arbitrary hierarchy of entries. There may be as many entries as you like,

andthey may have any nesting complexity you like. There may be
eitherone root , or multiple roots. However, you must
alwaysinclude at least one , even if your animation consists
entirelyof morphs.
Polygonsmay be directly attached to joints by enclosing them within
the group, perhaps with additional nesting entries,
asillustrated above. This will result in the polygon's vertices
beinghard-assigned to the joint it appears within. Alternatively,
youdeclare the polygons elsewhere in the egg file, and use
entries within the group to associate the
verticeswith the joints. This is the more common approach, since
itallows for soft-assignment of vertices to multiple joints.
Itis not necessary for every joint to have vertices at all. Every
jointshould include a transform entry, however, which defines the
initial,resting transform of the joint (but see also ,
above). If a transform is omitted, the identity transform is
assumed.
Someof the vertex definitions may include morph entries, as
describedin MORPH DESCRIPTION ENTRIES, above. These are meaningful
onlyfor vertices that are assigned, either implicitly or
explicitly,to at least one joint.
Youmay have multiple versions of a particular animated model–for
instance,multiple different LOD's, or multiple different clothing
options. Normally each different version is stored in a different
eggfile, but it is also possible to include multiple versions
withinthe same egg file. If the different versions are intended to
playthe same animations, they should all have the same
CHARACTER_NAME,and their joint hierarchies should exactly match in
structureand names.
Thestructure for an animation table should resemble the following:
CHARACTER_NAME{
JOINT_A{
<Xfm$Anim_S$>xform{
<Char*>order{sphrt}
fps{24}
<S$Anim>x{00101020. . . }

<S$Anim>y{00000...}
<S$Anim>z{2020202020...}
}
JOINT_B{
<Xfm$Anim_S$>xform{
<Char*>order{sphrt}
fps{24}
<S$Anim>x{...}
<S$Anim>y{...}
<S$Anim>z{...}
}
}
JOINT_C{
<Xfm$Anim_S$>xform{
<Char*>order{sphrt}
fps{24}
<S$Anim>x{...}
<S$Anim>y{...}
<S$Anim>z{...}
}
}
}
}
<S$Anim>MORPH_A{
fps{24}
{0000.10.20.31...}
}
<S$Anim>MORPH_B{
fps{24}
{...}
}
<S$Anim>MORPH_C{
fps{24}
{...}
}
}

}

}

The entry begins an animation table description. This
entrymust have at least one child: a named “”
(thisname is a literal keyword and must be present). The children
ofthis entry should be a hierarchy of additional
entries,one for each joint in the model. The joint structure and
namesdefined by the hierarchy should exactly match the
jointstructure and names defined by the hierarchy in the
correspondingmodel.

Each that corresponds to a joint should have one child, an
<Xfm$Anim_S$>entry named “xform” (this name is a literal keyword
andmust be present). Within this entry, there is a series of up to
twelve<S$Anim> entries, each with a one-letter name like “x”,”y”,
or”z”, which define the per-frame x, y, z position of the
correspondingjoint. There is one numeric entry for each frame, and
allframes represent the same length of time. You can also define
rotation,scale, and shear. See the full description of
<Xfm$Anim_S$>,above.

Withina particular animation bundle, all of the various components
throughoutthe various should define the same number of
frames,with the exception that if any of them define exactly one
framevalue, that value is understood to be replicated the
appropriatenumber of times to match the number of frames defined by
othercomponents.

(Notethat you may alternatively define an animation table with an
<Xfm$Anim>entry, which defines all of the individual components in
onebig matrix instead of individually. See the full description
above.)

Eachjoint defines its frame rate independently, with an “fps”
scalar. This determines the number of frames per second for the
framedata within this table. Typically, all joints have the same
framerate, but it is possible for different joints to animate at
differentspeeds.

Eachjoint also defines the order in which its components should be
composedto determine the complete transform matrix, with an “order”

scalar. This is described in more detail above.
Ifany of the vertices in the model have morphs, the top-level
alsoa literal keyword). This table in turn contains a list of
<S$Anim>entries, one for each named morph description. Each table
containsa list of numeric values, one per frame; as with the joint
data,there should be the same number of numeric values in all
tables,with the exception that just one value is understood to mean
holdthat value through the entire animation.
The"morph" table may be omitted if there are no morphs definedin
themodel.
Thereshould be a separate definition for each different
animation. The name should match the CHARACTER_NAME used
forthe model, above. Typically each bundle is stored in a separate
eggfile, but it is also possible to store multiple different
animationbundles within the same egg file. If you do this, you may
violatethe CHARACTER_NAME rule, and give each bundle a different
name;this will become the name of the animation in the Actor
interface.
Althoughanimations and models are typically stored in separate egg
files,it is possible to store them together in one large egg file.
TheActor interface will then make available all of the animations
itfinds within the egg file, by bundle name.

3.8 HOWTO CONTROL RENDER ORDER

Inmost simple scenes, you can naively attach geometry to the scene
graphand let Panda decide the order in which objects should be
rendered. Generally, it will do a good enough job, but there are
occasionsin which it is necessary to step in and take control of the
process.
Todo this well, you need to understand the implications of render
order. In a typical OpenGL- or DirectX-style Z-buffered system, the
orderin which primitives are sent to the graphics hardware is
theoreticallyunimportant, but in practice there are many important
reasonsfor rendering one object before another.
Firstly,state sorting is one important optimization. This means

choosingto render things that have similar state (texture, color, etc.)all at the same time, to minimize the number of times the graphicshardware has to be told to change state in a particular frame. This sort of optimization is particularly important for very high-endgraphics hardware, which achieves its advertised theoretical polygonthroughput only in the absence of any state changes; for many suchadvanced cards, each state change request will completely flush theregister cache and force a restart of the pipeline.
Secondly,some hardware has a different optimization requirement, and maybenefit from drawing nearer things before farther things, so that theZ-buffer algorithm can effectively short-circuit some of the advancedshading features in the graphics card for pixels that would beobscured anyway. This sort of hardware will draw things fastest whenthe scene is sorted in order from the nearest object to the farthestobject, or "front-to-back" ordering.
Finally,regardless of the rendering optimizations described above, a particularsorting order is required to render transparency properly (inthe absence of the specialized transparency support that only a fewgraphics cards provide). Transparent and semitransparent objects arenormally rendered by blending their semitransparent parts with whathas already been drawn to the framebuffer, which means that it is importantthat everything that will appear behind a semitransparent objectmust have already been drawn before the semitransparent parts ofthe occluding object is drawn. This implies that all semitransparentobjects must be drawn in order from farthest away to nearest,or in "back-to-front" ordering, and furthermore that the opaqueobjects should all be drawn before any of the semitransparent objects.
Pandaachieves these sometimes conflicting sorting requirements throughthe use of bins.

3.8.1 CULLBINS

TheCullBinManager is a global object that maintains a list of all of thecull bins in the world, and their properties. Initially, there arefive default bins, and they will be rendered in the following order:

BinName Sort Type

"background" 10 BT_fixed
"opaque" 20 BT_state_sorted
"transparent" 30 BT_back_to_front
"fixed" 40 BT_fixed
"unsorted" 50 BT_unsorted

WhenPanda traverses the scene graph each frame for rendering, it assignseach Geom it encounters into one of the bins defined in the CullBinManager. (The above lists only the default bins. Additional binsmay be created as needed, using either the CullBinManager::add_bin()method, or the Config.prc "cull-bin" variable.)

Youmay assign a node or nodes to an explicit bin using the NodePath::set_bin()interface. set_bin() requires two parameters, the binname and an integer sort parameter; the sort parameter is only meaningfulif the bin type is BT_fixed (more on this below), but it mustalways be specified regardless.

Ifa node is not explicitly assigned to a particular bin, then Panda willassign it into either the "opaque" or the "transparent"bin, accordingto whether it has transparency enabled or not. (Note that thereverse is not true: explicitly assigning an object into the "transparent"bin does not automatically enable transparency for the object.)

Whenthe entire scene has been traversed and all objects have been assignedto bins, then the bins are rendered in order according to theirsort parameter. Within each bin, the contents are sorted accordingto the bin type.

Thefollowing bin types may be specified:

BT_fixed

Renderall of the objects in the bin in a fixed order specified by theuser. This is according to the second parameter of the NodePath::set_bin()method; objects with a lower value are drawn first.

BT_state_sorted

Collectstogether objects that share similar state and renders

themtogether, in an attempt to minimize state transitions in the scene.

BT_back_to_front

Sortseach Geom according to the center of its bounding volume, in lineardistance from the camera plane, so that farther objects are drawnfirst. That is, in Panda's default right-handed Z-up coordinatesystem, objects with large positive Y are drawn before objectswith smaller positive Y.

BT_front_to_back

Thereverse of back_to_front, this sorts so that nearer objects aredrawn first.

BT_unsorted

Objectsare drawn in the order in which they appear in the scene graph,in a depth-first traversal from top to bottom and then from leftto right.

3.9 Howto make multipart actor

3.9.1 MULTIPARTACTORS vs. HALF-BODY ANIMATION

Sometimesyou want to be able to play two different animations on the sameActor at once. Panda does have support for blending two animationson the whole Actor simultaneously, but what if you want to playone animation (say, a walk cycle) on the legs while a completely differentanimation (say, a shoot animation) is playing on the torso? AlthoughPanda doesn't currently have support for playing two differentanimations on different parts of the same actor at once (half-bodyanimation), it does support loading up two completely differentmodels into one actor (multipart actors), which can be used toachieve the same effect, albeit with a bit more setup effort. Multipartactors are more powerful than half-body animations, since youcan completely mix-and-match the pieces with parts from other characters:for instance, you can swap out short legs for long legs to makeyour character taller. On the other hand, multipart actors are alsomore limited in that there cannot be any polygons that straddle theconnecting joint between the two parts.

3.9.2 BROADOVERVIEW

Whatyou have to do is split your character into two completely differentmodels: the legs and the torso. You don't have to do this inthe modeling package; you should be able to do it in the conversion process. The converter needs to be told to get out the entire skeleton,but just a subset of the geometry. Maya2egg, for instance, willdo this with the -subset command-line parameter.
Then,in a nutshell, you load up a multipart actor with the legs and thetorso as separate parts, and you can play the same animation on bothparts, or you can use the per-part interface to play a different animationon each part.

3.9.3 MOREDETAILS

Thatnutshell oversimplifies things only a little bit. Unless your differentanimations are very similar to each other, you will have issueskeeping the different parts from animating in different directions. To solve this, you need to parent them together properly, sothat the torso is parented to the hips. This means exposing the hipjoint in the legs model, and subtracting the hip joint animation fromthe torso model using egg-topstrip (because it will pick it up againwhen it gets stacked up on the hips). Also, you should strongly consideregg-optchar to remove the unused joints from each part's skeleton,although this step is just an optimization.
Unfortunately,all this only works if your character has no polygons thatstraddle the connecting joint between the hips and the torso. If itdoes, you may have to find a clever place to draw the line between them(under a shirt?) so that the pieces can animate in different directionswithout visible artifacts. If that can't be done, then the onlysolution is to add true half-body animation support to Panda. :)

3.9.4 NUTS AND BOLTS

You need to parent the two parts together in Panda. The complete processis this (of course, you'll need to flesh out the details of themaya2egg command line according to the needs of your model, and insertyour own filenames and joint names where appropriate):

(1)Extract out the model into two separate files, legs and torso.
Extractthe animation out twice too, even though both copies will
bethe same, just so it can conveniently exist in two different
eggfiles, one for the legs and one for the torso.
maya2egg-subset legs_group -a model -cn legs -o legs-model.egg myFile.mb
maya2egg-a chan -cn legs -o legs-walk.egg myFile.mb
maya2egg-subset torso_group -a model -cn torso -o torso-model.egg myFile.mb
maya2egg-a chan -cn torso -o torso-walk.egg myFile.mb
Notethat I use the -cn option to give the legs and torso pieces
differentcharacter names. It helps out Panda to know which
animationsare intended to be played with which models, and the
charactername serves this purpose–this way I can now just type:
pviewlegs-model.egg legs-walk.egg torso-model.egg torso-walk.eggPanda willbind up the
appropriateanimations to their associated
modelsautomatically, and I should see my character walking
normally. We could skip straight to step (5) now, but the
characterisn't stacked up yet, and he's only sticking together
nowbecause we're playing the walk animation on both parts at the
sametime–if we want to play different animations on different
parts,we have to stack them.
(2)Expose the hip joint on the legs:
egg-optchar-d opt -expose hip_joint legs-model.egg legs-walk.egg
(3)Strip out the hip joint animation from the torso and egg-optchar
itto remove the leg joints:
egg-topstrip-d strip -t hip_joint torso-model.egg torso-walk.egg
egg-optchar-d opt strip/torso-model.egg strip/torso-walk.egg
(4)Bamify everything.
egg2bam-o legs-model.bam opt/legs-model.egg
egg2bam-o legs-walk.bam opt/legs-walk.egg
egg2bam-o torso-model.bam opt/torso-model.egg
egg2bam-o torso-walk.bam opt/torso-walk.egg
(5)Create a multipart character in Panda. This means loading up the
torsomodel and parenting it, in toto, to the hip joint of the
legs. But the Actor interface handles this for you:
fromdirect.actorimportActor
a=Actor.Actor(

#part dictionary

{‘torso’:’torso-model.bam’,

‘legs’:’legs-model.bam’,

},

#anim dictionary

{‘torso’:{‘walk’:’torso-walk.bam’},

‘legs’:{‘walk’:’legs-walk.bam’},

})

#Tell the Actor how to stack the pieces.

a.attach(‘torso’,’legs’,’hip_joint’)

(6)You can now play animations on the whole actor, or on only part ofit:

a.loop(‘walk’)

a.stop()

a.loop(‘walk’,partName=’legs’)

3.10 MULTIGENMODEL FLAGS

Thisdocument describes the different kinds of model flags one can placein

thecomment field of MultiGen group beads. The general format for amodel

flagis:

{ {value} }

Themost up-to-date version of this document can be found in:

$PANDA/src/doc/howto.MultiGenModelFlags

QUICKREF

FLAG DESCRIPTION

{ {1} } Handle to show/hide, color, etc.a chunk

{ {1} } Handle to move, rotate, scale achunk

{ {barrier} } Invisible collision surface

{ {trigger} } Invisible trigger polygon

{ {floor} } Collides with vertical ray

(usedto specify avatar height and zone)

{ {sphere} } Invisible spherecollision surface

{ {trigger-sphere} } Invisible spherecollision surface

{ {camera-collide} } Invisiblecollision surface for camera

{ {camera-collide-sphere} } Invisiblecollision surface for camera

{ {camera-barrier} } Invisible collisionsurface for camera and colliders

{ {camera-barrier-sphere} } Invisible spherecollision surface for camera and colliders

{ {backstage} } Modeling reference object

{ {1} } Decal the node below to me

(likea window on a wall)

{ fps { # } } Set rate of animation for apfSequence

DETAILS

Theplayer uses several different types of model flags: HANDLES,BEHAVIORS, andPROPERTIES. The following sections give examples of some of the most commonflag/value pairs and describes what they are used for.

3.10.1 HANDLES

Theseflags give the programmers handles which they can use to show/hide,move around, control the texture, etc. of selected segments (chunks)of the model. The handle is the name of the object bead in whichone places the flag (so names like red-hut are more useful than nameslike o34).

{ {1} }

Usedto show/hide, change the color, or change the collision properties ofa chunk.

{ {1} }

Usedto move, rotate, or scale a chunk of the model. Also can be used (likethe flag) to show/hide, change the color, and changethe collisionproperties of a chunk.

3.10.2 BEHAVIORS

Theseflags are used to control collision properties, visibility and behaviorof selected chunks. An “X” in the associated column means:

VISIBLE the object can be seen (see NOTE below for invisible objects)

SOLID avatars can not pass through the object

EVENT an event is thrown whenever an avatar collides with the object

VISIBLE SOLID EVENT

——— ——— ———

{ {barrier} } X X

{ {trigger} } X

{ {backstage} }

Descriptions

- BARRIERS are invisible objects that block the avatars. Use these tofunnel avatars through doorways, keep them from falling off bridges,and so on.

- TRIGGERS can be used to signal when avatars have entered a certain areaof the model. One could place a trigger polygon in front of adoor, for example, so the player can tell when the avatar has movedthrough the door.

- BACKSTAGE objects are not translated over to the player. Modelers shoulduse this flag on reference objects that they include to help inthe modeling task (such as scale references)

IMPORTANTNOTE

Itis not necessary, and in fact some cases it will actually cause problemsif you set the transparency value for the invisible objects above(barrier, trigger, eye-trigger) to 0.0. These objects will automaticallybe invisible in the player if they have been flagged as oneof these three invisible types. If you wish to make it clear in MultiGenthat these objects are invisible objects, set the transparency valueto some intermediate level (0.5). Again, do not set the transparencyvalue to 0.0.

3.10.3 PROPERTIES

Theseare used to control properties of selected chunks.

{ fps { frame-rate } }

Thisspecifies the rate of animation for a pfSequence node

NOTES

---

1)Combinations

MultipleFlag/value pairs can be combined within an single field.

Forexample:

{ {1}

{barrier} }

Generally,the flag can be combined with most other flags

(exceptDCS). Each bead, however, can only have *one* flag.

2)Newlines, spaces, and case (usually) do not matter. This above entry

couldalso be written as:

{{1}{barrier}}

3)Where to place the flags

Allmodel flags except flags are generally placed in the

topmostgroup bead of the geometry to which the flag applies.

GROUP <- place flags here, except

|

-————————————

| | |

OBJECT1 OBJECT2 OBJECT3 . . . ..

| | |

polygons polygons polygons <- place flag here

Flagscan also be placed in object beads, though for consistency sake

itsbetter to place them in the group beads.

4)Flags at different levels in the model

Flagsin lower level beads generally override flags in upper level

beads.

5)For more detailed information see $PANDA/src/doc/eggSyntax.txt.

3.11 Multi-Texturing in Maya

A good rule of thumb is to create yourMulti-Layered shader first to get an idea of what kind of blendmodeyou want. You can do that by using Maya's kLayeredShader.

Following blendmode from Maya is supporteddirectly in Panda.

"Multiply" => "Modulate"

"Over" => "Decal"

"Add" => "Add"

More blendmodes will be supported very soon. Youshould be able to pview this change if you restart Maya from the"runmaya.bat" (or however you restart maya).

Once the shader is setup, you should create thetexture coordinates or uvsets for your multitexture. Make sure, theuvset name matches the shader names that you made

in the kLayeredShader. For Example, if the twoshaders (not the texure file name) in your kLayeredShader are called"base" and "top", then your geometry (that willhave

the layeresShader) will have two uvsets called”base” and “top”.

After this you will link the uvsets to theappropriate shaders.

A reminder note: by default the alpha channel ofthe texture on the bottom is dropped in the conversion. If you wantto retain the alpha channel of your texture,

please make a connection to the alpha channel inMaya when setting up the shader (alpha on the layerShader will behighlighted in yellow).

3.12 Config

This document describes the use of the Panda’sConfig.prc
configuration files and the runtime subsystem thatextracts values
from these files, defined in dtool/src/prc.
The Config.prc files are used for runtimeconfiguration only, and are
not related to the Config.pp files, which controlcompile-time
configuration. If you are looking fordocumentation on the Config.pp
files, see howto.use_ppremake.txt, andppremake-*.txt, in this
directory.

3.12.1 Using the prc files

In its default mode, when Panda starts up it willsearch in the
install/etc directory (or in the directory namedby the environment
variable PRC_DIR if it is set) for all files named*.prc (that is, any
files with an extension of “prc”) andread each of them for runtime
configuration. (It is possible to change thisdefault behavior; see
COMPILE-TIME OPTIONS FOR FINDING PRC FILES,below.)
All of the prc files are loaded in alphabeticalorder, so that the
files that have alphabetically later names areloaded last. Since
variables defined in an later file may shadowvariables defined in an
earlier file, this means that filenames towardsthe end of the
alphabet have the most precedence.
Panda by default installs a handful of system prcfiles into the
install/etc directory. These files have namesbeginning with digits,
like 20_panda.prc and 40_direct.prc, so that theywill be loaded in a
particular order. If you create your own prc filein this directory,
we recommend that you begin its filename withletters, so that it will
sort to the bottom of the list and will thereforeoverride any of the
default variables defined in the system prc files.

Within a particular prc file, you may define anynumber of
configuration variables and their associatedvalue. Each definition
must appear one per line, with at least one spaceseparating the
variable and itsdefinition, e.g.:
load-display pandagl
This specifies that the variable “load-display”should have the value
“pandagl”.
Comments may also appear in the file; they areintroduced by a leading
hash mark (#). A comment may be on a line byitself, or it may be on
the same line following a variable definition; ifit is on the same
line as a variable definition, the hash mark mustbe preceded by at
least one space to separate it from thedefinition.
The legal values that you may specify for anyparticular variable
depends on the variable. The complete list ofavailable variables and
the valid values for each is not documented here(a list of the most
commonly modified variables appears in anotherdocument, but also see
cvMgr.listVariables(), below).
Many variables accept any string value (such asload-display, above);
many others, such as aspect-ratio, expect anumeric value.
A large number of variables expect a simpleboolean true/false value.
You may observe the Python convention of using 0vs. 1 to represent
false vs. true; or you may literally type “false”or “true”, or just
“f” and “t”. For historicalreasons, Panda also recognizes the Scheme
convention of “#f” and “#t”.
Most variables only accept one value at a time. If there are two
different definitions for a given variable in thesame file, the
topmost definition applies. If there are twodifferent definitions in
two different files, the definition given in thefile loaded later
applies.
However, some variables accept multiple values. This is particularly
common for variables that name search directories,like model-path.
In the case of this kind of variable, alldefinitions given for the
variable are taken together; it is possible toextend the definition
by adding another prc file, but you cannot removeany value defined in
a previously-loaded prc file.

3.12.2 Defining config variables

New config variables may be defined on-the-fly ineither C++ or Python
code. To do this, create an instance of one ofthe following classes:
ConfigVariableString
ConfigVariableBool
ConfigVariableInt
ConfigVariableDouble
ConfigVariableFilename
ConfigVariableEnum (C++ only)
ConfigVariableList
ConfigVariableSearchPath
These each define a config variable of thecorresponding type. For
instance, a ConfigVariableInt defines a variablewhose value must
always be an integer value. The most commonvariable types are the
top four, which are self-explanatory; theremaining four are special
types:
ConfigVariableFilename -
This is a convenience class which behaves verymuch like a
ConfigVariableString, except that itautomatically converts from
OS-specific filenames that may be given in theprc file to
Panda-specific filenames, and it alsoautomatically expands
environment variable references, so that theuser may name a file
based on the value of an environment variable
(e.g. $PANDAMODELS/file.egg).
ConfigVariableEnum -
This is a special template class available inC++ only. It provides
a convenient way to define a variable that mayaccept any of a
handful of different values, each of which isdefined by a keyword.
For instance, the text-encoding variable may beset to any of
“iso8859”, “utf8”, or”unicode”, which correspond to
TextEncoder::E_iso8859, E_utf8, and E_unicode,respectively.
The ConfigVariableEnum class relies on a havingsensible pair of
functionsdefined for operator << (ostream) and operator >>(istream)
for the enumerated type. These two functionsshould
reverse each other, so that the output operatorgenerates a keyword
for each value of the enumerated type, and theinput operator

recognizes each of the keywords generated by theoutput operator.
This is a template class. It is templated onits enumerated type,
e.g. ConfigVariableEnum<TextEncoder::Encoding>.
ConfigVariableList -
This class defines a special config variablethat records all of its
definitions appearing in all prc files andretrieves them as a list,
instead of a standard config variable thatreturns only the topmost
definition. (See "some variables acceptmultiple values", above.)
Unlike the other kinds of config variables, aConfigVariableList is
read-only; it can be modified only by loadingadditional prc files,
rather than directly setting its value. Also,its constructor lacks
a default_value parameter, since there is nodefault value (if the
variable is not defined in any prc file, itsimply returns an empty
list).
ConfigVariableSearchPath -
This class is very similar to aConfigVariableList, above, except
that it is intended specifically to representthe multiple
directories of a search path. In general, a
ConfigVariableSearchPath variable can be used inplace of a
DSearchPath variable.
Unlike ConfigVariableList, instances of thisvariable can be locally
modified by appending or prepending additionaldirectory names.
In general, each of the constructors to the aboveclasses accepts the
following parameters:
(name, default_value, description = "",flags = 0)
The default_value parameter should be of the sametype as the variable
itself; for instance, the default_value for aConfigVariableBool must
be either true or false. The ConfigVariableListand ConfigVariableSearchPath constructors do not have a default_value
parameter.
The description should be a sentence or twodescribing the purpose of
the variable and the effects of setting it. Itwill be reported with
variable.getDescription() orConfigVariableManager.listVariables();
see QUERYING CONFIG VARIABLES, below.
The flags variable is usually set to 0, but it maybe an integer trust
level and/or the union of any of the values in theenumerated type
ConfigFlags::VariableFlags. For the most part,this is used to

restrict the variable from being set by unsignedprc files. See
SIGNED PRC FILES, below.
Once you have created a config variable of theappropriate type, you
may generally treat it directly as a simplevariable of that type.
This works in both C++ and in Python. Forinstance, you may write
code such as this:
ConfigVariableInt foo_level("foo-level",-1, "The level of foo");
if (foo_level < 0) {
cerr << "You didn't specify a validfoo_level!\n";
} else {
// Four snarfs for every foo.
int snarf_level = 4 * foo_level;
}
In rare cases, you may find that the implicittypecast operators
aren't resolved properly by the compiler; if thishappens, you can use
variable.get_value() to retrieve the variable'svalue explicitly.

3.12.3 Directly assigning config variables

In general, config variables can be directlyassigned values
appropriate to their type, as if they wereordinary variables. In
C++, the assignment operator is overloaded toperform this function,
e.g.:
foo_level = 5;
In Python, this syntax is not possible–theassignment operator in
Python completely replaces the value of theassigned symbol and cannot
be overloaded. So the above statement in Pythonwould replace
foo_level with an actual integer of the value 5. In many cases, this
is close enough to what you intended anyway, butif you want to keep
the original functionality of the config variable(e.g. so you can
restore it to its original value later), you needto use the
set_value() method instead, like this:
fooLevel.setValue(5)
When you assign a variable locally, the newdefinition shadows all prc
files that have been read or will ever be read,until you clear your
definition. To restore a variable to its originalvalue as defined by
the topmost prc file, use clear_local_value():

fooLevel.clearLocalValue()
This interface for assigning config variables isprimarily intended
for the convenience of developing an applicationinteractively; it is
sometimes useful to change the value of a variableon the fly.

3.12.4 Querying config variables

There are several mechanisms for finding out thevalues of individual
config variables, as well as for finding thecomplete list of
available config variables.
In particular, one easy way to query an existingconfig variable's
value is simply to create a new instance of thatvariable, e.g.:
print ConfigVariableInt("foo-level")
The default value and comment are optional ifanother instance of the
same config variable has previously been created,supplying these
parameters. However, it is an error if noinstance of a particular
config variable specifies a default value. It isalso an error (but
it is treated as a warning) if two differentinstances of a variable
specify different default values.
(Note that, although it is convenient to create anew instance of the
variable in order to query or modify its valueinteractively, we
recommend that all the references to a particularvariable in code
should use the same instance wherever possible. This minimizes the
potential confusion about which instance shoulddefine the variable's
default value and/or description, and reduceschance of conflicts
should two such instances differ.)
If you don't know the type of the variable, youcan also simply create
an instance of the generic ConfigVariable class,for the purpose of
querying an existing variable only (you should notdefine a new
variable using the generic class).
To find out more detail about a variable and itsvalue, use the ls()
method in Python (or the write() method in C++),e.g.:
ConfigVariable("foo-level").ls()
In addition to the variable's current and defaultvalues, this also
prints a list of all of the prc files thatcontributed to the value of
the variable, as well as the description providedfor the variable.
To get a list of all known config variables, usethe methods on

ConfigVariableManager. In C++, you can get apointer this object via
ConfigVariableManager::get_global_ptr(); inPython, use the cvMgr
builtin, created by ShowBase.py.
print cvMgr
Lists all of the variables in active use: allof the variables
whose value has been set by one or more prcfiles, along with the
name of the prc file that defines that value.
cvMgr.listVariables()
Lists all of the variables currently known tothe config system;
that is, all variables for which aConfigVariable instance has
been created at runtime, whether or not itsvalue has been changed
from the default. This may omit variablesdefined in some unused
subsystem (like pandaegg, for instance), andit will omit
variables defined by Python code which hasn'tyet been executed
(e.g. variables within defined with a functionthat hasn't yet been called).
This will also omit variables deemed to be"dynamic" variables,
for instance all of the notify-level-*variables, and variables
such as pstats-active-*. These are omittedsimply to keep the
list of variable names manageable, since thelist of dynamic
variable names tends to be very large. Use
cvMgr.listDynamicVariables() if you want tosee these variable
names.
cvMgr.listUnusedVariables()
Lists all of the variables that have beendefined by some prc
file, but which are not known to the configsystem (no
ConfigVariable instance has yet been createdfor this variable).
These variables may represent misspellings ortypos in your prc
file, or they may be old variables which areno longer used in the
system. However, they may also be legitimatevariables for some
subsystem or application which simply has notbeen loaded; there
is no way for Panda to make this distinction.

3.12.5 Re-reading prc files

If you modify a prc file at some point after Pandahas started, Panda
will not automatically know that it needs toreload its config files
and will not therefore automatically recognizeyour change. However,

you can force this to happen by making thefollowing call:

ConfigPageManager::get_global_ptr()->reload_implicit_pages()

Or, in Python:

cpMgr.reloadImplicitPages()

This will tell Panda to re-read all of the prcfiles it found

automatically at startup and update the variables'values accordingly.

3.12.6 Runtime prc file management

In addition to the prc files that are found andloaded automatically by Panda at startup, you can load files up atruntime as needed. The

functions to manage this are defined inload_prc_file.h:

ConfigPage *page = load_prc_file("myPage.prc")

. . .

unload_prc_file(page);

(The above shows the C++ syntax; the correspondingPython code is

similar, but of course the functions are namedloadPrcFile() and

unloadPrcFile().)

That is to say, you can call load_prc_file() toload up a new prc file

at any time. Each file you load is added to aLIFO stack of prc

files. If a variable is defined in more than oneprc file, the

topmost file on the stack (i.e. the one mostrecently loaded) is the

one that defines the variable's value.

You can call unload_prc_file() at any time tounload a file that you

have previously loaded. This removes the filefrom the stack and

allows any variables it modified to return totheir previous value.

The single parameter to unload_prc_file() shouldbe the pointer that

was returned from the corresponding call toload_prc_file(). Once you

have called unload_prc_file(), the pointer isinvalid and should no

longer be used. It is an error to callunload_prc_file() twice on the

same pointer.

The filename passed to load_prc_file() may referto any file that is

on the standard prc file search path (e.g.$PRC_DIR), as well as on

the model-path. It may be a physical file ondisk, or a subfile of a

multifile (and mounted via Panda's virtual filesystem).

If your prc file is stored as an in-memory stringinstead of as a disk

file (for instance, maybe you just built it up),you can use the

load_prc_file_data() method to load the prc filefrom the string data.
The first parameter is an arbitrary name to assignto your in-memory
prc file; supply a filename if you have one, oruse some other name
that is meaningful to you.
You can see the complete list of prc files thathave been loaded into
the config system at any given time, includingfiles loaded explicitly
via load_prc_file(), as well as files found in thestandard prc file
search path and loaded implicitly at startup. Simply use
ConfigPageManager::write(), e.g. in Python:
print cpMgr

3.12.7 Compile-time options for finding prc files

As described above in USING THE PRC FILES, Panda's default startup
behavior is to load all files named *.prc in the directory named by
the environment variable PRC_DIR. This isactually a bit of an
oversimplification. The complete default behavioris as follows:
(1) If PRC_PATH is set, separate it into a list ofdirectories and
make a search path out of it.
(2) If PRC_DIR is set, prepend it onto the searchpath defined by
PRC_PATH, above.
(3) If neither was set, put the compiled-in valuefor DEFAULT_PRC_DIR,
which is usually the install/etc directory,alone on the search
path.
Steps (1), (2), and (3) define what isreferred to in this
document as "the standard prc searchpath". You can query this
search path via cpMgr.getSearchPath().
(4) Look for all files named *.prc on eachdirectory of the resulting
search path, and load them up in reversesearch path order, and
within each directory, in forward alphabeticalorder. This means
that directories listed first on the searchpath override
directories listed later, and within adirectory, files
alphabetically later override filesalphabetically earlier.
This describes the default behavior, without anymodifications to
Config.pp. If you wish, you can further fine-tuneeach of the above
steps by defining various Config.pp variables atcompile time. The
following Config.pp variables may be defined:

#define PRC_PATH_ENVVARS PRC_PATH
#define PRC_DIR_ENVVARS PRC_DIR
These name the environment variable(s) to useinstead of PRC_PATH
and PRC_DIR. In either case, you may namemultiple environment
variables separated by a space; each variable isconsulted one at a
time, in the order named, and the results areconcatenated.
For instance, if you put the following line inyour Config.pp file:
#define PRC_PATH_ENVVARS CFG_PATH ETC_PATH
Then instead of checking $PRC_PATH in step (1),above, Panda will
first check $CFG_PATH, and then $ETC_PATH, andthe final search path
will be the concatenation of both.
You can also define either or both ofPRC_PATH_ENVVARS or
PRC_DIR_ENVVARS to the empty string; this willdisable runtime
checking of environment variables, and force allprc files to be
loaded from the directory named byDEFAULT_PRC_DIR.
#define PRC_PATTERNS *.prc
This describes the filename patterns that areused to identify prc
files in each directory in step(4), above. Thedefault is *.prc,
but you can change this if you have any reasonto. You can specify
multiple filename patterns separated by a space. For instance, if
you still have some config files named"Configrc", following an
older Panda convention, you can define thefollowing in your
Config.pp file:
#define PRC_PATTERNS *.prc Configrc
This will cause Panda to recognize files named"Configrc", as well
as any file ending in the extension prc, as alegitimate prc file.
#define DEFAULT_PRC_DIR $[INSTALL_DIR]/etc
This is the directory from which to load prcfiles if all of the
variables named by PRC_PATH_ENVVARS andPRC_DIR_ENVVARS are
undefined or empty.
#define DEFAULT_PATHSEP
This doesn't strictly apply to the configsystem, since it globally
affects search paths throughout Panda. Thisspecifies the character
or characters used to separate the differentdirectory names of a
search path, for instance $PRC_PATH. Thedefault character is ':'
on Unix, and ';' on Windows. If you specifymultiple characters,

any of them may be used as a separator.

3.12.8 Executable prc files

One esoteric feature of Panda's config system isthe ability to
automatically execute a standalone program whichgenerates a prc file
as output.
This feature is not enabled by default. To enableit, you must define
the Config.pp variable PRC_EXECUTABLE_PATTERNSbefore you build Panda.
This variable is similar to PRC_PATTERNS,described above, except it
names file names which, when found along thestandard prc search path,
should be taken to be the name of an executableprogram. Panda will
execute each of these programs, in the appropriateorder according to
alphabetical sorting with the regular prc files,and whatever the
program writes to standard output is taken to bethe contents of a prc
file.
By default the contents of the environmentvariable
$PRC_EXECUTABLE_ARGS are passed as arguments tothe executable
program. You can change this to a differentenvironment variable by
redefining PRC_EXECUTABLE_ARGS_ENVVAR in yourConfig.pp (or prevent
the passing of arguments by defining this to theempty string).

3.12.9 Signed prc files

Another esoteric feature of Panda's config systemis the ability to
restrict certain config variables to modificationonly by a prc file
that has been provided by an authorized source. This is primarily
useful when Panda is to be used for deployment ofapplications (games,
etc.) to a client; it has little utility in afully trusted
environment.
When this feature is enabled, you can specify anoptional trust level
to each ConfigVariable constructor. The trustlevel is an integer
value, greater than 0 (and <=ConfigFlags::F_trust_level_mask), which
should be or'ed in with the flags parameter.
A number of random keys must be generated ahead oftime and compiled
into Panda; there must be a different key for eachdifferent trust
level. Each prc file can then optionally besigned by exactly one of
the available keys. When a prc file has beensigned by a recognized

key, Panda assigns the corresponding trust levelto that prc file. An unsigned prc file has an implicit trust level of0.
If a signed prc file is modified in any way afterit has been signed, its signature will no longer match the contents ofthe file and its trust level drops to 0. The newly-modified filemust be signed again to restore its trust level.
When a ConfigVariable is constructed with anonzero trust level, that variable's value may then not be set by any prcfile with a trust level lower that the variable's trust level. If aprc file with an insufficient trust level attempts to modify thevariable, the new value is ignored, and the value from the previoustrusted prc file (or the variable's default value) is retained.
The default trust level for a ConfigVariable is 0,which means the variable can be set by any prc file, signed orunsigned. To set any nonzero trust level, pass the integer trust levelvalue as the flags parameter to the ConfigVariable constructor. Toexplicitly specify a trust level of 0, pass ConfigFlags::F_open.
To specify a ConfigVariable that cannot be set byany prc files at all, regardless of trust level, useConfigFlags::F_closed.
This feature is not enabled by default. It issomewhat complicated to enable this feature, because doing so requresgenerating one or more private/public key pairs, and compiling the publickeys into the low-level Panda system so that it can recognizesigned prc files when they are provided, and compiling the private keysinto standalone executables, one for each private key, that can beused to officially sign approved prc files. This initial setuptherefore requires a bit of back-and-forth building and rebuilding in thedtool directory.
To enable this feature, follow the followingprocedure.
(1) Decide how many different trust levels yourequire. You can have as many as you want, but most applicationswill require only one trust level, or possibly two. The rareapplication will require three or more. If you decide to use multipletrust levels, you can make a distinction between configvariables that are somewhat sensitive and those that are highly sensitive.
(2) Obtain and install the OpenSSL library, if itis not already installed (http://www.openssl.org). Adjustyour Config.pp file as

necessary to point to the installed OpenSSLheaders and libraries
(in particular, define SSL_IPATH andSSL_LIBS), and then ppremake
and make install your dtool tree. It is notnecessary to build
the panda tree or any subsequent trees yet.
(3) Set up a directory to hold the generatedpublic keys. The
contents of this directory must be accessibleto anyone building
Panda for your application; it also must havea lifetime at least
as long as the lifetime of your application. It probably makes
sense to make this directory part of yourapplication's source
tree. The contents of this directory will notbe particularly
sensitive and need not be kept any more secretthan the rest of
your application's source code.
(4) Set up a directory in a secure place to holdthe generated private
keys. The contents of this directory shouldbe regarded as
somewhat sensitive, and should not beavailable to more than a
manageable number of developers. It need notbe accessible to
people building Panda. However, thisdirectory should have a
lifetime as long as the lifetime of yourapplication. Depending
on your environment, it may or may not makesense to make this
directory a part of your application's sourcetree; it can be the
same directory as that chosen for (3), above.
(5) Run the program make-prc-key. This programgenerates the public
and private key pairs for each of your trustlevels. The
following is an example:
make-prc-key -a /keys.cxx -b/sign#.cxx 1 2
The output of make-prc-key will be compilableC++ source code.
The first parameter, -a, specifies the name ofthe public key
output file. This file will contain all of thepublic keys for
the different trust levels, and will becomepart of the libdtool
library. It is not particularly sensitive,and must be accessible
to anyone who will be compiling dtool.
The second parameter, -b, specifies acollection of output files,
one for each trust level. Each file can becompiled as a
standalone program (that links with libdtool);the resulting
program can then be used to sign any prc fileswith the
corresponding trust level. The hash character'#' appearing in

the filename will be filled in with thenumeric trust level.
The remaining arguments to make-prc-key arethe list of trust
levels to generate key pairs for. In theexample above, we are
generating two key pairs, for trust level 1and for trust level 2.
The program will prompt you to enter a passphrase for each
private key. This pass phrase is used toencrypt the private key
as written into the output file, to reduce thesensitivity of the
prc signing program (and its source code). The user of the
signing program must re-enter this pass phrasein order to sign a
prc file. You may specify a different passphrase for each trust
level, or you may use the -p “passphrase” command-line option to
provide the same pass phrase for all trustlevels. If you do not
want to use the pass phrase feature at all,use -p “”, and keep
the generated programs in a safe place.
(6) Modify your Config.pp file (for yourself, andfor anyone else who
will be building dtool for your application)to add the following
line:
#define PRC_PUBLIC_KEYS_FILENAME/keys.cxx
Where /keys.cxx is the filenamed by -a, above.
Consider whether you want to enforce the trustlevel in the
development environment. The default is torespect the trust
level only when Panda is compiled for arelease build, i.e. when
OPTIMIZE is set to 4. You can redefinePRC_RESPECT_TRUST_LEVEL if
you want to change this default behavior.
Re-run ppremake and then make install indtool.
(7) Set up a Sources.pp file in your private keydirectory to compile
the files named by -b against dtool. Itshould contain an entry
something like these for each trust level:
#begin bin_target
#define OTHER_LIBS dtool
#define USE_PACKAGES ssl
#define TARGET sign1
#define SOURCES sign1.cxx
#end bin_target
#begin bin_target
#define OTHER_LIBS dtool

#define USE_PACKAGES ssl
#define TARGET sign2
#define SOURCES sign2.cxx
#end bin_target
(8) If your private key directory is not a part ofyour application
source hierarchy (or your application does notuse ppremake),
create a Package.pp in the same directory tomark the root of a
ppremake source tree. You can simply copy thePackage.pp file
from panda/Package.pp. You do not need to dothis if your private
key directory is already part of appremake-controlled source
hierarchy.
(9) Run ppremake and then make install in theprivate key directory.
This will generate the programs sign1 andsign2 (or whatever you
have named them). Distribute these programsto the appropriate
people who have need to sign prc files, andtell them the pass
phrases that you used to generate them.
(10) Build the rest of the Panda trees normally.
Advanced tip: if you follow the directions above,your sign1 and sign2
programs will require libdtool.dll at runtime, andmay need to be
recompiled from time to time if you get a newversion of dtool. To
avoid this, you can link these programsstatically, so that they are
completely standalone. This requires one moreback-and-forth
rebuilding of dtool:
(a) Put the following line in your Config.pp file:
#define LINK_ALL_STATIC 1
(b) Run ppremake and make clean install in dtool. Note that you must
make clean. This will generate a staticversion of libdtool.lib.
(c) Run ppremake and make clean install in yourprivate key directory,
to recompile the sign programs against the newstatic libdtool.lib.
(d) Remove (or comment out) the LINK_ALL_STATICline in your Config.pp
file.
(e) Run ppremake and make clean install in dtoolto restore the normal
dynamic library, so that future builds ofpanda and the rest of
your application will use the dynamiclibdtool.dll properly.

CHAPTER 4

Miscellaneous and F.A.Q.

lorem ipsum

CHAPTER 5

Appendix

lorem ipsum