PyFFI

Release 2.2.4.dev4

Amorilia

Jan 06, 2020

CONTENTS

Release 2.2.4.dev4

Date Jan 06, 2020

CHAPTER

ONE

PYFFI

The Python File Format Interface, briefly PyFFI, is an open source Python library for processing block structured binary files:

- **Simple:** Reading, writing, and manipulating complex binary files in a Python environment is easy! Currently, PyFFI supports the NetImmerse/Gamebryo NIF and KFM formats, CryTek's CGF format, the FaceGen EGM format, the DDS format, and the TGA format.
- **Batteries included:** Many tools for files used by 3D games, such as optimizers, stripifier, tangent space calculator, 2d/3d hull algorithms, inertia calculator, as well as a general purpose file editor QSkope (using PyQt4), are included.
- **Modular:** Its highly modular design makes it easy to add support for new formats, and also to extend existing functionality.

1.1 Download

Get PyFFI from Github, or install it with:

```
easy_install -U PyFFI
```

or:

```
pip3 install PyFFI
```

1.2 Developing

To get the latest (but possibly unstable) code, clone PyFFI from its Git repository:

```
git clone --recursive git://github.com/niftools/pyffi.git
virtualenv -p python3 venv
source venv/bin/activate
pip install -r requirements/requirements-dev.txt
```

Be sure to use the –recursive flag to ensure that you also get all of the submodules.

If you wish to code on PyFFI and send your contributions back upstream, get a github account and fork PyFFI.

1.3 Testing

We love tests, they help guarantee that things keep working they way they should. You can run them yourself with the following:

```
source venv/bin/activate
nosetest -v test
```

or:

```
source venv/bin/activate
py.test -v tests
```

1.4 Documentation

All our documentation is written in ReST and can be generated into HTML, LaTeX, PDF and more thanks to Sphinx. You can generate it yourself:

```
source venv/bin/activate
cd docs
make html -a
```

1.5 Examples

- The Blender NIF Plugin
- QSkope PyFFI's general purpose file editor.
- The niftoaster (PyFFI's "swiss army knife") can for instance optimize NIF files, and much more.

1.6 Questions? Suggestions?

- Open an issue at the issue tracker.

1.7 Documentation

1.7.1 Introduction

Example and Problem Description

Consider an application which processes images stored in for instance the Targa format:

```
>>> # read the file
>>> stream = open("tests/tga/test.tga", "rb")
>>> data = bytearray(stream.read()) # read bytes
>>> stream.close()
>>> # do something with the data...
```

(continues on next page)

(continued from previous page)

```
>>> data[8] = 20 # change x origin
>>> data[10] = 20 # change y origin
>>> # etc... until we are finished processing the data
>>> # write the file
>>> from tempfile import TemporaryFile
>>> stream = TemporaryFile()
>>> dummy = stream.write(data) # py3k returns number of bytes written
>>> stream.close()
```

This methodology will work for any file format, but it is usually not very convenient. For complex file formats, the *do something with the data* part of the program would be necessarily quite complicated for the programmer. For this reason, it is convenient to convert the data (a sequence of bytes) into an organized collection of Python objects (a class suits this purpose perfectly) that clearly reveal what is stored in the data. Such organized collection is called an *interface*:

```
>>> import struct
>>> from tempfile import TemporaryFile
>>> class TgaFile:
...     """A simple class for reading and writing Targa files."""
...     def read(self, filename):
...         """Read tga file from stream."""
...         stream = open(filename, "rb")
...         self.image_id_length, self.colormap_type, self.image_type, \
...         self.colormap_index, self.colormap_length, self.colormap_size, \
...         self.x_origin, self.y_origin, self.width, self.height, \
...         self.pixel_size, self.flags = struct.unpack("<BBBHHBHHHHBB",
...                                                     stream.read(18))
...         self.image_id = stream.read(self.image_id_length)
...         if self.colormap_type:
...             self.colormap = [
...                 stream.read(self.colormap_size >> 3)
...                 for i in range(self.colormap_length)]
...         else:
...             self.colormap = []
...         self.image = [[stream.read(self.pixel_size >> 3)
...                        for i in range(self.width)]
...                       for j in range(self.height)]
...         stream.close()
...     def write(self, filename=None):
...         """Read tga file from stream."""
...         if filename:
...             stream = open(filename, "wb")
...         else:
...             stream = TemporaryFile()
...         stream.write(struct.pack("<BBBHHBHHHHBB",
...             self.image_id_length, self.colormap_type, self.image_type,
...             self.colormap_index, self.colormap_length,
...             self.colormap_size,
...             self.x_origin, self.y_origin, self.width, self.height,
...             self.pixel_size, self.flags))
...         stream.write(self.image_id)
...         for entry in self.colormap:
...             stream.write(entry)
...         for line in self.image:
...             for pixel in line:
...                 stream.write(pixel)
```

(continues on next page)

(continued from previous page)

```
...         stream.close()
>>> data = TgaFile()
>>> # read the file
>>> data.read("tests/tga/test.tga")
>>> # do something with the data...
>>> data.x_origin = 20
>>> data.y_origin = 20
>>> # etc... until we are finished processing the data
>>> # write the file
>>> data.write()
```

The reading and writing part of the code has become a lot more complicated, but the benefit is immediately clear: instead of working with a sequence of bytes, we can directly work with the members of our `TgaFile` class, and our code no longer depends on how exactly image data is organized in a Targa file. In other words, our code can now use the semantics of the `TgaFile` class, and is consequently much easier to understand and to maintain.

In practice, however, when taking the above approach as given, the additional code that enables this semantic translation is often difficult to maintain, for the following reasons:

- **Duplication:** Any change in the reader part must be reflected in the writer part, and vice versa. Moreover, the same data types tend to occur again and again, leading to nearly identical code for each read/write operation. A partial solution to this problem would be to create an additional class for each data type, each with its read and write method.
- **No validation:** What if `test/tga/test.tga` is not a Targa file at all, or is corrupted? What if `image_id` changes length but `image_id_length` is not updated accordingly? Can we catch such bugs and prevent data to become corrupted?
- **Boring:** Writing *interface* code gets boring very quickly.

What is PyFFI?

PyFFI aims to solve all of the above problems:

- The *interface* classes are *generated at runtime*, from an easy to maintain description of the file format. The generated classes provides semantic access to *all* information in the files.
- Validation is automatically enforced by the generated classes, except in a few rare cases when automatic validation might cause substantial overhead. These cases are well documented and simply require an explicit call to the validation method.
- The generated classes can easily be extended with additional class methods, for instance to provide common calculations (for example: converting a single pixel into greyscale).
- Very high level functions can be implemented as *spell*s (for example: convert a height map into a normal map).

1.7.2 Installation

Requirements

To run PyFFI's graphical file editor QSkope, you need PyQt4.

Using the Windows installer

Simply download and run the Windows installer provided in our Releases.

Manual installation

If you install PyFFI manually, and you already have an older version of PyFFI installed, then you **must** uninstall (see *Uninstall*) the old version before installing the new one.

Installing via setuptools

If you have setuptools installed, simply run:

```
easy_install -U PyFFI
```

at the command prompt.

Installing from source package

First, get the source package. Untar or unzip the source package via either:

```
tar xfvz PyFFI-x.x.x.tar.gz
```

or:

```
unzip PyFFI-x.x.x.zip
```

Change to the PyFFI directory and run the setup script:

```
cd PyFFI-x.x.x
python setup.py install
```

Uninstall

You can uninstall PyFFI manually simply by deleting the `pyffi` folder from your Python's `site-packages` folder, which is typically at:

```
C:\Python25\Lib\site-packages\pyffi
```

or:

```
/usr/lib/python2.5/site-packages/pyffi
```

1.7.3 pyffi — Interfacing block structured files

pyffi.formats — File format interfaces

When experimenting with any of the supported file formats, you can specify an alternate location where you store your modified format description by means of an environment variable. For instance, to tell the library to use your version of `cgf.xml`, set the `CGFXMLPATH` environment variable to the directory where `cgf.xml` can be found. The environment variables `NIFXMLPATH`, `KFMXMLPATH`, `DDSXMLPATH`, and `TGAXMLPATH` work similarly.

Supported formats

pyffi.formats.bsa — Bethesda Archive (.bsa)

Warning: This module is still a work in progress, and is not yet ready for production use.

A .bsa file is an archive format used by Bethesda (Morrowind, Oblivion, Fallout 3).

Implementation

class pyffi.formats.bsa.**BsaFormat**

Bases: pyffi.object_models.xml.FileFormat

This class implements the BSA format.

class BZString(***kwargs*)

Bases: pyffi.object_models.common.SizedString

get_size(*data=None*)

Return number of bytes this type occupies in a file.

Returns Number of bytes.

read(*stream*, *data=None*)

Read string from stream.

Parameters **stream** (*file*) – The stream to read from.

write(*stream*, *data=None*)

Write string to stream.

Parameters **stream** (*file*) – The stream to write to.

Data

alias of *Header*

class FileVersion(***kwargs*)

Bases: pyffi.object_models.common.UInt

Basic type which implements the header of a BSA file.

get_size(*data=None*)

Return number of bytes the header string occupies in a file.

Returns Number of bytes.

read(*stream*, *data*)

Read header string from stream and check it.

Parameters **stream** (*file*) – The stream to read from.

write(*stream*, *data*)
Write the header string to stream.
Parameters **stream** (*file*) – The stream to write to.

class Hash(***kwargs*)
Bases: pyffi.object_models.common.UInt64

get_detail_display()
Return an object that can be used to display the instance.

class Header(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.bsa._Header, pyffi.object_models.Data

A class to contain the actual bsa data.

inspect(*stream*)
Quickly checks if stream contains BSA data, and reads the header.
Parameters **stream** (*file*) – The stream to inspect.

inspect_quick(*stream*)
Quickly checks if stream contains BSA data, and gets the version, by looking at the first 8 bytes.
Parameters **stream** (*file*) – The stream to inspect.

read(*stream*)
Read a bsa file.
Parameters **stream** (file) – The stream from which to read.

write(*stream*)
Write a bsa file.
Parameters **stream** (file) – The stream to which to write.

UInt32
alias of pyffi.object_models.common.UInt

class ZString(***kwargs*)
Bases: pyffi.object_models.xml.basic.BasicBase, pyffi.object_models.editable.EditableLineEdit

String of variable length (null terminated).

```
>>> from tempfile import TemporaryFile
>>> f = TemporaryFile()
>>> s = ZString()
>>> if f.write('abcdefghijklmnopqrst\x00'.encode("ascii")): pass # b'abc...'
>>> if f.seek(0): pass # ignore result for py3k
>>> s.read(f)
>>> str(s)
'abcdefghijklmnopqrst'
>>> if f.seek(0): pass # ignore result for py3k
>>> s.set_value('Hi There!')
>>> s.write(f)
>>> if f.seek(0): pass # ignore result for py3k
>>> m = ZString()
>>> m.read(f)
>>> str(m)
'Hi There!'
```

get_hash(*data=None*)
Return a hash value for this string.
Returns An immutable object that can be used as a hash.

get_size(*data=None*)
Return number of bytes this type occupies in a file.
Returns Number of bytes.

get_value()
Return the string.
Returns The stored string.
Return type C{bytes}

read(*stream*, *data=None*)
Read string from stream.
Parameters **stream** (*file*) – The stream to read from.

set_value(*value*)
Set string to C{value}.
Parameters **value** (str (will be encoded as default) or C{bytes}) – The value to assign.

write(*stream*, *data=None*)
Write string to stream.
Parameters **stream** (*file*) – The stream to write to.

static version_number(*version_str*)
Converts version string into an integer.

Parameters **version_str** (*str*) – The version string.

Returns A version integer.

```
>>> BsaFormat.version_number('103')
103
>>> BsaFormat.version_number('XXX')
-1
```

Regression tests

Read a BSA file

```
>>> # check and read bsa file
>>> from os.path import dirname
>>> dirpath = __file__
>>> for i in range(4): #recurse up to root repo dir
...     dirpath = dirname(dirpath)
>>> repo_root = dirpath
>>> format_root = os.path.join(repo_root, 'tests', 'formats', 'bsa')
>>> stream = open(os.path.join(format_root, 'test.bsa'), 'rb')
>>> data = BsaFormat.Data()
>>> data.inspect_quick(stream)
>>> data.version
103
>>> data.inspect(stream)
>>> data.folders_offset
36
>>> hex(data.archive_flags.get_attributes_values(data))
'0x703'
>>> data.num_folders
1
>>> data.num_files
```

(continues on next page)

(continued from previous page)

```
7
>>> #data.read(stream)
>>> # TODO check something else...
```

Parse all BSA files in a directory tree

```
>>> for stream, data in BsaFormat.walkData(format_root):
...     try:
...         # the replace call makes the doctest also pass on windows
...         os_path = stream.name
...         split = (os_path.split(os.sep))[-4:]
...         rejoin = os.path.join(*split).replace(os.sep, "/")
...         print("reading %s" % rejoin)
...         data.read(stream)
...     except Exception:
...         print(
...             "Warning: read failed due corrupt file,"
...             " corrupt format description, or bug.")
reading tests/formats/bsa/test.bsa
```

Create an BSA file from scratch and write to file

```
>>> data = BsaFormat.Data()
>>> # TODO store something...
>>> from tempfile import TemporaryFile
>>> stream = TemporaryFile()
>>> #data.write(stream)
```

pyffi.formats.cgf — Crytek (.cgf and .cga)

Implementation

class pyffi.formats.cgf.**CgfFormat**

Bases: pyffi.object_models.xml.FileFormat

Stores all information about the cgf file format.

class **AbstractMtlChunk**(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.cgf.Chunk

Common parent for MtlChunk and MtlNameChunk.

class **AbstractObjectChunk**(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.cgf.Chunk

Common parent for HelperChunk and MeshChunk.

class **BoneLink**(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.object_models.xml.struct_.StructBase

A bone link.

exception CgfError
Bases: Exception

Exception for CGF specific errors.

class Chunk(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.cgf._Chunk, object

apply_scale(*scale*)
Apply scale factor on data.

tree(*block_type=None*, *follow_all=True*)
A generator for parsing all blocks in the tree (starting from and including C{self}).
Parameters
- **block_type** – If not None, yield only blocks of the type C{block_type}.
- **follow_all** – If C{block_type} is not None, then if this is True the function will parse the whole tree. Otherwise, the function will not follow branches that start by a non-C{block_type} block.

class ChunkHeader(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A CGF chunk header.

class ChunkTable(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.cgf._ChunkTable, object

get_chunk_types()
Iterate all chunk types (in the form of Python classes) referenced in this table.

class ChunkType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

An unsigned 32-bit integer, describing the chunk type.

class ChunkVersion(***kwargs*)
Bases: pyffi.object_models.common.UInt

The version of a particular chunk, or the version of the chunk table.

class Data(*filetype=4294901760*, *game='Far Cry'*)
Bases: pyffi.object_models.Data

A class to contain the actual cgf data.

Note that L{versions} and L{chunk_table} are not automatically kept in sync with the L{chunks}, but they are resynchronized when calling L{write}.

Variables

- **game** – The cgf game.
- **header** – The cgf header.
- ***chunks*** – List of chunks (the actual data).
- ***versions*** – List of chunk versions.

get_detail_child_names(*edge_filter=(True, True)*)
Generator which yields all child names of this item in the detail view.

Override this method if the node has children.
Returns Generator for detail tree child names.
Return type generator yielding strs

get_detail_child_nodes(*edge_filter=(True, True)*)
Generator which yields all children of this item in the detail view (by default, all acyclic and active ones).

Override this method if the node has children.
Parameters **edge_filter** (`EdgeFilter` or `type(None)`) – The edge type to include.
Returns Generator for detail tree child nodes.
Return type generator yielding `DetailNodes`

get_global_child_nodes(*edge_filter=(True, True)*)
Returns chunks without parent.

inspect(*stream*)
Quickly checks whether the stream appears to contain cgf data, and read the cgf header and chunk table. Resets stream to original position.

Call this function if you only need to inspect the header and chunk table.
Parameters **stream** (`file`) – The file to inspect.

inspect_version_only(*stream*)
This function checks the version only, and is faster than the usual inspect function (which reads the full chunk table). Sets the L{header} and L{game} instance variables if the stream contains a valid cgf file.

Call this function if you simply wish to check that a file is a cgf file without having to parse even the header.
Raises **ValueError** – If the stream does not contain a cgf file.
Parameters **stream** (`file`) – The stream from which to read.

read(*stream*)
Read a cgf file. Does not reset stream position.
Parameters **stream** (`file`) – The stream from which to read.

replace_global_node(*oldbranch*, *newbranch*, *edge_filter=(True, True)*)
Replace a particular branch in the graph.

update_versions()
Update L{versions} for the given chunks and game.

write(*stream*)
Write a cgf file. The L{header} and L{chunk_table} are recalculated from L{chunks}. Returns number of padding bytes written (this is for debugging purposes only).
Parameters **stream** (*`file`*) – The stream to which to write.
Returns Number of padding bytes written.

class DataStreamChunk(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.formats.cgf._DataStreamChunk`, `object`

apply_scale(*scale*)
Apply scale factor on data.

class ExportFlagsChunk(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.formats.cgf.Chunk`

Export information.

class FRGB(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.object_models.xml.struct_.StructBase`

R32G32B32 (float).

class Face(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A mesh face.

class FileOffset(***kwargs*)
Bases: pyffi.object_models.common.Int

Points to a position in a file.

class FileSignature(***kwargs*)
Bases: pyffi.object_models.xml.basic.BasicBase

The CryTek file signature with which every cgf file starts.

get_hash(*data=None*)
Return a hash value for the signature.
Returns An immutable object that can be used as a hash.

get_size(*data=None*)
Return number of bytes that the signature occupies in a file.
Returns Number of bytes.

get_value()
Get signature.
Returns The signature.

read(*stream*, *data*)
Read signature from stream.
Parameters **stream** (*file*) – The stream to read from.

set_value(*value*)
Not implemented.

write(*stream*, *data*)
Write signature to stream.
Parameters **stream** (*file*) – The stream to read from.

class FileType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

An unsigned 32-bit integer, describing the file type.

class GeomNameListChunk(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Obsolete, not decoded.

class Header(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

The CGF header.

class IRGB(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

R8G8B8.

class IRGBA(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

R8G8B8A8.

class InitialPosMatrix(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A bone initial position matrix.

class MRMChunk(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Obsolete, not decoded.

class Matrix33(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.cgf._Matrix33, object

as_list()
Return matrix as 3x3 list.

as_tuple()
Return matrix as 3x3 tuple.

get_copy()
Return a copy of the matrix.

get_determinant()
Return determinant.

get_inverse()
Get inverse (assuming is_scale_rotation is true!).

get_scale()
Gets the scale (assuming is_scale_rotation is true!).

get_scale_quat()
Decompose matrix into scale and quaternion.

get_scale_rotation()
Decompose the matrix into scale and rotation, where scale is a float and rotation is a C{Matrix33}. Returns a pair (scale, rotation).

get_transpose()
Get transposed of the matrix.

is_identity()
Return True if the matrix is close to identity.

is_rotation()
Returns True if the matrix is a rotation matrix (a member of SO(3)).

is_scale_rotation()
Returns true if the matrix decomposes nicely into scale * rotation.

set_identity()
Set to identity matrix.

set_scale_rotation(*scale*, *rotation*)
Compose the matrix as the product of scale * rotation.

class Matrix44(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.cgf._Matrix44, object

as_list()
Return matrix as 4x4 list.

as_tuple()
Return matrix as 4x4 tuple.

get_copy()
Create a copy of the matrix.

get_inverse(*fast=True*)
Calculates inverse (fast assumes is_scale_rotation_translation is True).

get_matrix_33()
Returns upper left 3x3 part.

get_translation()
Returns lower left 1x3 part.

is_identity()
Return `True` if the matrix is close to identity.

set_identity()
Set to identity matrix.

set_matrix_33(*m*)
Sets upper left 3x3 part.

set_rows(*row0*, *row1*, *row2*, *row3*)
Set matrix from rows.

set_translation(*translation*)
Returns lower left 1x3 part.

sup_norm()
Calculate supremum norm of matrix (maximum absolute value of all entries).

class MtlListChunk(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.object_models.xml.struct_.StructBase`

Obsolete, not decoded.

class PatchMeshChunk(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.object_models.xml.struct_.StructBase`

Obsolete, not decoded.

class Ptr(***kwargs*)
Bases: `pyffi.formats.cgf.Ref`

Reference to a chunk, down the hierarchy.

get_refs(*data=None*)
Ptr does not point down, so get_refs returns empty list.
Returns C{[]}

class Quat(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.object_models.xml.struct_.StructBase`

A quaternion (x,y,z,w).

class Ref(***kwargs*)
Bases: `pyffi.object_models.xml.basic.BasicBase`

Reference to a chunk, up the hierarchy.

fix_links(*data*)
Resolve chunk index into a chunk.
Keyword Arguments **block_dct** – Dictionary mapping block index to block.

get_hash(*data=None*)
Return a hash value for the chunk referred to.
Returns An immutable object that can be used as a hash.

get_links(*data=None*)
Return the chunk reference.
Returns Empty list if no reference, or single item list containing the reference.

get_refs(*data=None*)
Return the chunk reference.
Returns Empty list if no reference, or single item list containing the reference.

get_size(*data=None*)
Return number of bytes this type occupies in a file.
Returns Number of bytes.

get_value()
Get chunk being referred to.
Returns The chunk being referred to.

read(*stream*, *data*)
Read chunk index.
Parameters **stream** (*file*) – The stream to read from.

set_value(*value*)
Set chunk reference.
Parameters **value** (*L{CgfFormat.Chunk}*) – The value to assign.

write(*stream*, *data*)
Write chunk index.
Parameters **stream** (*file*) – The stream to write to.

class SencePropsChunk(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Not decoded. Nowhere used?

class SizedString(***kwargs*)
Bases: pyffi.object_models.xml.basic.BasicBase, pyffi.object_models.editable.EditableLineEdit

Basic type for strings. The type starts with an unsigned int which describes the length of the string.

```
>>> from tempfile import TemporaryFile
>>> f = TemporaryFile()
>>> from pyffi.object_models import FileFormat
>>> data = FileFormat.Data()
>>> s = SizedString()
>>> if f.write('\x07\x00\x00\x00abcdefg'.encode("ascii")): pass # ignore␣
↪result for py3k
>>> if f.seek(0): pass # ignore result for py3k
>>> s.read(f, data)
>>> str(s)
'abcdefg'
>>> if f.seek(0): pass # ignore result for py3k
>>> s.set_value('Hi There')
>>> s.write(f, data)
>>> if f.seek(0): pass # ignore result for py3k
>>> m = SizedString()
>>> m.read(f, data)
>>> str(m)
'Hi There'
```

get_hash(*data=None*)
Return a hash value for this string.

Returns An immutable object that can be used as a hash.

get_size(*data=None*)
Return number of bytes this type occupies in a file.
Returns Number of bytes.

get_value()
Return the string.
Returns The stored string.

read(*stream*, *data*)
Read string from stream.
Parameters **stream** (*file*) – The stream to read from.

set_value(*value*)
Set string to C{value}.
Parameters **value** (*str*) – The value to assign.

write(*stream*, *data*)
Write string to stream.
Parameters **stream** (*file*) – The stream to write to.

String
alias of pyffi.object_models.common.ZString

class String128(***kwargs*)
Bases: pyffi.object_models.common.FixedString

String of fixed length 128.

class String16(***kwargs*)
Bases: pyffi.object_models.common.FixedString

String of fixed length 16.

class String256(***kwargs*)
Bases: pyffi.object_models.common.FixedString

String of fixed length 256.

class String32(***kwargs*)
Bases: pyffi.object_models.common.FixedString

String of fixed length 32.

class String64(***kwargs*)
Bases: pyffi.object_models.common.FixedString

String of fixed length 64.

class Tangent(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Tangents. Divide each component by 32767 to get the actual value.

class UV(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Texture coordinate.

class UVFace(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A texture face (vertex indices).

class UnknownAAFC0005Chunk(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.cgf.Chunk

Unknown. An extra block written by the XSI exporter.

class Vector3(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.cgf._Vector3, object

bool
alias of pyffi.object_models.common.Bool

byte
alias of pyffi.object_models.common.Byte

char
alias of pyffi.object_models.common.Char

float
alias of pyffi.object_models.common.Float

int
alias of pyffi.object_models.common.Int

short
alias of pyffi.object_models.common.Short

ubyte
alias of pyffi.object_models.common.UByte

uint
alias of pyffi.object_models.common.UInt

ushort
alias of pyffi.object_models.common.UShort

static version_number(*version_str*)
Converts version string into an integer.

Parameters **version_str** (*str*) – The version string.

Returns A version integer.

```
>>> hex(CgfFormat.version_number('744'))
'0x744'
```

Regression tests

Read a CGF file

```
>>> # get file version and file type, and read cgf file
>>> from os.path import dirname
>>> dirpath = __file__
>>> for i in range(4): #recurse up to root repo dir
...     dirpath = dirname(dirpath)
>>> repo_root = dirpath
>>> format_root = os.path.join(repo_root, 'tests', 'formats', 'cgf')
>>> stream = open(os.path.join(format_root, 'test.cgf'), 'rb')
>>> data = CgfFormat.Data()
>>> # read chunk table only
```

(continues on next page)

(continued from previous page)

```
>>> data.inspect(stream)
>>> # check chunk types
>>> list(chunktype.__name__ for chunktype in data.chunk_table.get_chunk_types())
['SourceInfoChunk', 'TimingChunk']
>>> data.chunks # no chunks yet
[]
>>> # read full file
>>> data.read(stream)
>>> # get all chunks
>>> for chunk in data.chunks:
...     print(chunk)
<class '...SourceInfoChunk'> instance at ...
* source_file : <None>
* date : Fri Sep 28 22:40:44 2007
* author : blender@BLENDER

<class '...TimingChunk'> instance at ...
* secs_per_tick : 0.0002083333...
* ticks_per_frame : 160
* global_range :
    <class '...RangeEntity'> instance at ...
    * name : GlobalRange
    * start : 0
    * end : 100
* num_sub_ranges : 0
```

Parse all CGF files in a directory tree

```
>>> for stream, data in CgfFormat.walkData(format_root):
...     try:
...         # the replace call makes the doctest also pass on windows
...         os_path = stream.name
...         split = (os_path.split(os.sep))[-4:]
...         rejoin = os.path.join(*split).replace(os.sep, "/")
...         print("reading %s" % rejoin)
...         data.read(stream)
...     except Exception:
...         print("Warning: read failed due corrupt file, corrupt format description,␣
↪or bug.")
...     print(len(data.chunks))
...     # do something with the chunks
...     for chunk in data.chunks:
...         chunk.apply_scale(2.0)
reading tests/formats/cgf/invalid.cgf
Warning: read failed due corrupt file, corrupt format description, or bug.
0
reading tests/formats/cgf/monkey.cgf
14
reading tests/formats/cgf/test.cgf
2
reading tests/formats/cgf/vcols.cgf
6
```

Create a CGF file from scratch

```
>>> from pyffi.formats.cgf import CgfFormat
>>> node1 = CgfFormat.NodeChunk()
>>> node1.name = "hello"
>>> node2 = CgfFormat.NodeChunk()
>>> node1.num_children = 1
>>> node1.children.update_size()
>>> node1.children[0] = node2
>>> node2.name = "world"
>>> from tempfile import TemporaryFile
>>> stream = TemporaryFile()
>>> data = CgfFormat.Data() # default is far cry
>>> data.chunks = [node1, node2]
>>> # note: write returns number of padding bytes
>>> data.write(stream)
0
>>> # py3k returns 0 on seek; this hack removes return code from doctest
>>> if stream.seek(0): pass
>>> data.inspect_version_only(stream)
>>> hex(data.header.version)
'0x744'
>>> data.read(stream)
>>> # get all chunks
>>> for chunk in data.chunks:
...     print(chunk)
<class 'pyffi.formats.cgf.NodeChunk'> instance at 0x...
* name : hello
* object : None
* parent : None
* num_children : 1
* material : None
* is_group_head : False
* is_group_member : False
* reserved_1 :
    <class 'pyffi.object_models.xml.array.Array'> instance at 0x...
    0: 0
    1: 0
* transform :
    [  0.000  0.000  0.000  0.000 ]
    [  0.000  0.000  0.000  0.000 ]
    [  0.000  0.000  0.000  0.000 ]
    [  0.000  0.000  0.000  0.000 ]
* pos : [  0.000  0.000  0.000 ]
* rot :
    <class 'pyffi.formats.cgf.Quat'> instance at 0x...
    * x : 0.0
    * y : 0.0
    * z : 0.0
    * w : 0.0
* scl : [  0.000  0.000  0.000 ]
* pos_ctrl : None
* rot_ctrl : None
* scl_ctrl : None
* property_string : <None>
* children :
    <class 'pyffi.object_models.xml.array.Array'> instance at 0x...
```

(continues on next page)

(continued from previous page)

```
    0: <class 'pyffi.formats.cgf.NodeChunk'> instance at 0x...

<class 'pyffi.formats.cgf.NodeChunk'> instance at 0x...
* name : world
* object : None
* parent : None
* num_children : 0
* material : None
* is_group_head : False
* is_group_member : False
* reserved_1 :
    <class 'pyffi.object_models.xml.array.Array'> instance at 0x...
    0: 0
    1: 0
* transform :
    [  0.000  0.000  0.000  0.000 ]
    [  0.000  0.000  0.000  0.000 ]
    [  0.000  0.000  0.000  0.000 ]
    [  0.000  0.000  0.000  0.000 ]
* pos : [  0.000  0.000  0.000 ]
* rot :
    <class 'pyffi.formats.cgf.Quat'> instance at 0x...
    * x : 0.0
    * y : 0.0
    * z : 0.0
    * w : 0.0
* scl : [  0.000  0.000  0.000 ]
* pos_ctrl : None
* rot_ctrl : None
* scl_ctrl : None
* property_string : <None>
* children : <class 'pyffi.object_models.xml.array.Array'> instance at 0x...
```

pyffi.formats.dae — COLLADA (.dae)

Warning: This module is not yet fully implemented, and is certainly not yet useful in its current state.

Implementation

class pyffi.formats.dae.**DaeFormat**

Bases: pyffi.object_models.xsd.FileFormat

This class implements the DAE format.

class Data(*version=17039616*)

Bases: pyffi.object_models.Data

A class to contain the actual collada data.

getVersion()

Get the collada version, as integer (for instance, 1.4.1 would be 0x01040100).

Returns The version, as integer.

inspect(*stream*)
Quickly checks whether the stream appears to contain collada data. Resets stream to original position. If the stream turns out to be collada, L{getVersion} is guaranteed to return the version.

Call this function if you simply wish to check that a file is a collada file without having to parse it completely.

Parameters **stream** (*file*) – The file to inspect.
Returns `True` if stream is collada, `False` otherwise.

read(*stream*)
Read collada data from stream.

Parameters **stream** (*file*) – The file to read from.

write(*stream*)
Write collada data to stream.

Parameters **stream** (*file*) – The file to write to.

Regression tests

Create a DAE file

```
>>> daedata = DaeFormat.Data()
>>> print(daedata.collada)
<...Collada object at ...>
```

Read a DAE file

```
>>> from os.path import dirname
>>> dirpath = __file__
>>> for i in range(4): #recurse up to root repo dir
...     dirpath = dirname(dirpath)
>>> repo_root = dirpath
>>> format_root = os.path.join(repo_root, 'tests', 'formats', 'dae')
>>> # check and read dae file
>>> stream = open(os.path.join(format_root, 'cube.dae'), 'rb')
>>> daedata = DaeFormat.Data()
>>> daedata.read(stream)
Traceback (most recent call last):
    ...
NotImplementedError
>>> # get DAE file root element
>>> #print(daedata.getRootElement())
>>> stream.close()
```

Parse all DAE files in a directory tree

```
>>> for stream, data in DaeFormat.walkData(format_root):
...     try:
...         # the replace call makes the doctest also pass on windows
...         os_path = stream.name
...         split = (os_path.split(os.sep))[-4:]
...         rejoin = os.path.join(*split).replace(os.sep, "/")
...         print("reading %s" % rejoin)
...         data.read(stream)
...     except Exception:
...         print("Warning: read failed due corrupt file, corrupt format description,␣
↪or bug.")
reading tests/formats/dae/cube.dae
Warning: read failed due corrupt file, corrupt format description, or bug.
```

Create a DAE file from scratch and write to file

```
>>> daedata = DaeFormat.Data()
>>> from tempfile import TemporaryFile
>>> stream = TemporaryFile()
>>> daedata.write(stream)
Traceback (most recent call last):
    ...
NotImplementedError
```

pyffi.formats.dds — DirectDraw Surface (.dds)

Implementation

class pyffi.formats.dds.**DdsFormat**

Bases: pyffi.object_models.xml.FileFormat

This class implements the DDS format.

class Data(*version=150994944*)

Bases: pyffi.object_models.Data

A class to contain the actual dds data.

get_detail_child_names(*edge_filter=(True, True)*)

Generator which yields all child names of this item in the detail view.

Override this method if the node has children.

Returns Generator for detail tree child names.

Return type generator yielding strs

get_detail_child_nodes(*edge_filter=(True, True)*)

Generator which yields all children of this item in the detail view (by default, all acyclic and active ones).

Override this method if the node has children.

Parameters **edge_filter** (EdgeFilter or type(None)) – The edge type to include.

Returns Generator for detail tree child nodes.

Return type generator yielding DetailNodes

inspect(*stream*)
Quickly checks if stream contains DDS data, and reads the header.
Parameters **stream** (*file*) – The stream to inspect.

inspect_quick(*stream*)
Quickly checks if stream contains DDS data, and gets the version, by looking at the first 8 bytes.
Parameters **stream** (*file*) – The stream to inspect.

read(*stream*, *verbose=0*)
Read a dds file.
Parameters
- **stream** (file) – The stream from which to read.
- **verbose** (int) – The level of verbosity.

write(*stream*, *verbose=0*)
Write a dds file.
Parameters
- **stream** (file) – The stream to which to write.
- **verbose** (int) – The level of verbosity.

class FourCC(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

An unsigned 32-bit integer, describing the compression type.

class HeaderString(***kwargs*)
Bases: pyffi.object_models.xml.basic.BasicBase

Basic type which implements the header of a DDS file.

get_detail_display()
Return an object that can be used to display the instance.

get_hash(*data=None*)
Return a hash value for this value.
Returns An immutable object that can be used as a hash.

get_size(*data=None*)
Return number of bytes the header string occupies in a file.
Returns Number of bytes.

read(*stream*, *data*)
Read header string from stream and check it.
Parameters **stream** (*file*) – The stream to read from.

write(*stream*, *data*)
Write the header string to stream.
Parameters **stream** (*file*) – The stream to write to.

PixelData
alias of pyffi.object_models.common.UndecodedData

byte
alias of pyffi.object_models.common.Byte

char
alias of pyffi.object_models.common.Char

float
alias of pyffi.object_models.common.Float

int
alias of `pyffi.object_models.common.Int`

short
alias of `pyffi.object_models.common.Short`

ubyte
alias of `pyffi.object_models.common.UByte`

uint
alias of `pyffi.object_models.common.UInt`

ushort
alias of `pyffi.object_models.common.UShort`

static version_number(*version_str*)
Converts version string into an integer.

Parameters **version_str** (*str*) – The version string.

Returns A version integer.

```
>>> hex(DdsFormat.version_number('DX10'))
'0xa000000'
```

Regression tests

Read a DDS file

```
>>> # check and read dds file
>>> from os.path import dirname
>>> dirpath = __file__
>>> for i in range(4): #recurse up to root repo dir
...     dirpath = dirname(dirpath)
>>> repo_root = dirpath
>>> format_root = os.path.join(repo_root, 'tests', 'formats', 'dds')
>>> file = os.path.join(format_root, 'test.dds')
>>> stream = open(file, 'rb')
>>> data = DdsFormat.Data()
>>> data.inspect(stream)
>>> data.header.pixel_format.size
32
>>> data.header.height
20
>>> data.read(stream)
>>> len(data.pixeldata.get_value())
888
```

Parse all DDS files in a directory tree

```
>>> for stream, data in DdsFormat.walkData(format_root):
...     try:
...         # the replace call makes the doctest also pass on windows
...         os_path = stream.name
...         split = (os_path.split(os.sep))[-4:]
...         rejoin = os.path.join(*split).replace(os.sep, "/")
...         print("reading %s" % rejoin)
...     except Exception:
...         print(
...             "Warning: read failed due corrupt file,"
...             " corrupt format description, or bug.")
reading tests/formats/dds/test.dds
```

Create a DDS file from scratch and write to file

```
>>> data = DdsFormat.Data()
>>> from tempfile import TemporaryFile
>>> stream = TemporaryFile()
>>> data.write(stream)
```

Get list of versions

```
>>> for vnum in sorted(DdsFormat.versions.values()):
...     print('0x%08X' % vnum)
0x09000000
0x0A000000
```

pyffi.formats.egm — EGM (.egm)

An .egm file contains facial shape modifiers, that is, morphs that modify static properties of the face, such as nose size, chin shape, and so on.

Implementation

class pyffi.formats.egm.**EgmFormat**

Bases: pyffi.object_models.xml.FileFormat

This class implements the EGM format.

class Data(*version=2*, *num_vertices=0*)

Bases: pyffi.object_models.Data

A class to contain the actual egm data.

add_asym_morph()

Add an asymmetric morph, and return it.

add_sym_morph()

Add a symmetric morph, and return it.

apply_scale(*scale*)
Apply scale factor to all morphs.

get_detail_child_names(*edge_filter=(True, True)*)
Generator which yields all child names of this item in the detail view.

Override this method if the node has children.
Returns Generator for detail tree child names.
Return type generator yielding `strs`

get_detail_child_nodes(*edge_filter=(True, True)*)
Generator which yields all children of this item in the detail view (by default, all acyclic and active ones).

Override this method if the node has children.
Parameters **edge_filter** (`EdgeFilter` or `type(None)`) – The edge type to include.
Returns Generator for detail tree child nodes.
Return type generator yielding `DetailNodes`

get_global_child_nodes(*edge_filter=(True, True)*)
Generator which yields all children of this item in the global view, of given edge type (default is edges of type 0).

Override this method.
Returns Generator for global node children.

inspect(*stream*)
Quickly checks if stream contains EGM data, and reads the header.
Parameters **stream** (*file*) – The stream to inspect.

inspect_quick(*stream*)
Quickly checks if stream contains EGM data, and gets the version, by looking at the first 8 bytes.
Parameters **stream** (*file*) – The stream to inspect.

read(*stream*)
Read a egm file.
Parameters **stream** (`file`) – The stream from which to read.

write(*stream*)
Write a egm file.
Parameters **stream** (`file`) – The stream to which to write.

class FileSignature(***kwargs*)
Bases: `pyffi.object_models.xml.basic.BasicBase`

Basic type which implements the header of a EGM file.

get_detail_display()
Return an object that can be used to display the instance.

get_hash(*data=None*)
Return a hash value for this value.
Returns An immutable object that can be used as a hash.

get_size(*data=None*)
Return number of bytes the header string occupies in a file.
Returns Number of bytes.

read(*stream*, *data*)
Read header string from stream and check it.
Parameters **stream** (*file*) – The stream to read from.

write(*stream*, *data*)
Write the header string to stream.
Parameters **stream** (*file*) – The stream to write to.

class FileVersion(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.basic.BasicBase

get_detail_display()
Return an object that can be used to display the instance.

get_hash(*data=None*)
Returns a hash value (an immutable object) that can be used to identify the object uniquely.

get_size(*data=None*)
Returns size of the object in bytes.

get_value()
Return object value.

read(*stream*, *data*)
Read object from file.

set_value(*value*)
Set object value.

write(*stream*, *data*)
Write object to file.

class MorphRecord(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.egm._MorphRecord, object

```
>>> # create morph with 3 vertices.
>>> morph = EgmFormat.MorphRecord(argument=3)
>>> morph.set_relative_vertices(
...     [(3, 5, 2), (1, 3, 2), (-9, 3, -1)])
>>> # scale should be 9/32768.0 = 0.0002746...
>>> morph.scale
0.0002746...
>>> for vert in morph.get_relative_vertices():
...     print([int(1000 * x + 0.5) for x in vert])
[3000, 5000, 2000]
[1000, 3000, 2000]
[-8999, 3000, -999]
```

apply_scale(*scale*)
Apply scale factor to data.

```
>>> # create morph with 3 vertices.
>>> morph = EgmFormat.MorphRecord(argument=3)
>>> morph.set_relative_vertices(
...     [(3, 5, 2), (1, 3, 2), (-9, 3, -1)])
>>> morph.apply_scale(2)
>>> for vert in morph.get_relative_vertices():
...     print([int(1000 * x + 0.5) for x in vert])
[6000, 10000, 4000]
[2000, 6000, 4000]
[-17999, 6000, -1999]
```

byte
alias of pyffi.object_models.common.Byte

char
alias of pyffi.object_models.common.Char

float
alias of pyffi.object_models.common.Float

int
alias of pyffi.object_models.common.Int

short
alias of pyffi.object_models.common.Short

ubyte
alias of pyffi.object_models.common.UByte

uint
alias of pyffi.object_models.common.UInt

ushort
alias of pyffi.object_models.common.UShort

static version_number(*version_str*)
Converts version string into an integer.

Parameters **version_str** (*str*) – The version string.

Returns A version integer.

```
>>> EgmFormat.version_number('002')
2
>>> EgmFormat.version_number('XXX')
-1
```

Regression tests

Read a EGM file

```
>>> # check and read egm file
>>> from os.path import dirname
>>> dirpath = __file__
>>> for i in range(4): #recurse up to root repo dir
...     dirpath = dirname(dirpath)
>>> repo_root = dirpath
>>> format_root = os.path.join(repo_root, 'tests', 'formats', 'egm')
>>> file = os.path.join(format_root, 'mmouthxivilai.egm')
>>> stream = open(file, 'rb')
>>> data = EgmFormat.Data()
>>> data.inspect_quick(stream)
>>> data.version
2
>>> data.inspect(stream)
>>> data.header.num_vertices
89
>>> data.header.num_sym_morphs
50
>>> data.header.num_asym_morphs
30
>>> data.header.time_date_stamp
```

(continues on next page)

(continued from previous page)

```
2001060901
>>> data.read(stream)
>>> data.sym_morphs[0].vertices[0].x
17249
```

Parse all EGM files in a directory tree

```
>>> for stream, data in EgmFormat.walkData(format_root):
...     try:
...         # the replace call makes the doctest also pass on windows
...         os_path = stream.name
...         split = (os_path.split(os.sep))[-4:]
...         rejoin = os.path.join(*split).replace(os.sep, "/")
...         print("reading %s" % rejoin)
...     except Exception:
...         print(
...             "Warning: read failed due corrupt file,"
...             " corrupt format description, or bug.")
reading tests/formats/egm/mmouthxivilai.egm
```

Create an EGM file from scratch and write to file

```
>>> data = EgmFormat.Data(num_vertices=10)
>>> data.header.num_vertices
10
>>> morph = data.add_sym_morph()
>>> len(morph.vertices)
10
>>> morph.scale = 0.4
>>> morph.vertices[0].z = 123
>>> morph.vertices[9].x = -30000
>>> morph = data.add_asym_morph()
>>> morph.scale = 2.3
>>> morph.vertices[3].z = -5
>>> morph.vertices[4].x = 99
>>> from tempfile import TemporaryFile
>>> stream = TemporaryFile()
>>> data.write(stream)
```

pyffi.formats.egt — EGT (.egt)

An .egt file contains texture tones for the different races.

Implementation

class pyffi.formats.egt.**EgtFormat**
Bases: pyffi.object_models.xml.FileFormat

This class implements the EGT format.

Data
alias of *Header*

class FileSignature(***kwargs*)
Bases: pyffi.object_models.xml.basic.BasicBase

Basic type which implements the header of a EGT file.

get_detail_display()
Return an object that can be used to display the instance.

get_hash(*data=None*)
Return a hash value for this value.
Returns An immutable object that can be used as a hash.

get_size(*data=None*)
Return number of bytes the header segtng occupies in a file.
Returns Number of bytes.

read(*stream*, *data*)
Read header string from stream and check it.
Parameters **stream** (*file*) – The stream to read from.

write(*stream*, *data*)
Write the header segtng to stream.
Parameters **stream** (*file*) – The stream to write to.

class FileVersion(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.basic.BasicBase

get_detail_display()
Return an object that can be used to display the instance.

get_hash(*data=None*)
Returns a hash value (an immutable object) that can be used to identify the object uniquely.

get_size(*data=None*)
Returns size of the object in bytes.

get_value()
Return object value.

read(*stream*, *data*)
Read object from file.

set_value(*value*)
Set object value.

write(*stream*, *data*)
Write object to file.

class Header(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.egt._Header, pyffi.object_models.Data

A class to contain the actual egt data.

get_global_child_nodes(*edge_filter=(True, True)*)
Generator which yields all children of this item in the global view, of given edge type (default is edges of type 0).

Override this method.
Returns Generator for global node children.

inspect(*stream*)
Quickly checks if stream contains EGT data, and reads everything up to the arrays.
Parameters **stream** (*file*) – The stream to inspect.

inspect_quick(*stream*)
Quickly checks if stream contains EGT data, by looking at the first 8 bytes. Reads the signature and the version.
Parameters **stream** (*file*) – The stream to inspect.

read(*stream*)
Read a egt file.
Parameters **stream** (file) – The stream from which to read.

write(*stream*)
Write a egt file.
Parameters **stream** (file) – The stream to which to write.

byte
alias of pyffi.object_models.common.Byte

char
alias of pyffi.object_models.common.Char

float
alias of pyffi.object_models.common.Float

int
alias of pyffi.object_models.common.Int

short
alias of pyffi.object_models.common.Short

ubyte
alias of pyffi.object_models.common.UByte

uint
alias of pyffi.object_models.common.UInt

ushort
alias of pyffi.object_models.common.UShort

static version_number(*version_str*)
Converts version segtng into an integer.

Parameters **version_str** (*str*) – The version segtng.

Returns A version integer.

```
>>> EgtFormat.version_number('003')
3
```

(continues on next page)

(continued from previous page)

```
>>> EgtFormat.version_number('XXX')
-1
```

Regression tests

Read a EGT file

```
>>> # check and read egt file
>>> from os.path import dirname
>>> dirpath = __file__
>>> for i in range(4): #recurse up to root repo dir
...     dirpath = dirname(dirpath)
>>> repo_root = dirpath
>>> format_root = os.path.join(repo_root, 'tests', 'formats', 'egt')
>>> file = os.path.join(format_root, 'test.egt')
>>> stream = open(file, 'rb')
>>> data = EgtFormat.Data()
>>> data.inspect(stream)
>>> # do some stuff with header?
>>> data.read(stream)
>>> # do more stuff?
```

Parse all EGT files in a directory tree

```
>>> for stream, data in EgtFormat.walkData(format_root):
...     try:
...         # the replace call makes the doctest also pass on windows
...         os_path = stream.name
...         split = (os_path.split(os.sep))[-4:]
...         rejoin = os.path.join(*split).replace(os.sep, "/")
...         print("reading %s" % rejoin)
...     except Exception:
...         print(
...             "Warning: read failed due corrupt file,"
...             " corrupt format description, or bug.")
reading tests/formats/egt/test.egt
```

Create an EGT file from scratch and write to file

```
>>> data = EgtFormat.Data()
>>> from tempfile import TemporaryFile
>>> stream = TemporaryFile()
>>> data.write(stream)
```

pyffi.formats.esp — Elder Scrolls plugin/master/save files (.esp, .esm, and .ess)

Implementation

class pyffi.formats.esp.**EspFormat**
Bases: pyffi.object_models.xml.FileFormat

This class implements the ESP format.

class Data
Bases: pyffi.object_models.Data

A class to contain the actual esp data.

get_detail_child_names(*edge_filter=(True, True)*)
Generator which yields all child names of this item in the detail view.

Override this method if the node has children.
Returns Generator for detail tree child names.
Return type generator yielding strs

get_detail_child_nodes(*edge_filter=(True, True)*)
Generator which yields all children of this item in the detail view (by default, all acyclic and active ones).

Override this method if the node has children.
Parameters **edge_filter** (EdgeFilter or type(None)) – The edge type to include.
Returns Generator for detail tree child nodes.
Return type generator yielding DetailNodes

get_global_child_nodes(*edge_filter=(True, True)*)
Generator which yields all children of this item in the global view, of given edge type (default is edges of type 0).

Override this method.
Returns Generator for global node children.

inspect(*stream*)
Quickly checks if stream contains ESP data, and reads the header.
Parameters **stream** (*file*) – The stream to inspect.

inspect_quick(*stream*)
Quickly checks if stream contains ESP data, and gets the version, by looking at the first 8 bytes.
Parameters **stream** (*file*) – The stream to inspect.

read(*stream*)
Read a esp file.
Parameters **stream** (file) – The stream from which to read.

write(*stream*)
Write a esp file.
Parameters **stream** (file) – The stream to which to write.

class GRUP
Bases: pyffi.formats.esp._GRUP, object

get_global_child_nodes(*edge_filter=(True, True)*)
Generator which yields all children of this item in the global view, of given edge type (default is edges of type 0).

Override this method.

Returns Generator for global node children.

read(*stream*, *data*)
Read structure from stream.

write(*stream*, *data*)
Write structure to stream.

class Record
Bases: pyffi.formats.esp._Record, object

get_global_child_nodes(*edge_filter=(True, True)*)
Generator which yields all children of this item in the global view, of given edge type (default is edges of type 0).

Override this method.
Returns Generator for global node children.

get_sub_record(*sub_record_type*)
Find first subrecord of given type.

read(*stream*, *data*)
Read structure from stream.

write(*stream*, *data*)
Write structure to stream.

class RecordType(***kwargs*)
Bases: pyffi.object_models.common.FixedString

class SubRecord(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A subrecord.

class ZString(***kwargs*)
Bases: pyffi.object_models.xml.basic.BasicBase, pyffi.object_models.editable.EditableLineEdit

String of variable length (null terminated).

```
>>> from tempfile import TemporaryFile
>>> f = TemporaryFile()
>>> s = ZString()
>>> if f.write('abcdefghijklmnopqrst\x00'.encode("ascii")): pass # b'abc...'
>>> if f.seek(0): pass # ignore result for py3k
>>> s.read(f)
>>> str(s)
'abcdefghijklmnopqrst'
>>> if f.seek(0): pass # ignore result for py3k
>>> s.set_value('Hi There!')
>>> s.write(f)
>>> if f.seek(0): pass # ignore result for py3k
>>> m = ZString()
>>> m.read(f)
>>> str(m)
'Hi There!'
```

get_hash(*data=None*)
Return a hash value for this string.
Returns An immutable object that can be used as a hash.

get_size(*data=None*)
Return number of bytes this type occupies in a file.
Returns Number of bytes.

get_value()
Return the string.
Returns The stored string.
Return type C{bytes}

read(*stream*, *data=None*)
Read string from stream.
Parameters **stream** (*file*) – The stream to read from.

set_value(*value*)
Set string to C{value}.
Parameters **value** (str (will be encoded as default) or C{bytes}) – The value to assign.

write(*stream*, *data=None*)
Write string to stream.
Parameters **stream** (*file*) – The stream to write to.

byte
alias of pyffi.object_models.common.Byte

char
alias of pyffi.object_models.common.Char

float
alias of pyffi.object_models.common.Float

int
alias of pyffi.object_models.common.Int

short
alias of pyffi.object_models.common.Short

ubyte
alias of pyffi.object_models.common.UByte

uint
alias of pyffi.object_models.common.UInt

uint64
alias of pyffi.object_models.common.UInt64

ushort
alias of pyffi.object_models.common.UShort

static version_number(*version_str*)
Converts version string into an integer.

Parameters **version_str** (*str*) – The version string.

Returns A version integer.

```
>>> hex(EspFormat.version_number('1.2'))
'0x102'
```

Regression tests

Read a ESP file

```
>>> # check and read esp file
>>> from os.path import dirname
>>> dirpath = __file__
>>> for i in range(4): #recurse up to root repo dir
...     dirpath = dirname(dirpath)
>>> repo_root = dirpath
>>> format_root = os.path.join(repo_root, 'tests', 'formats', 'esp')
>>> file = os.path.join(format_root, 'test.esp')
>>> stream = open(file, 'rb')
>>> data = EspFormat.Data()
>>> data.inspect(stream)
>>> # do some stuff with header?
>>> #data.header....
>>> data.read(stream)
>>> # do some stuff...
```

Parse all ESP files in a directory tree

```
>>> for stream, data in EspFormat.walkData(format_root):
...     try:
...         # the replace call makes the doctest also pass on windows
...         os_path = stream.name
...         split = (os_path.split(os.sep))[-4:]
...         rejoin = os.path.join(*split).replace(os.sep, "/")
...         print("reading %s" % rejoin)
...     except Exception:
...         print(
...             "Warning: read failed due corrupt file,"
...             " corrupt format description, or bug.")
reading tests/formats/esp/test.esp
```

Create an ESP file from scratch and write to file

```
>>> data = EspFormat.Data()
>>> from tempfile import TemporaryFile
>>> stream = TemporaryFile()
>>> data.write(stream)
```

pyffi.formats.kfm — NetImmerse/Gamebryo Keyframe Motion (.kfm)

Implementation

class pyffi.formats.kfm.**KfmFormat**
Bases: pyffi.object_models.xml.FileFormat

This class implements the kfm file format.

class Data(*version=33685515*)
Bases: pyffi.object_models.Data

A class to contain the actual kfm data.

get_global_child_nodes(*edge_filter=(True, True)*)
Generator which yields all children of this item in the global view, of given edge type (default is edges of type 0).

Override this method.
Returns Generator for global node children.

get_global_display()
Display the KFM file name.

inspect(*stream*)
Quick heuristic check if stream contains KFM data, by looking at the first 64 bytes. Sets version and reads header string.
Parameters **stream** (*file*) – The stream to inspect.

read(*stream*)
Read a kfm file.
Parameters **stream** (file) – The stream from which to read.

write(*stream*)
Write a kfm file.
Parameters **stream** (file) – The stream to which to write.

class FilePath(***kwargs*)
Bases: pyffi.object_models.common.SizedString

get_hash(*data=None*)
Return a hash value for this value. For file paths, the hash value is case insensitive.
Returns An immutable object that can be used as a hash.

class HeaderString(***kwargs*)
Bases: pyffi.object_models.xml.basic.BasicBase

The kfm header string.

get_detail_display()
Return an object that can be used to display the instance.

get_hash(*data=None*)
Return a hash value for this value.
Returns An immutable object that can be used as a hash.

get_size(*data=None*)
Return number of bytes the header string occupies in a file.
Returns Number of bytes.

get_value()
Return object value.

read(*stream*, *data*)
Read header string from stream and check it.
Parameters
- **stream** (*file*) – The stream to read from.
- **data** (pyffi.formats.kfm.KfmFormat.Data) – The KfmFormat.Data()

set_value(*value*)
Set object value.

static version_string(*version*)
Transforms version number into a version string.
Parameters **version** (*int*) – The version number.
Returns A version string.

```
>>> KfmFormat.HeaderString.version_string(0x0202000b)
';Gamebryo KFM File Version 2.2.0.0b'
>>> KfmFormat.HeaderString.version_string(0x01024b00)
';Gamebryo KFM File Version 1.2.4b'
```

write(*stream*, *data*)
Write the header string to stream.
Parameters
- **stream** (*file*) – The stream to write to.
- **data** (Data) – The fileformat data to use

class SizedString(***kwargs*)
Bases: pyffi.object_models.xml.basic.BasicBase, pyffi.object_models.editable.EditableLineEdit

Basic type for strings. The type starts with an unsigned int which describes the length of the string.

```
>>> from tempfile import TemporaryFile
>>> f = TemporaryFile()
>>> from pyffi.object_models import FileFormat
>>> data = FileFormat.Data()
>>> s = SizedString()
>>> if f.write('\x07\x00\x00\x00abcdefg'.encode("ascii")): pass # ignore␣
↪result for py3k
>>> if f.seek(0): pass # ignore result for py3k
>>> s.read(f, data)
>>> str(s)
'abcdefg'
>>> if f.seek(0): pass # ignore result for py3k
>>> s.set_value('Hi There')
>>> s.write(f, data)
>>> if f.seek(0): pass # ignore result for py3k
>>> m = SizedString()
>>> m.read(f, data)
>>> str(m)
'Hi There'
```

get_hash(*data=None*)
Return a hash value for this string.
Returns An immutable object that can be used as a hash.

get_size(*data=None*)
Return number of bytes this type occupies in a file.
Returns Number of bytes.

get_value()
Return the string.
Returns The stored string.

read(*stream*, *data*)
Read string from stream.
Parameters **stream** (*file*) – The stream to read from.

set_value(*value*)
Set string to C{value}.
Parameters **value** (*str*) – The value to assign.

write(*stream*, *data*)
Write string to stream.
Parameters **stream** (*file*) – The stream to write to.

TextString
alias of pyffi.object_models.common.UndecodedData

byte
alias of pyffi.object_models.common.UByte

char
alias of pyffi.object_models.common.Char

float
alias of pyffi.object_models.common.Float

int
alias of pyffi.object_models.common.Int

short
alias of pyffi.object_models.common.Short

uint
alias of pyffi.object_models.common.UInt

ushort
alias of pyffi.object_models.common.UShort

static version_number(*version_str*)
Converts version string into an integer.

Parameters **version_str** (*str*) – The version string.

Returns A version integer.

```
>>> hex(KfmFormat.version_number('1.0'))
'0x1000000'
>>> hex(KfmFormat.version_number('1.2.4b'))
'0x1024b00'
>>> hex(KfmFormat.version_number('2.2.0.0b'))
'0x202000b'
```

Regression tests

Read a KFM file

```
>>> # read kfm file
>>> from os.path import dirname
>>> dirpath = __file__
>>> for i in range(4): #recurse up to root repo dir
...     dirpath = dirname(dirpath)
>>> repo_root = dirpath
>>> files_dir = os.path.join(repo_root, 'tests', 'spells', 'kfm', 'files')
>>> file = os.path.join(files_dir, 'test.kfm')
>>> stream = open(file, 'rb')
>>> data = KfmFormat.Data()
>>> data.inspect(stream)
>>> data.read(stream)
>>> stream.close()
>>> print(data.nif_file_name.decode("ascii"))
Test.nif
>>> # get all animation file names
>>> for anim in data.animations:
...     print(anim.kf_file_name.decode("ascii"))
Test_MD_Idle.kf
Test_MD_Run.kf
Test_MD_Walk.kf
Test_MD_Die.kf
```

Parse all KFM files in a directory tree

```
>>> for stream, data in KfmFormat.walkData(files_dir):
...     try:
...         # the replace call makes the doctest also pass on windows
...         os_path = stream.name
...         split = (os_path.split(os.sep))[-5:]
...         rejoin = os.path.join(*split).replace("\\", "/")
...         print("reading %s" % rejoin)
...     except Exception:
...         print(
...             "Warning: read failed due corrupt file,"
...             " corrupt format description, or bug.")
reading tests/spells/kfm/files/invalid.kfm
reading tests/spells/kfm/files/test.kfm
```

Create a KFM model from scratch and write to file

```
>>> data = KfmFormat.Data()
>>> data.nif_file_name = "Test.nif"
>>> data.num_animations = 4
>>> data.animations.update_size()
>>> data.animations[0].kf_file_name = "Test_MD_Idle.kf"
>>> data.animations[1].kf_file_name = "Test_MD_Run.kf"
>>> data.animations[2].kf_file_name = "Test_MD_Walk.kf"
```

(continues on next page)

(continued from previous page)

```
>>> data.animations[3].kf_file_name = "Test_MD_Die.kf"
>>> from tempfile import TemporaryFile
>>> stream = TemporaryFile()
>>> data.write(stream)
>>> stream.close()
```

Get list of versions and games

```
>>> for vnum in sorted(KfmFormat.versions.values()):
...     print('0x%08X' % vnum)
0x01000000
0x01024B00
0x0200000B
0x0201000B
0x0202000B
0x0202001B
>>> for game, versions in sorted(KfmFormat.games.items(),
...                              key=lambda x: x[0]):
...     print("%s " % game + " ".join('0x%08X' % vnum for vnum in versions))
Civilization IV 0x01000000 0x01024B00 0x0200000B
Dragonica 0x0202001B
Emerge 0x0201000B 0x0202000B
Loki 0x01024B00
Megami Tensei: Imagine 0x0201000B
Oblivion 0x01024B00
Prison Tycoon 0x01024B00
Pro Cycling Manager 0x01024B00
Red Ocean 0x01024B00
Sid Meier's Railroads 0x0200000B
The Guild 2 0x01024B00
```

pyffi.formats.nif — NetImmerse/Gamebryo (.nif and .kf)

Implementation

class pyffi.formats.nif.**NifFormat**

Bases: pyffi.object_models.xml.FileFormat

This class contains the generated classes from the xml.

ARCHIVE_CLASSES = [<class 'pyffi.formats.bsa.BsaFormat'>]

class ATextureRenderData(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif._ATextureRenderData, object

save_as_dds(*stream*)

Save image as DDS file.

class AVObject(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.object_models.xml.struct_.StructBase

Used in NiDefaultAVObjectPalette.

property av_object

property name

class AbstractAdditionalGeometryData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

class AdditionalDataBlock(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

property block_offsets

property block_size

property data

property data_sizes

property has_data

property num_blocks

property num_data

class AdditionalDataInfo(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

property block_index

property channel_offset

property data_type

property num_channel_bytes

property num_channel_bytes_per_element

property num_total_bytes_per_element

property unknown_byte_1

class AlphaFormat(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

An unsigned 32-bit integer, describing how transparency is handled in a texture.

ALPHA_BINARY = 1

ALPHA_DEFAULT = 3

ALPHA_NONE = 0

ALPHA_SMOOTH = 2

class AnimationType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

Animation type used on this position. This specifies the function of this position.

Lean = 4

Sit = 1

Sleep = 2

class ApplyMode(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

An unsigned 32-bit integer, describing the apply mode of a texture.

APPLY_DECAL = 1

APPLY_HILIGHT = 3

APPLY_HILIGHT2 = 4

APPLY_MODULATE = 2

APPLY_REPLACE = 0

class ArkTexture(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A texture reference used by NiArkTextureExtraData.

property texture_name

property texturing_property

property unknown_bytes

property unknown_int_3

property unknown_int_4

class AvoidNode(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

Morrowind specific?

class BSAnimNotes(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Bethesda-specific node.

property unknown_short_1

class BSBehaviorGraphExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Links a nif with a Havok Behavior .hkx animation file

property behaviour_graph_file

property controls_base_skeleton

class BSBlastNode(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

Bethesda-Specific node.

property unknown_byte_1

property unknown_short_2

class BSBoneLODExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Unknown

property bone_l_o_d_count

property bone_l_o_d_info

class BSBound(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._BSBound, object

apply_scale(*scale*)
Scale data.

class BSDamageStage(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

Bethesda-Specific node.

property unknown_byte_1

property unknown_short_2

class BSDebrisNode(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

Bethesda-Specific node.

property unknown_byte_1

property unknown_short_2

class BSDecalPlacementVectorExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Bethesda-specific node. (for dynamic decal projection?)

property num_vector_blocks

property unknown_float_1

property vector_blocks

class BSDismemberBodyPartType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

Biped bodypart data used for visibility control of triangles. Options are Fallout 3, except where marked for Skyrim (uses SBP prefix) Skyrim BP names are listed only for vanilla names, different creatures have different defnitions for naming.

BP_BRAIN = 13

BP_HEAD = 1

BP_HEAD2 = 2

BP_LEFTARM = 3

BP_LEFTARM2 = 4

BP_LEFTLEG = 7

BP_LEFTLEG2 = 8

BP_LEFTLEG3 = 9

BP_RIGHTARM = 5

BP_RIGHTARM2 = 6

BP_RIGHTLEG = 10

BP_RIGHTLEG2 = 11

BP_RIGHTLEG3 = 12

BP_SECTIONCAP_BRAIN = 113

BP_SECTIONCAP_HEAD = 101

BP_SECTIONCAP_HEAD2 = 102

BP_SECTIONCAP_LEFTARM = 103

```
BP_SECTIONCAP_LEFTARM2 = 104
BP_SECTIONCAP_LEFTLEG = 107
BP_SECTIONCAP_LEFTLEG2 = 108
BP_SECTIONCAP_LEFTLEG3 = 109
BP_SECTIONCAP_RIGHTARM = 105
BP_SECTIONCAP_RIGHTARM2 = 106
BP_SECTIONCAP_RIGHTLEG = 110
BP_SECTIONCAP_RIGHTLEG2 = 111
BP_SECTIONCAP_RIGHTLEG3 = 112
BP_TORSO = 0
BP_TORSOCAP_BRAIN = 213
BP_TORSOCAP_HEAD = 201
BP_TORSOCAP_HEAD2 = 202
BP_TORSOCAP_LEFTARM = 203
BP_TORSOCAP_LEFTARM2 = 204
BP_TORSOCAP_LEFTLEG = 207
BP_TORSOCAP_LEFTLEG2 = 208
BP_TORSOCAP_LEFTLEG3 = 209
BP_TORSOCAP_RIGHTARM = 205
BP_TORSOCAP_RIGHTARM2 = 206
BP_TORSOCAP_RIGHTLEG = 210
BP_TORSOCAP_RIGHTLEG2 = 211
BP_TORSOCAP_RIGHTLEG3 = 212
BP_TORSOSECTION_BRAIN = 13000
BP_TORSOSECTION_HEAD = 1000
BP_TORSOSECTION_HEAD2 = 2000
BP_TORSOSECTION_LEFTARM = 3000
BP_TORSOSECTION_LEFTARM2 = 4000
BP_TORSOSECTION_LEFTLEG = 7000
BP_TORSOSECTION_LEFTLEG2 = 8000
BP_TORSOSECTION_LEFTLEG3 = 9000
BP_TORSOSECTION_RIGHTARM = 5000
BP_TORSOSECTION_RIGHTARM2 = 6000
BP_TORSOSECTION_RIGHTLEG = 10000
BP_TORSOSECTION_RIGHTLEG2 = 11000
BP_TORSOSECTION_RIGHTLEG3 = 12000
```

```
SBP_130_HEAD = 130
SBP_131_HAIR = 131
SBP_141_LONGHAIR = 141
SBP_142_CIRCLET = 142
SBP_143_EARS = 143
SBP_150_DECAPITATEDHEAD = 150
SBP_230_HEAD = 230
SBP_30_HEAD = 30
SBP_31_HAIR = 31
SBP_32_BODY = 32
SBP_33_HANDS = 33
SBP_34_FOREARMS = 34
SBP_35_AMULET = 35
SBP_36_RING = 36
SBP_37_FEET = 37
SBP_38_CALVES = 38
SBP_39_SHIELD = 39
SBP_40_TAIL = 40
SBP_41_LONGHAIR = 41
SBP_42_CIRCLET = 42
SBP_43_EARS = 43
SBP_44_DRAGON_BLOODHEAD_OR_MOD_MOUTH = 44
SBP_45_DRAGON_BLOODWINGL_OR_MOD_NECK = 45
SBP_46_DRAGON_BLOODWINGR_OR_MOD_CHEST_PRIMARY = 46
SBP_47_DRAGON_BLOODTAIL_OR_MOD_BACK = 47
SBP_48_MOD_MISC1 = 48
SBP_49_MOD_PELVIS_PRIMARY = 49
SBP_50_DECAPITATEDHEAD = 50
SBP_51_DECAPITATE = 51
SBP_52_MOD_PELVIS_SECONDARY = 52
SBP_53_MOD_LEG_RIGHT = 53
SBP_54_MOD_LEG_LEFT = 54
SBP_55_MOD_FACE_JEWELRY = 55
SBP_56_MOD_CHEST_SECONDARY = 56
SBP_57_MOD_SHOULDER = 57
SBP_58_MOD_ARM_LEFT = 58
```

SBP_59_MOD_ARM_RIGHT = 59

SBP_60_MOD_MISC2 = 60

SBP_61_FX01 = 61

class BSDismemberSkinInstance(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._BSDismemberSkinInstance, object

get_dismember_partitions()
Return triangles and body part indices.

class BSDistantTreeShaderProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.BSShaderProperty

Bethesda-specific node.

class BSEffectShaderProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiProperty

Skyrim non-PP shader model, used primarily for transparency effects, often as decal.

property emissive_color

property emissive_multiple

property falloff_start_angle

property falloff_start_opacity

property falloff_stop_angle

property falloff_stop_opacity

property greyscale_texture

property shader_flags_1

property shader_flags_2

property soft_falloff_depth

property source_texture

property texture_clamp_mode

property uv_offset

property uv_scale

class BSEffectShaderPropertyColorController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiFloatInterpController

This controller is used to animate colors in BSEffectShaderProperty.

property type_of_controlled_color

class BSEffectShaderPropertyFloatController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiFloatInterpController

This controller is used to animate float variables in BSEffectShaderProperty.

property type_of_controlled_variable

class BSFadeNode(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

Bethesda-specific fade node.

class BSFrustumFOVController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

Bethesda-specific node.

property interpolator

class BSFurnitureMarker(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Unknown. Marks furniture sitting positions?

property num_positions

property positions

class BSFurnitureMarkerNode(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.BSFurnitureMarker

Furniture Marker for actors

class BSInvMarker(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Orientation marker for Skyrim's inventory view. How to show the nif in the player's inventory. Typically attached to the root node of the nif tree. If not present, then Skyrim will still show the nif in inventory, using the default values. Name should be 'INV' (without the quotes). For rotations, a short of "4712" appears as "4.712" but "959" appears as "0.959" meshesweaponsdaedricdaedricbowskinned.nif

property rotation_x

property rotation_y

property rotation_z

property zoom

class BSKeyframeController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiKeyframeController

An extended keyframe controller.

property data_2

class BSLODTriShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTriBasedGeom

A variation on NiTriShape, for visibility control over vertex groups.

property level_0_size

property level_1_size

property level_2_size

class BSLagBoneController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

A controller that trails a bone behind an actor.

property linear_rotation

property linear_velocity

property maximum_distance

class BSLeafAnimNode(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

Unknown, related to trees.

class BSLightingShaderProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiProperty

Skyrim PP shader for assigning material/shader/texture.

property alpha

property emissive_color

property emissive_multiple

property environment_map_scale

property eye_cubemap_scale

property glossiness

property hair_tint_color

property left_eye_reflection_center

property lighting_effect_1

property lighting_effect_2

property max_passes

property parallax_envmap_strength

property parallax_inner_layer_texture_scale

property parallax_inner_layer_thickness

property parallax_refraction_scale

property refraction_strength

property right_eye_reflection_center

property scale

property shader_flags_1

property shader_flags_2

property skin_tint_color

property sparkle_parameters

property specular_color

property specular_strength

property texture_clamp_mode

property texture_set

property uv_offset

property uv_scale

class BSLightingShaderPropertyColorController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiFloatInterpController

This controller is used to animate colors in BSLightingShaderProperty.

property type_of_controlled_color

class BSLightingShaderPropertyFloatController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiFloatInterpController

This controller is used to animate float variables in BSLightingShaderProperty.

property type_of_controlled_variable

class BSLightingShaderPropertyShaderType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

Values for configuring the shader type in a BSLightingShaderProperty

Default = 0

Heightmap = 3

WorldMap1 = 9

WorldMap2 = 13

WorldMap3 = 15

WorldMap4 = 18

class BSMasterParticleSystem(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

Bethesda-Specific node.

property max_emitter_objects

property num_particle_systems

property particle_systems

class BSMaterialEmittanceMultController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiFloatInterpController

Bethesda-Specific node.

class BSMultiBound(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Bethesda-specific node.

property data

class BSMultiBoundAABB(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.BSMultiBoundData

Bethesda-specific node.

property extent

property position

class BSMultiBoundData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Abstract base type for bounding data.

class BSMultiBoundNode(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

Bethesda-specific node.

property multi_bound

property unknown_int

class BSMultiBoundOBB(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.BSMultiBoundData

Oriented bounding box.

property center

property rotation

property size

class BSMultiBoundSphere(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.BSMultiBoundData

Bethesda-specific node.

property radius

property unknown_int_1

property unknown_int_2

property unknown_int_3

class BSNiAlphaPropertyTestRefController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiAlphaController

Unkown

class BSOrderedNode(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

Bethesda-Specific node.

property alpha_sort_bound

property is_static_bound

class BSPSysArrayEmitter(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysVolumeEmitter

Particle emitter that uses a node, its children and subchildren to emit from. Emission will be evenly spread along points from nodes leading to their direct parents/children only.

class BSPSysHavokUpdateModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

property modifier

property nodes

property num_nodes

class BSPSysInheritVelocityModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

property unknown_float_1

property unknown_float_2

property unknown_float_3

property unknown_int_1

class BSPSysLODModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

property uknown_float_1

property uknown_float_2

property uknown_float_3

property uknown_float_4

class BSPSysMultiTargetEmitterCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierCtlr

Particle system (multi?) emitter controller.

property data

property unknown_int_1

property unknown_short_1

property visibility_interpolator

class BSPSysRecycleBoundModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

property unknown_float_1

property unknown_float_2

property unknown_float_3

property unknown_float_4

property unknown_float_5

property unknown_float_6

property unknown_int_1

class BSPSysScaleModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

property floats

property num_floats

class BSPSysSimpleColorModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

Bethesda-Specific Particle node.

property color_1_end_percent

property color_1_start_percent

property color_2_end_percent

property color_2_start_percent

property colors

property fade_in_percent

property fade_out_percent

class BSPSysStripUpdateModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

Bethesda-Specific (mesh?) Particle System Modifier.

property update_delta_time

class BSPSysSubTexModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

Similar to a Flip Controller, this handles particle texture animation on a single texture atlas

property end_frame

property frame_count

property frame_count_fudge

property loop_start_frame

property loop_start_frame_fudge

property start_frame

property start_frame_fudge

class BSPackedAdditionalDataBlock(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

property atom_sizes

property block_offsets

property data

property has_data

property num_atoms

property num_blocks

property num_total_bytes

property num_total_bytes_per_element

property unknown_int_1

class BSPackedAdditionalGeometryData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.AbstractAdditionalGeometryData

property block_infos

property blocks

property num_block_infos

property num_blocks

property num_vertices

class BSParentVelocityModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

Particle modifier that adds a blend of object space translation and rotation to particles born in world space.

property damping

class BSPartFlag(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.bit_struct.BitStructBase

Editor flags for the Body Partitions.

property pf_editor_visible

property pf_start_net_boneset

property reserved_bits_1

class BSProceduralLightningController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiFloatInterpController

Skyrim, Paired with dummy TriShapes, this controller generates lightning shapes for special effects. First interpolator controls Generation.

property byte_1

property byte_2

property byte_3

property distance_weight

property float_2

property float_5

property fork

property interpolator_10

property interpolator_2_mutation

property interpolator_3

property interpolator_4

property interpolator_5

property interpolator_6

property interpolator_7

property interpolator_8

property interpolator_9_arc_offset

property strip_width

property unknown_short_1

property unknown_short_2

property unknown_short_3

class BSRefractionFirePeriodController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

Bethesda-specific node.

property interpolator

class BSRefractionStrengthController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiFloatInterpController

Bethesda-Specific node.

class BSRotAccumTransfInterpolator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTransformInterpolator

class BSSegment(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Bethesda-specific node.

property flags

property internal_index

property unknown_byte_1

class BSSegmentFlags(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.bit_struct.BitStructBase

An unsigned 32-bit integer, describing what's inside the segment.

property bsseg_water

property reserved_bits_0

class BSSegmentedTriShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTriShape

Bethesda-specific node.

property num_segments

property segment

class BSShaderFlags(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.bit_struct.BitStructBase

Shader Property Flags

property sf_alpha_texture

property sf_decal_single_pass

property sf_dynamic_alpha

property sf_dynamic_decal_single_pass

property sf_empty

property sf_environment_mapping

property sf_external_emittance

property sf_eye_environment_mapping

property sf_face_gen

property sf_fire_refraction

property sf_hair

property sf_localmap_hide_secret

property sf_low_detail

property sf_multiple_textures

property sf_non_projective_shadows

property sf_parallax_occulsion

property sf_parallax_shader_index_15

property sf_refraction

property sf_remappable_textures

property sf_shadow_frustum

property sf_shadow_map

property sf_single_pass

property sf_skinned

property sf_specular

property sf_tree_billboard

property sf_unknown_1

property sf_unknown_2

property sf_unknown_3

property sf_unknown_4

property sf_vertex_alpha

property sf_window_environment_mapping

property sf_z_buffer_test

class BSShaderFlags2 (*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.object_models.xml.bit_struct.BitStructBase`

Shader Property Flags 2

property sf_2_1_st_light_is_point_light

property sf_2_2_nd_light

property sf_2_3_rd_light

property sf_2_alpha_decal

property sf_2_billboard_and_envmap_light_fade

property sf_2_envmap_light_fade

property sf_2_fit_slope

property sf_2_lod_building

property sf_2_lod_landscape

property sf_2_no_fade

property sf_2_no_lod_land_blend

property sf_2_no_transparecny_multisampling

property sf_2_premult_alpha

property sf_2_refraction_tint

property sf_2_show_in_local_map

property sf_2_skip_normal_maps

property sf_2_uniform_scale

property sf_2_unknown_1

property sf_2_unknown_10

property sf_2_unknown_2

property sf_2_unknown_3

property sf_2_unknown_4

property sf_2_unknown_5

property sf_2_unknown_6

property sf_2_unknown_7

property sf_2_unknown_8

property sf_2_unknown_9

property sf_2_vats_selection

property sf_2_vertex_colors

property sf_2_vertex_lighting

property sf_2_wireframe

property sf_2_z_buffer_write

class BSShaderLightingProperty(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif.BSShaderProperty`

Bethesda-specific property.

property texture_clamp_mode

class BSShaderNoLightingProperty(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif.BSShaderLightingProperty`

Bethesda-specific property.

property falloff_start_angle

property falloff_start_opacity

property falloff_stop_angle

property falloff_stop_opacity

property file_name

class BSShaderPPLightingProperty(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif.BSShaderLightingProperty`

Bethesda-specific Shade node.

property emissive_color

property refraction_fire_period

property refraction_strength

property texture_set

property unknown_float_4

property unknown_float_5

class BSShaderProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiProperty

Bethesda-specific Property node

property environment_map_scale

property shader_flags

property shader_flags_2

property shader_type

property smooth

class BSShaderTextureSet(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Bethesda-specific Texture Set.

property num_textures

property textures

class BSShaderType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

The type of animation interpolation (blending) that will be used on the associated key frames.

SHADER_DEFAULT = 1

SHADER_LIGHTING30 = 29

SHADER_NOLIGHTING = 33

SHADER_SKIN = 14

SHADER_SKY = 10

SHADER_TALL_GRASS = 0

SHADER_TILE = 32

SHADER_WATER = 17

class BSSkyShaderProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiProperty

Skyrim Sky shader block.

property shader_flags_1

property shader_flags_2

property sky_object_type

property source_texture

property uv_offset

property uv_scale

class BSStripPSysData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysData

Bethesda-Specific (mesh?) Particle System Data.

property unknown_byte_6

property unknown_float_8

property unknown_int_7

property unknown_short_5

class BSStripParticleSystem(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiParticleSystem

Bethesda-Specific (mesh?) Particle System.

class BSTreadTransfInterpolator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiInterpolator

Bethesda-specific node.

property data

property num_tread_transforms

property tread_transforms

class BSTreadTransform(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Bethesda-specific node.

property name

property transform_1

property transform_2

class BSTreadTransformData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Bethesda-specific node.

property rotation

property scale

property translation

class BSTreeNode(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

Node for handling Trees, Switches branch configurations for variation?

property bones

property bones_1

property num_bones_1

property num_bones_2

class BSValueNode(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

Bethesda-Specific node. Found on fxFire effects

property unknown_byte

property value

class BSWArray(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Bethesda-specific node.

property items

property num_items

class BSWaterShaderProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiProperty

Skyrim water shader property, different from "WaterShaderProperty" seen in Fallout.

property shader_flags_1

property shader_flags_2

property unknown_short_3

property uv_offset

property uv_scale

property water_direction

property water_shader_flags

class BSWindModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

Particle Modifier that uses the wind value from the gamedata to alter the path of particles.

property strength

class BSXFlags(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiIntegerExtraData

Controls animation and collision. Integer holds flags: Bit 0 : enable havok, bAnimated(Skyrim) Bit 1 : enable collision, bHavok(Skyrim) Bit 2 : is skeleton nif?, bRagdoll(Skyrim) Bit 3 : enable animation, bComplex(Skyrim) Bit 4 : FlameNodes present, bAddon(Skyrim) Bit 5 : EditorMarkers present Bit 6 : bDynamic(Skyrim) Bit 7 : bArticulated(Skyrim) Bit 8 : bIKTarget(Skyrim) Bit 9 : Unknown(Skyrim)

class BallAndSocketDescriptor(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

property unknown_4_bytes

property unknown_floats_1

property unknown_floats_2

property unknown_int_1

class BillboardMode(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

Determines the way the billboard will react to the camera. Billboard mode is stored in lowest 3 bits although Oblivion vanilla nifs uses values higher than 7.

ALWAYS_FACE_CAMERA = 0

ALWAYS_FACE_CENTER = 3

BSROTATE_ABOUT_UP = 5

RIGID_FACE_CAMERA = 2

RIGID_FACE_CENTER = 4

ROTATE_ABOUT_UP = 1

ROTATE_ABOUT_UP2 = 9

BlockTypeIndex
alias of pyffi.object_models.common.UShort

class BodyPartList(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Body part list for DismemberSkinInstance

property body_part

property part_flag

class BoneLOD(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Stores Bone Level of Detail info in a BSBoneLODExtraData

property bone_name

property distance

class BoundVolumeType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

BASE_BV = 4294967295

BOX_BV = 1

CAPSULE_BV = 2

HALFSPACE_BV = 5

SPHERE_BV = 0

UNION_BV = 4

class BoundingBox(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Bounding box.

property radius

property rotation

property translation

property unknown_int

class BoundingVolume(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

property box

property capsule

property collision_type

property half_space

property sphere

property union

class BoxBV(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Box Bounding Volume

property axis

property center

property extent

class ByteArray(***kwargs*)
Bases: pyffi.object_models.xml.basic.BasicBase

Array (list) of bytes. Implemented as basic type to speed up reading and also to prevent data to be dumped by __str__.

get_hash(*data=None*)
Returns a hash value (an immutable object) that can be used to identify the object uniquely.

get_size(*data=None*)
Returns size of the object in bytes.

get_value()
Return object value.

read(*stream*, *data*)
Read object from file.

set_value(*value*)
Set object value.

write(*stream*, *data*)
Write object to file.

class ByteColor3(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A color without alpha (red, green, blue).

property b

property g

property r

class ByteColor4(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A color with alpha (red, green, blue, alpha).

property a

property b

property g

property r

class ByteMatrix(***kwargs*)
Bases: pyffi.object_models.xml.basic.BasicBase

Matrix of bytes. Implemented as basic type to speed up reading and to prevent data being dumped by __str__.

get_hash(*data=None*)
Returns a hash value (an immutable object) that can be used to identify the object uniquely.

get_size(*data=None*)
Returns size of the object in bytes.

get_value()
Return object value.

read(*stream*, *data*)
Read object from file.

set_value(*value*)
Set object value.

write(*stream*, *data*)
Write object to file.

class CStreamableAssetData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

property root

property unknown_bytes

class CapsuleBV(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Capsule Bounding Volume

property center

property origin

property unknown_float_1

property unknown_float_2

class ChannelConvention(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

CC_COMPRESSED = 4

CC_EMPTY = 5

CC_FIXED = 0

CC_INDEX = 3

class ChannelData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Channel data

property bits_per_channel

property convention

property type

property unknown_byte_1

class ChannelType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

CHNL_ALPHA = 3

CHNL_BLUE = 2

CHNL_COMPRESSED = 4

CHNL_EMPTY = 19

CHNL_GREEN = 1

CHNL_INDEX = 16

CHNL_RED = 0

class CloningBehavior(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

Sets how objects are to be cloned.

CLONING_BLANK_COPY = 2

CLONING_COPY = 1

CLONING_SHARE = 0

class CollisionMode(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

CM_NOTEST = 3

CM_USE_ABV = 2

CM_USE_NIBOUND = 4

CM_USE_OBB = 0

CM_USE_TRI = 1

class Color3(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A color without alpha (red, green, blue).

property b

property g

property r

class Color4(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A color with alpha (red, green, blue, alpha).

property a

property b

property g

property r

class ComponentFormat(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

The data format of components.

F_FLOAT16_1 = 66097

F_FLOAT16_2 = 131634

F_FLOAT16_3 = 197171

F_FLOAT16_4 = 262708

F_FLOAT32_1 = 66613

F_FLOAT32_2 = 132150

F_FLOAT32_3 = 197687

```
F_FLOAT32_4 = 263224
F_INT16_1 = 66065
F_INT16_2 = 131602
F_INT16_3 = 197139
F_INT16_4 = 262676
F_INT32_1 = 66593
F_INT32_2 = 132130
F_INT32_3 = 197667
F_INT32_4 = 263204
F_INT8_1 = 65793
F_INT8_2 = 131330
F_INT8_3 = 196867
F_INT8_4 = 262404
F_NORMINT16_1 = 66073
F_NORMINT16_2 = 131610
F_NORMINT16_3 = 197147
F_NORMINT16_4 = 262684
F_NORMINT32_1 = 66601
F_NORMINT32_2 = 132138
F_NORMINT32_3 = 197675
F_NORMINT32_4 = 263212
F_NORMINT8_1 = 65801
F_NORMINT8_2 = 131338
F_NORMINT8_3 = 196875
F_NORMINT8_4 = 262412
F_NORMINT_10_10_10_2 = 66621
F_NORMINT_10_10_10_L1 = 66618
F_NORMINT_11_11_10 = 66619
F_NORMUINT16_1 = 66077
F_NORMUINT16_2 = 131614
F_NORMUINT16_3 = 197151
F_NORMUINT16_4 = 262688
F_NORMUINT32_1 = 66605
F_NORMUINT32_2 = 132142
F_NORMUINT32_3 = 197679
F_NORMUINT32_4 = 263216
```

F_NORMUINT8_1 = 65805

F_NORMUINT8_2 = 131342

F_NORMUINT8_3 = 196879

F_NORMUINT8_4 = 262416

F_NORMUINT8_4_BGRA = 262460

F_UINT16_1 = 66069

F_UINT16_2 = 131606

F_UINT16_3 = 197143

F_UINT16_4 = 262680

F_UINT32_1 = 66597

F_UINT32_2 = 132134

F_UINT32_3 = 197671

F_UINT32_4 = 263208

F_UINT8_1 = 65797

F_UINT8_2 = 131334

F_UINT8_3 = 196871

F_UINT8_4 = 262408

F_UINT_10_10_10_2 = 66622

F_UINT_10_10_10_L1 = 66617

F_UNKNOWN = 0

class ConsistencyType(***kwargs*)

Bases: `pyffi.object_models.xml.enum.EnumBase`

Used by NiGeometryData to control the volatility of the mesh. While they appear to be flags they behave as an enum.

CT_MUTABLE = 0

CT_STATIC = 16384

CT_VOLATILE = 32768

class ControllerLink(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif._ControllerLink`, `object`

```
>>> from pyffi.formats.nif import NifFormat
>>> link = NifFormat.ControllerLink()
>>> link.node_name_offset
-1
>>> link.set_node_name("Bip01")
>>> link.node_name_offset
0
>>> link.get_node_name()
b'Bip01'
>>> link.node_name
b'Bip01'
>>> link.set_node_name("Bip01 Tail")
```

(continues on next page)

(continued from previous page)

```
>>> link.node_name_offset
6
>>> link.get_node_name()
b'Bip01 Tail'
>>> link.node_name
b'Bip01 Tail'
```

get_controller_type()

get_node_name()

Return the node name.

```
>>> # a doctest
>>> from pyffi.formats.nif import NifFormat
>>> link = NifFormat.ControllerLink()
>>> link.string_palette = NifFormat.NiStringPalette()
>>> palette = link.string_palette.palette
>>> link.node_name_offset = palette.add_string("Bip01")
>>> link.get_node_name()
b'Bip01'
```

```
>>> # another doctest
>>> from pyffi.formats.nif import NifFormat
>>> link = NifFormat.ControllerLink()
>>> link.node_name = "Bip01"
>>> link.get_node_name()
b'Bip01'
```

get_property_type()

get_variable_1()

get_variable_2()

set_controller_type(*text*)

set_node_name(*text*)

set_property_type(*text*)

set_variable_1(*text*)

set_variable_2(*text*)

class CoordGenType(***kwargs*)

Bases: pyffi.object_models.xml.enum.EnumBase

Determines the way that UV texture coordinates are generated.

CG_DIFFUSE_CUBE_MAP = 4

CG_SPECULAR_CUBE_MAP = 3

CG_SPHERE_MAP = 2

CG_WORLD_PARALLEL = 0

CG_WORLD_PERSPECTIVE = 1

class CycleType(***kwargs*)

Bases: pyffi.object_models.xml.enum.EnumBase

The animation cyle behavior.

CYCLE_CLAMP = 2

CYCLE_LOOP = 0

CYCLE_REVERSE = 1

class Data(*version=67108866*, *user_version=0*, *user_version_2=0*)
Bases: pyffi.object_models.Data

A class to contain the actual nif data.

Note that L{header} and L{blocks} are not automatically kept in sync with the rest of the nif data, but they are resynchronized when calling L{write}.

Variables

- ***version*** – The nif version.
- ***user_version*** – The nif user version.
- ***user_version_2*** – The nif user version 2.
- **roots** – List of root blocks.
- **header** – The nif header.
- ***blocks*** – List of blocks.
- **modification** – Neo Steam ("neosteam") or Ndoors ("ndoors") or Joymaster Interactive Howling Sword ("jmihs1") or Laxe Lore ("laxelore") style nif?

class VersionUInt(***kwargs*)
Bases: pyffi.object_models.common.UInt

get_detail_display()
Return an object that can be used to display the instance.

set_value(*value*)
Set value to C{value}. Calls C{int(value)} to convert to integer.
Parameters **value** (*int*) – The value to assign.

get_detail_child_names(*edge_filter=(True, True)*)
Generator which yields all child names of this item in the detail view.

Override this method if the node has children.
Returns Generator for detail tree child names.
Return type generator yielding strs

get_detail_child_nodes(*edge_filter=(True, True)*)
Generator which yields all children of this item in the detail view (by default, all acyclic and active ones).

Override this method if the node has children.
Parameters **edge_filter** (EdgeFilter or type(None)) – The edge type to include.
Returns Generator for detail tree child nodes.
Return type generator yielding DetailNodes

get_global_child_nodes(*edge_filter=(True, True)*)
Generator which yields all children of this item in the global view, of given edge type (default is edges of type 0).

Override this method.
Returns Generator for global node children.

inspect(*stream*)
Quickly checks whether the stream appears to contain nif data, and read the nif header. Resets stream to original position.

Call this function if you only need to inspect the header of the nif.
Parameters **stream** (`file`) – The file to inspect.

inspect_version_only(*stream*)
This function checks the version only, and is faster than the usual inspect function (which reads the full header). Sets the L{version} and L{user_version} instance variables if the stream contains a valid NIF file.

Call this function if you simply wish to check that a file is a NIF file without having to parse even the header.
Raises **ValueError** – If the stream does not contain a NIF file.
Parameters **stream** (`file`) – The stream from which to read.

read(*stream*)
Read a NIF file. Does not reset stream position.
Parameters **stream** (`file`) – The stream from which to read.

replace_global_node(*oldbranch*, *newbranch*, *edge_filter=(True*, *True)*)
Replace a particular branch in the graph.

property user_version

property user_version_2

property version

write(*stream*)
Write a NIF file. The L{header} and the L{blocks} are recalculated from the tree at L{roots} (e.g. list of block types, number of blocks, list of block types, list of strings, list of block sizes etc.).
Parameters **stream** (*file*) – The stream to which to write.

class DataStreamAccess(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.object_models.xml.bit_struct.BitStructBase`

Determines how the data stream is accessed?

property cpu_read

property cpu_write_mutable

property cpu_write_static

property cpu_write_static_inititialized

property cpu_write_volatile

property gpu_read

property gpu_write

class DataStreamUsage(***kwargs*)
Bases: `pyffi.object_models.xml.enum.EnumBase`

Determines how a data stream is used?

USAGE_SHADER_CONSTANT = 2

USAGE_USER = 3

USAGE_VERTEX = 1

USAGE_VERTEX_INDEX = 0

class DeactivatorType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

DEACTIVATOR_INVALID = 0

DEACTIVATOR_NEVER = 1

DEACTIVATOR_SPATIAL = 2

class DecalVectorArray(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Array of Vectors for Decal placement in BSDecalPlacementVectorExtraData.

property normals

property num_vectors

property points

class DecayType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

Determines decay function. Used by NiPSysBombModifier.

DECAY_EXPONENTIAL = 2

DECAY_LINEAR = 1

DECAY_NONE = 0

class DistantLODShaderProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.BSShaderProperty

Bethesda-specific node.

EPSILON = 0.0001

class EffectShaderControlledColor(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

An unsigned 32-bit integer, describing which color in BSEffectShaderProperty to animate.

class EffectShaderControlledVariable(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

An unsigned 32-bit integer, describing which float variable in BSEffectShaderProperty to animate.

EmissiveMultiple = 0

class EffectType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

The type of information that's store in a texture used by a NiTextureEffect.

EFFECT_ENVIRONMENT_MAP = 2

EFFECT_FOG_MAP = 3

EFFECT_PROJECTED_LIGHT = 0

EFFECT_PROJECTED_SHADOW = 1

class ElementReference(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

property normalize_flag

property semantic

class EmitFrom(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

Controls which parts of the mesh that the particles are emitted from.

EMIT_FROM_EDGE_CENTER = 2

EMIT_FROM_EDGE_SURFACE = 4

EMIT_FROM_FACE_CENTER = 1

EMIT_FROM_FACE_SURFACE = 3

EMIT_FROM_VERTICES = 0

class EndianType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

ENDIAN_BIG = 0

ENDIAN_LITTLE = 1

class ExportInfo(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Information about how the file was exported

property creator

property export_info_1

property export_info_2

property unknown

class ExtraMeshDataEpicMickey(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

property unknown_int_1

property unknown_int_2

property unknown_int_3

property unknown_int_4

property unknown_int_5

property unknown_int_6

class ExtraMeshDataEpicMickey2(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

property end

property start

property unknown_shorts

class ExtraVectorsFlags(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

None = 0

Tangents_Bitangents = 16

class FaceDrawMode(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

This enum lists the different face culling options.

DRAW_BOTH = 3

DRAW_CCW = 1

DRAW_CCW_OR_BOTH = 0

DRAW_CW = 2

class Fallout3HavokMaterial(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

A material, used by havok shape objects in Fallout 3. Bit 5: flag for PLATFORM (for values 32-63 substract 32 to know material number) Bit 6: flag for STAIRS (for values 64-95 substract 64 to know material number) Bit 5+6: flag for STAIRS+PLATFORM (for values 96-127 substract 96 to know material number)

MAT_BABY_RATTLE = 30

MAT_BARREL = 23

MAT_BOTTLE = 24

MAT_BOTTLECAP = 14

MAT_BROKEN_CONCRETE = 19

MAT_CHAIN = 13

MAT_CLOTH = 1

MAT_DIRT = 2

MAT_ELEVATOR = 15

MAT_GLASS = 3

MAT_GRASS = 4

MAT_HEAVY_METAL = 11

MAT_HEAVY_STONE = 10

MAT_HEAVY_WOOD = 12

MAT_HOLLOW_METAL = 16

MAT_LUNCHBOX = 29

MAT_METAL = 5

MAT_ORGANIC = 6

MAT_PISTOL = 26

MAT_RIFLE = 27

MAT_RUBBER_BALL = 31

MAT_SAND = 18

MAT_SHEET_METAL = 17

MAT_SHOPPING_CART = 28

MAT_SKIN = 7

MAT_SODA_CAN = 25

MAT_STONE = 0

MAT_VEHICLE_BODY = 20

MAT_VEHICLE_PART_HOLLOW = 22

MAT_VEHICLE_PART_SOLID = 21

MAT_WATER = 8

MAT_WOOD = 9

class Fallout3Layer(***kwargs*)

Bases: `pyffi.object_models.xml.enum.EnumBase`

Sets mesh color in Fallout 3 GECK. Anything higher than 72 is also null.

ACOUSTIC_SPACE = 21

ACTORZONE = 22

ADDONARM = 71

ADDONCHEST = 70

ADDONHEAD = 69

ADDONLEG = 72

ANIM_STATIC = 2

AVOIDBOX = 31

BIPED = 29

BODY = 46

CAMERAPICK = 35

CAMERASPHERE = 33

CHAIN = 68

CHARCONTROLLER = 30

CLOUD_TRAP = 16

CLUTTER = 4

COLLISIONBOX = 32

CUSTOMPICK1 = 39

CUSTOMPICK2 = 40

DEBRIS_LARGE = 20

DEBRIS_SMALL = 19

DOORDETECTION = 34

DROPPINGPICK = 42

GASTRAP = 24

GROUND = 17

HEAD = 45

```
INVISIBLE_WALL = 27
ITEMPICK = 36
LINEOFSIGHT = 37
L_CALF = 53
L_FOOT = 54
L_FORE_ARM = 50
L_HAND = 51
L_THIGH = 52
L_UPPER_ARM = 49
NONCOLLIDABLE = 15
NULL = 43
OTHER = 44
PACK = 67
PATHPICK = 38
PONYTAIL = 65
PORTAL = 18
PROJECTILE = 6
PROJECTILEZONE = 23
PROPS = 10
QUIVER = 63
R_CALF = 59
R_FOOT = 60
R_FORE_ARM = 56
R_HAND = 57
R_THIGH = 58
R_UPPER_ARM = 55
SHELLCASING = 25
SHIELD = 62
SPELL = 7
SPELLEXPLOSION = 41
SPINE1 = 47
SPINE2 = 48
STATIC = 1
TAIL = 61
TERRAIN = 13
TRANSPARENT = 3
```

TRANSPARENT_SMALL = 26

TRANSPARENT_SMALL_ANIM = 28

TRAP = 14

TREES = 9

TRIGGER = 12

UNIDENTIFIED = 0

WATER = 11

WEAPON = 64

WING = 66

class FieldType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

The force field's type.

FIELD_POINT = 1

FIELD_WIND = 0

class FilePath(***kwargs*)
Bases: pyffi.formats.nif.string

A file path.

get_hash(*data=None*)
Returns a case insensitive hash value.

class FileVersion(***kwargs*)
Bases: pyffi.object_models.common.UInt

get_detail_display()
Return an object that can be used to display the instance.

read(*stream*, *data*)
Read value from stream.
Parameters **stream** (*file*) – The stream to read from.

set_value()
Set value to C{value}. Calls C{int(value)} to convert to integer.
Parameters **value** (*int*) – The value to assign.

write(*stream*, *data*)
Write value to stream.
Parameters **stream** (*file*) – The stream to write to.

class Flags(***kwargs*)
Bases: pyffi.object_models.common.UShort

class Footer(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._Footer, object

read(*stream*, *data*)
Read structure from stream.

write(*stream*, *data*)
Write structure to stream.

class ForceType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

The type of force? May be more valid values.

FORCE_PLANAR = 0

FORCE_SPHERICAL = 1

FORCE_UNKNOWN = 2

class FurnitureEntryPoints(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.bit_struct.BitStructBase

Furniture entry points. It specifies the direction(s) from where the actor is able to enter (and leave) the position.

property behind

property front

property left

property right

property up

class FurniturePosition(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Describes a furniture position?

property animation_type

property entry_properties

property heading

property offset

property orientation

property position_ref_1

property position_ref_2

class FxButton(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.FxWidget

Unknown.

class FxRadioButton(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.FxWidget

Unknown.

property buttons

property num_buttons

property unknown_int_1

property unknown_int_2

property unknown_int_3

class FxWidget(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

Firaxis-specific UI widgets?

property unknown_292_bytes

property unknown_3

class HairShaderProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.BSShaderProperty

Bethesda-specific node.

class HalfSpaceBV(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

property center

property normal

property unknown_float_1

class HavokColFilter(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

ColFilter property for Havok. It contains Layer, Flags and Part Number

property flags_and_part_number

property layer

property unknown_short

class HavokMaterial(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

property material

class Header(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._Header, object

has_block_type(*block_type*)
Check if header has a particular block type.
Raises ValueError – If number of block types is zero (only nif versions 10.0.1.0 and up store block types in header).
Parameters **block_type** (*L{NifFormat.NiObject}*) – The block type.
Returns True if the header's list of block types has the given block type, or a subclass of it. False otherwise.
Return type bool

class HeaderString(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.basic.BasicBase

get_detail_display()
Return an object that can be used to display the instance.

get_hash(*data=None*)
Returns a hash value (an immutable object) that can be used to identify the object uniquely.

get_size(*data=None*)
Returns size of the object in bytes.

read(*stream*, *data*)
Read object from file.

static version_string(*version*, *modification=None*)
Transforms version number into a version string.

```
>>> NifFormat.HeaderString.version_string(0x03000300)
'NetImmerse File Format, Version 3.03'
>>> NifFormat.HeaderString.version_string(0x03010000)
'NetImmerse File Format, Version 3.1'
>>> NifFormat.HeaderString.version_string(0x0A000100)
'NetImmerse File Format, Version 10.0.1.0'
>>> NifFormat.HeaderString.version_string(0x0A010000)
'Gamebryo File Format, Version 10.1.0.0'
>>> NifFormat.HeaderString.version_string(0x0A010000,
...                                       modification="neosteam")
'NS'
>>> NifFormat.HeaderString.version_string(0x14020008,
...                                       modification="ndoors")
'NDSNIF....@....@...., Version 20.2.0.8'
>>> NifFormat.HeaderString.version_string(0x14030009,
...                                       modification="jmihs1")
'Joymaster HS1 Object Format - (JMI), Version 20.3.0.9'
```

write(*stream*, *data*)
Write object to file.

class HingeDescriptor(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

This constraint allows rotation about a specified axis.

property axle_a

property axle_b

property perp_2_axle_in_a_1

property perp_2_axle_in_a_2

property perp_2_axle_in_b_1

property perp_2_axle_in_b_2

property pivot_a

property pivot_b

class ImageType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

Determines how the raw image data is stored in NiRawImageData.

RGB = 1

RGBA = 2

class InertiaMatrix(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._InertiaMatrix, object

as_list()
Return matrix as 3x3 list.

as_tuple()
Return matrix as 3x3 tuple.

get_copy()
Return a copy of the matrix.

is_identity()
Return `True` if the matrix is close to identity.

set_identity()
Set to identity matrix.

class Key(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.object_models.xml.struct_.StructBase`

A generic key with support for interpolation. Type 1 is normal linear interpolation, type 2 has forward and backward tangents, and type 3 has tension, bias and continuity arguments. Note that color4 and byte always seem to be of type 1.

property backward

property forward

property tbc

property time

property value

class KeyGroup(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.object_models.xml.struct_.StructBase`

Array of vector keys (anything that can be interpolated, except rotations).

property interpolation

property keys

property num_keys

class KeyType(***kwargs*)
Bases: `pyffi.object_models.xml.enum.EnumBase`

The type of animation interpolation (blending) that will be used on the associated key frames.

CONST_KEY = 5

LINEAR_KEY = 1

QUADRATIC_KEY = 2

TBC_KEY = 3

XYZ_ROTATION_KEY = 4

class LODRange(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.object_models.xml.struct_.StructBase`

The distance range where a specific level of detail applies.

property far_extent

property near_extent

property unknown_ints

class LightMode(***kwargs*)
Bases: `pyffi.object_models.xml.enum.EnumBase`

An unsigned 32-bit integer, describing how vertex colors influence lighting.

LIGHT_MODE_EMISSIVE = 0

LIGHT_MODE_EMI_AMB_DIF = 1

class Lighting30ShaderProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.BSShaderPPLightingProperty

Bethesda-specific node.

class LightingShaderControlledColor(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

An unsigned 32-bit integer, describing which color in BSLightingShaderProperty to animate.

class LightingShaderControlledVariable(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

An unsigned 32-bit integer, describing which float variable in BSLightingShaderProperty to animate.

Alpha = 12

Glossiness = 9

class LimitedHingeDescriptor(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._LimitedHingeDescriptor, object

update_a_b(*transform*)
Update B pivot and axes from A using the given transform.

class LineString(***kwargs*)
Bases: pyffi.object_models.xml.basic.BasicBase

Basic type for strings ending in a newline character (0x0a).

```
>>> from tempfile import TemporaryFile
>>> f = TemporaryFile()
>>> l = NifFormat.LineString()
>>> f.write('abcdefg\x0a'.encode())
8
>>> f.seek(0)
0
>>> l.read(f)
>>> str(l)
'abcdefg'
>>> f.seek(0)
0
>>> l.set_value('Hi There')
>>> l.write(f)
>>> f.seek(0)
0
>>> m = NifFormat.LineString()
>>> m.read(f)
>>> str(m)
'Hi There'
```

get_hash(*data=None*)
Returns a hash value (an immutable object) that can be used to identify the object uniquely.

get_size(*data=None*)
Returns size of the object in bytes.

get_value()
Return object value.

read(*stream*, *data=None*)
Read object from file.

set_value(*value*)
Set object value.

write(*stream*, *data=None*)
Write object to file.

class MTransform(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

property rotation

property scale

property translation

class MatchGroup(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Group of vertex indices of vertices that match.

property num_vertices

property vertex_indices

class MaterialData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Data stored per-material by NiRenderObject

property material_extra_data

property material_name

class Matrix22(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A 2x2 matrix of float values. Stored in OpenGL column-major format.

property m_11

property m_12

property m_21

property m_22

class Matrix33(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._Matrix33, object

as_list()
Return matrix as 3x3 list.

as_tuple()
Return matrix as 3x3 tuple.

get_copy()
Return a copy of the matrix.

get_determinant()
Return determinant.

get_inverse()
Get inverse (assuming is_scale_rotation is true!).

get_scale()
Gets the scale (assuming is_scale_rotation is true!).

get_scale_quat()
Decompose matrix into scale and quaternion.

get_scale_rotation()
Decompose the matrix into scale and rotation, where scale is a float and rotation is a C{Matrix33}. Returns a pair (scale, rotation).

get_transpose()
Get transposed of the matrix.

is_identity()
Return `True` if the matrix is close to identity.

is_rotation()
Returns `True` if the matrix is a rotation matrix (a member of SO(3)).

is_scale_rotation()
Returns true if the matrix decomposes nicely into scale * rotation.

set_identity()
Set to identity matrix.

set_scale_rotation(*scale*, *rotation*)
Compose the matrix as the product of scale * rotation.

sup_norm()
Calculate supremum norm of matrix (maximum absolute value of all entries).

class Matrix44(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.formats.nif._Matrix44`, `object`

as_list()
Return matrix as 4x4 list.

as_tuple()
Return matrix as 4x4 tuple.

get_copy()
Create a copy of the matrix.

get_inverse(*fast=True*)
Calculates inverse (fast assumes is_scale_rotation_translation is True).

get_matrix_33()
Returns upper left 3x3 part.

get_scale_quat_translation()

get_scale_rotation_translation()

get_translation()
Returns lower left 1x3 part.

is_identity()
Return `True` if the matrix is close to identity.

is_scale_rotation_translation()

set_identity()
Set to identity matrix.

set_matrix_33(*m*)
Sets upper left 3x3 part.

set_rows(*row0*, *row1*, *row2*, *row3*)
Set matrix from rows.

set_scale_rotation_translation(*scale*, *rotation*, *translation*)

set_translation(*translation*)
Returns lower left 1x3 part.

sup_norm()
Calculate supremum norm of matrix (maximum absolute value of all entries).

class MeshData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

property component_semantics

property is_per_instance

property num_components

property num_submeshes

property stream

property submesh_to_region_map

class MeshPrimitiveType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

Describes the type of primitives stored in a mesh object.

MESH_PRIMITIVE_LINESTRIPS = 2

MESH_PRIMITIVE_POINTS = 4

MESH_PRIMITIVE_QUADS = 3

MESH_PRIMITIVE_TRIANGLES = 0

MESH_PRIMITIVE_TRISTRIPS = 1

class MipMap(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Description of a MipMap within a NiPixelData object.

property height

property offset

property width

class MipMapFormat(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

An unsigned 32-bit integer, describing how mipmaps are handled in a texture.

MIP_FMT_DEFAULT = 2

MIP_FMT_NO = 0

MIP_FMT_YES = 1

class MoppDataBuildType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

A byte describing if MOPP Data is organized into chunks (PS3) or not (PC)

BUILD_NOT_SET = 2

BUILT_WITHOUT_CHUNK_SUBDIVISION = 1

BUILT_WITH_CHUNK_SUBDIVISION = 0

class Morph(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Geometry morphing data component.

property frame_name

property interpolation

property keys

property num_keys

property unknown_int

property vectors

class MorphWeight(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

property interpolator

property weight

class MotionQuality(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

The motion type. Determines quality of motion?

MO_QUAL_BULLET = 6

MO_QUAL_CHARACTER = 8

MO_QUAL_CRITICAL = 5

MO_QUAL_DEBRIS = 3

MO_QUAL_FIXED = 1

MO_QUAL_INVALID = 0

MO_QUAL_KEYFRAMED = 2

MO_QUAL_KEYFRAMED_REPORT = 9

MO_QUAL_MOVING = 4

MO_QUAL_USER = 7

class MotionSystem(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

The motion system. 4 (Box) is used for everything movable. 7 (Keyframed) is used on statics and animated stuff.

MO_SYS_BOX = 4

MO_SYS_BOX_STABILIZED = 5

MO_SYS_CHARACTER = 9

MO_SYS_DYNAMIC = 1

MO_SYS_FIXED = 7

MO_SYS_INVALID = 0

MO_SYS_KEYFRAMED = 6

MO_SYS_SPHERE = 2

MO_SYS_SPHERE_INERTIA = 3

MO_SYS_THIN_BOX = 8

class MotorDescriptor(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.object_models.xml.struct_.StructBase

property unknown_byte_1

property unknown_float_1

property unknown_float_2

property unknown_float_3

property unknown_float_4

property unknown_float_5

property unknown_float_6

class MultiTextureElement(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.object_models.xml.struct_.StructBase

property clamp

property filter

property has_image

property image

property ps_2_k

property ps_2_l

property unknown_short_3

property uv_set

class Ni3dsAlphaAnimator(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif.NiObject

Unknown.

property num_1

property num_2

property parent

property unknown_1

property unknown_2

class Ni3dsAnimationNode(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif.NiObject

Unknown. Only found in 2.3 nifs.

property child

property count

property has_data

property name

property unknown_array

property unknown_floats_1

property unknown_floats_2

property unknown_short

class Ni3dsColorAnimator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Unknown!

property unknown_1

class Ni3dsMorphShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Unknown!

property unknown_1

class Ni3dsParticleSystem(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Unknown!

property unknown_1

class Ni3dsPathController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Unknown!

property unknown_1

class NiAVObject(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._NiAVObject, object

```
>>> from pyffi.formats.nif import NifFormat
>>> node = NifFormat.NiNode()
>>> prop1 = NifFormat.NiProperty()
>>> prop1.name = "hello"
>>> prop2 = NifFormat.NiProperty()
>>> prop2.name = "world"
>>> node.get_properties()
[]
>>> node.set_properties([prop1, prop2])
>>> [prop.name for prop in node.get_properties()]
[b'hello', b'world']
>>> [prop.name for prop in node.properties]
[b'hello', b'world']
>>> node.set_properties([])
>>> node.get_properties()
[]
>>> # now set them the other way around
>>> node.set_properties([prop2, prop1])
>>> [prop.name for prop in node.get_properties()]
[b'world', b'hello']
>>> [prop.name for prop in node.properties]
[b'world', b'hello']
>>> node.remove_property(prop2)
```

(continues on next page)

(continued from previous page)

```
>>> [prop.name for prop in node.properties]
[b'hello']
>>> node.add_property(prop2)
>>> [prop.name for prop in node.properties]
[b'hello', b'world']
```

add_property(*prop*)
Add the given property to the property list.
Parameters **prop** (*L{NifFormat.NiProperty}*) – The property block to add.

apply_scale(*scale*)
Apply scale factor on data.
Parameters **scale** – The scale factor.

get_properties()
Return a list of the properties of the block.
Returns The list of properties.
Return type list of L{NifFormat.NiProperty}

get_transform(*relative_to=None*)
Return scale, rotation, and translation into a single 4x4 matrix, relative to the C{relative_to} block (which should be another NiAVObject connecting to this block). If C{relative_to} is None, then returns the transform stored in C{self}, or equivalently, the target is assumed to be the parent.
Parameters **relative_to** – The block relative to which the transform must be calculated. If None, the local transform is returned.

remove_property(*prop*)
Remove the given property to the property list.
Parameters **prop** (*L{NifFormat.NiProperty}*) – The property block to remove.

set_properties(*proplist*)
Set the list of properties from the given list (destroys existing list).
Parameters **proplist** (list of L{NifFormat.NiProperty}) – The list of property blocks to set.

set_transform(*m*)
Set rotation, translation, and scale, from a 4x4 matrix.
Parameters **m** – The matrix to which the transform should be set.

class NiAVObjectPalette(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Unknown.

class NiAdditionalGeometryData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.AbstractAdditionalGeometryData

property block_infos

property blocks

property num_block_infos

property num_blocks

property num_vertices

class NiAlphaController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiFloatInterpController

Time controller for transparency.

property data

class NiAlphaProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiProperty

Transparency. Flags 0x00ED.

property flags

property threshold

property unknown_int_2

property unknown_short_1

class NiAmbientLight(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiLight

Ambient light source.

class NiArkAnimationExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Unknown node.

property unknown_bytes

property unknown_ints

class NiArkImporterExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Unknown node.

property importer_name

property unknown_bytes

property unknown_floats

property unknown_int_1

property unknown_int_2

class NiArkShaderExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Unknown node.

property unknown_int

property unknown_string

class NiArkTextureExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Unknown node.

property num_textures

property textures

property unknown_byte

property unknown_int_2

property unknown_ints_1

class NiArkViewportInfoExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Unknown node.

property unknown_bytes

class NiAutoNormalParticles(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiParticles

Unknown.

class NiAutoNormalParticlesData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiParticlesData

Particle system data object (with automatic normals?).

class NiBSAnimationNode(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

Bethesda-specific extension of Node with animation properties stored in the flags, often 42?

class NiBSBoneLODController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiBoneLODController

A simple LOD controller for bones.

class NiBSPArrayController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiParticleSystemController

A particle system controller, used by BS in conjunction with NiBSParticleNode.

class NiBSParticleNode(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

Unknown.

class NiBSplineBasisData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Stores the number of control points of a B-spline.

property num_control_points

class NiBSplineCompFloatInterpolator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiBSplineFloatInterpolator

Unknown.

property base

property bias

property multiplier

property offset

class NiBSplineCompPoint3Interpolator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiBSplinePoint3Interpolator

Unknown.

class NiBSplineCompTransformEvaluator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

class NiBSplineCompTransformInterpolator(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif._NiBSplineCompTransformInterpolator, object

apply_scale(*scale*)

Apply scale factor on data.

get_rotations()

Return an iterator over all rotation keys.

get_scales()

Return an iterator over all scale keys.

get_translations()

Return an iterator over all translation keys.

class NiBSplineData(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif._NiBSplineData, object

```
>>> # a doctest
>>> from pyffi.formats.nif import NifFormat
>>> block = NifFormat.NiBSplineData()
>>> block.num_short_control_points = 50
>>> block.short_control_points.update_size()
>>> for i in range(block.num_short_control_points):
...     block.short_control_points[i] = 20 - i
>>> list(block.get_short_data(12, 4, 3))
[(8, 7, 6), (5, 4, 3), (2, 1, 0), (-1, -2, -3)]
>>> offset = block.append_short_data([(1,2),(4,3),(13,14),(8,2),(33,33)])
>>> offset
50
>>> list(block.get_short_data(offset, 5, 2))
[(1, 2), (4, 3), (13, 14), (8, 2), (33, 33)]
>>> list(block.get_comp_data(offset, 5, 2, 10.0, 32767.0))
[(11.0, 12.0), (14.0, 13.0), (23.0, 24.0), (18.0, 12.0), (43.0, 43.0)]
>>> block.append_float_data([(1.0,2.0),(3.0,4.0),(0.5,0.25)])
0
>>> list(block.get_float_data(0, 3, 2))
[(1.0, 2.0), (3.0, 4.0), (0.5, 0.25)]
>>> block.append_comp_data([(1,2),(4,3)])
(60, 2.5, 1.5)
>>> list(block.get_short_data(60, 2, 2))
[(-32767, -10922), (32767, 10922)]
>>> list(block.get_comp_data(60, 2, 2, 2.5, 1.5))
[(1.0, 2.00...), (4.0, 2.99...)]
```

append_comp_data(*data*)

Append data as compressed list.

Parameters **data** – A list of elements, where each element is a tuple of integers. (Note: cannot be an interator; maybe this restriction will be removed in a future version.)

Returns The offset, bias, and multiplier.

append_float_data(*data*)

Append data.

Parameters **data** – A list of elements, where each element is a tuple of floats. (Note: cannot be an interator; maybe this restriction will be removed in a future version.)

Returns The offset at which the data was appended.

append_short_data(*data*)

Append data.

Parameters **data** – A list of elements, where each element is a tuple of integers. (Note: cannot be an interator; maybe this restriction will be removed in a future version.)
Returns The offset at which the data was appended.

get_comp_data(*offset*, *num_elements*, *element_size*, *bias*, *multiplier*)
Get an interator to the data, converted to float with extra bias and multiplication factor. If C{x} is the short value, then the returned value is C{bias + x * multiplier / 32767.0}.
Parameters
- **offset** – The offset in the data where to start.
- **num_elements** – Number of elements to get.
- **element_size** – Size of a single element.
- **bias** – Value bias.
- **multiplier** – Value multiplier.

Returns A list of C{num_elements} tuples of size C{element_size}.

get_float_data(*offset*, *num_elements*, *element_size*)
Get an iterator to the data.
Parameters
- **offset** – The offset in the data where to start.
- **num_elements** – Number of elements to get.
- **element_size** – Size of a single element.

Returns A list of C{num_elements} tuples of size C{element_size}.

get_short_data(*offset*, *num_elements*, *element_size*)
Get an iterator to the data.
Parameters
- **offset** – The offset in the data where to start.
- **num_elements** – Number of elements to get.
- **element_size** – Size of a single element.

Returns A list of C{num_elements} tuples of size C{element_size}.

class NiBSplineFloatInterpolator(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.formats.nif.NiBSplineInterpolator`

Unknown.

class NiBSplineInterpolator(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.formats.nif._NiBSplineInterpolator`, `object`

get_times()
Return an iterator over all key times.

@todo: When code for calculating the bsplines is ready, this function will return exactly self.basis_data.num_control_points - 1 time points, and not self.basis_data.num_control_points as it is now.

class NiBSplinePoint3Interpolator(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.formats.nif.NiBSplineInterpolator`

Unknown.

property unknown_floats

class NiBSplineTransformInterpolator(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.formats.nif._NiBSplineTransformInterpolator`, `object`

apply_scale(*scale*)
Apply scale factor on data.

get_rotations()
Return an iterator over all rotation keys.

get_scales()
Return an iterator over all scale keys.

get_translations()
Return an iterator over all translation keys.

class NiBezierMesh(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiAVObject

Unknown

property bezier_triangle

property count_1

property count_2

property data_2

property num_bezier_triangles

property points_1

property points_2

property unknown_3

property unknown_4

property unknown_5

property unknown_6

class NiBezierTriangle4(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Sub data of NiBezierMesh

property matrix

property unknown_1

property unknown_2

property unknown_3

property unknown_4

property unknown_5

property unknown_6

property vector_1

property vector_2

class NiBillboardNode(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

These nodes will always be rotated to face the camera creating a billboard effect for any attached objects.

In pre-10.1.0.0 the Flags field is used for BillboardMode. Bit 0: hidden Bits 1-2: collision mode Bit 3: unknown (set in most official meshes) Bits 5-6: billboard mode

Collision modes: 00 NONE 01 USE_TRIANGLES 10 USE_OBBS 11 CONTINUE

Billboard modes: 00 ALWAYS_FACE_CAMERA 01 ROTATE_ABOUT_UP 10 RIGID_FACE_CAMERA 11 ALWAYS_FACE_CENTER

property billboard_mode

class NiBinaryExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Binary extra data object. Used to store tangents and bitangents in Oblivion.

property binary_data

class NiBinaryVoxelData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Voxel data object.

property num_unknown_bytes_2

property num_unknown_vectors

property unknown_5_ints

property unknown_7_floats

property unknown_bytes_1

property unknown_bytes_2

property unknown_short_1

property unknown_short_2

property unknown_short_3

property unknown_vectors

class NiBinaryVoxelExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Voxel extra data object.

property data

property unknown_int

class NiBlendBoolInterpolator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiBlendInterpolator

An interpolator for a bool.

property bool_value

class NiBlendFloatInterpolator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiBlendInterpolator

An interpolator for a float.

property float_value

class NiBlendInterpolator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiInterpolator

An extended type of interpolater.

property unknown_int

property unknown_short

class NiBlendPoint3Interpolator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiBlendInterpolator

Interpolates a point?

property point_value

class NiBlendTransformInterpolator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiBlendInterpolator

Unknown.

class NiBone(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

A NiNode used as a skeleton bone?

class NiBoneLODController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

Level of detail controller for bones. Priority is arranged from low to high.

property node_groups

property num_node_groups

property num_node_groups_2

property num_shape_groups

property num_shape_groups_2

property shape_groups_1

property shape_groups_2

property unknown_int_1

property unknown_int_2

property unknown_int_3

class NiBoolData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Timed boolean data.

property data

class NiBoolInterpController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiSingleInterpController

A controller that interpolates floating point numbers?

class NiBoolInterpolator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiKeyBasedInterpolator

Unknown.

property bool_value

property data

class NiBoolTimelineInterpolator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiBoolInterpolator

Unknown.

class NiBooleanExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Boolean extra data.

property boolean_data

class NiCamera(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiAVObject

Camera object.

property frustum_bottom

property frustum_far

property frustum_left

property frustum_near

property frustum_right

property frustum_top

property lod_adjust

property unknown_int

property unknown_int_2

property unknown_int_3

property unknown_link

property unknown_short

property use_orthographic_projection

property viewport_bottom

property viewport_left

property viewport_right

property viewport_top

class NiClod(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTriBasedGeom

A shape node that holds continuous level of detail information. Seems to be specific to Freedom Force.

class NiClodData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTriBasedGeomData

Holds mesh data for continuous level of detail shapes. Pesumably a progressive mesh with triangles specified by edge splits. Seems to be specific to Freedom Force. The structure of this is uncertain and highly experimental at this point. No file with this data can currently be read properly.

property unknown_clod_shorts_1

property unknown_clod_shorts_2

property unknown_clod_shorts_3

property unknown_count_1

property unknown_count_2

property unknown_count_3

property unknown_float

property unknown_short

property unknown_shorts

class NiClodSkinInstance(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiSkinInstance

A copy of NISkinInstance for use with NiClod meshes.

class NiCollisionData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiCollisionObject

Collision box.

property bounding_volume

property collision_mode

property propagation_mode

property use_abv

class NiCollisionObject(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

This is the most common collision object found in NIF files. It acts as a real object that is visible and possibly (if the body allows for it) interactive. The node itself is simple, it only has three properties. For this type of collision object, bhkRigidBody or bhkRigidBodyT is generally used.

property target

class NiColorData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Color data for material color controller.

property data

class NiColorExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Unknown.

property data

class NiControllerManager(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

Unknown. Root of all controllers?

property controller_sequences

property cumulative

property num_controller_sequences

property object_palette

class NiControllerSequence(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._NiControllerSequence, object

add_controlled_block()
Create new controlled block, and return it.

```
>>> seq = NifFormat.NiControllerSequence()
>>> seq.num_controlled_blocks
0
>>> ctrlblock = seq.add_controlled_block()
>>> seq.num_controlled_blocks
1
>>> isinstance(ctrlblock, NifFormat.ControllerLink)
True
```

class NiDataStream(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif.NiObject

property access

property cloning_behavior

property component_formats

property data

property num_bytes

property num_components

property num_regions

property regions

property streamable

property usage

class NiDefaultAVObjectPalette(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif.NiAVObjectPalette

Unknown. Refers to a list of objects. Used by NiControllerManager.

property num_objs

property objs

property unknown_int

class NiDirectionalLight(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif.NiLight

Directional light source.

class NiDitherProperty(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif.NiProperty

Unknown.

property flags

class NiDynamicEffect(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif.NiAVObject

A dynamic effect such as a light or environment map.

property affected_node_list_pointers

property affected_nodes

property num_affected_node_list_pointers

property num_affected_nodes

property switch_state

class NiEnvMappedTriShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObjectNET

Unknown

property child_2

property child_3

property children

property num_children

property unknown_1

property unknown_matrix

class NiEnvMappedTriShapeData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTriShapeData

Holds mesh data using a list of singular triangles.

class NiExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

A generic extra data object.

property name

property next_extra_data

class NiExtraDataController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiSingleInterpController

An controller for extra data.

class NiFlipController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiFloatInterpController

Texture flipping controller.

property delta

property images

property num_sources

property sources

property texture_slot

property unknown_int_2

class NiFloatData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Possibly the 1D position along a 3D path.

property data

class NiFloatExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Float extra data.

property float_data

class NiFloatExtraDataController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraDataController

Unknown.

property controller_data

property num_extra_bytes

property unknown_bytes

property unknown_extra_bytes

class NiFloatInterpController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiSingleInterpController

A controller that interpolates floating point numbers?

class NiFloatInterpolator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiKeyBasedInterpolator

Unknown.

property data

property float_value

class NiFloatsExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Unknown.

property data

property num_floats

class NiFogProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiProperty

Describes... fog?

property flags

property fog_color

property fog_depth

class NiFurSpringController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

property bones

property bones_2

property num_bones

property num_bones_2

property unknown_float

property unknown_float_2

class NiGeomMorpherController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiInterpController

Time controller for geometry morphing.

property always_update

property data

property extra_flags

property interpolator_weights

property interpolators

property num_interpolators

property num_unknown_ints

property unknown_2

property unknown_ints

class NiGeometry(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif._NiGeometry, object

```
>>> from pyffi.formats.nif import NifFormat
>>> id44 = NifFormat.Matrix44()
>>> id44.set_identity()
>>> skelroot = NifFormat.NiNode()
>>> skelroot.name = 'skelroot'
>>> skelroot.set_transform(id44)
>>> bone1 = NifFormat.NiNode()
>>> bone1.name = 'bone1'
>>> bone1.set_transform(id44)
>>> bone2 = NifFormat.NiNode()
>>> bone2.name = 'bone2'
>>> bone2.set_transform(id44)
>>> bone21 = NifFormat.NiNode()
>>> bone21.name = 'bone21'
>>> bone21.set_transform(id44)
>>> bone22 = NifFormat.NiNode()
>>> bone22.name = 'bone22'
>>> bone22.set_transform(id44)
>>> bone211 = NifFormat.NiNode()
>>> bone211.name = 'bone211'
>>> bone211.set_transform(id44)
>>> skelroot.add_child(bone1)
>>> bone1.add_child(bone2)
>>> bone2.add_child(bone21)
>>> bone2.add_child(bone22)
>>> bone21.add_child(bone211)
>>> geom = NifFormat.NiTriShape()
>>> geom.name = 'geom'
>>> geom.set_transform(id44)
>>> geomdata = NifFormat.NiTriShapeData()
>>> skininst = NifFormat.NiSkinInstance()
>>> skindata = NifFormat.NiSkinData()
>>> skelroot.add_child(geom)
>>> geom.data = geomdata
>>> geom.skin_instance = skininst
>>> skininst.skeleton_root = skelroot
>>> skininst.data = skindata
>>> skininst.num_bones = 4
>>> skininst.bones.update_size()
>>> skininst.bones[0] = bone1
>>> skininst.bones[1] = bone2
>>> skininst.bones[2] = bone22
```

(continues on next page)

(continued from previous page)

```
>>> skininst.bones[3] = bone211
>>> skindata.num_bones = 4
>>> skindata.bone_list.update_size()
>>> [child.name for child in skelroot.children]
[b'bone1', b'geom']
>>> skindata.set_transform(id44)
>>> for bonedata in skindata.bone_list:
...     bonedata.set_transform(id44)
>>> affectedbones = geom.flatten_skin()
>>> [bone.name for bone in affectedbones]
[b'bone1', b'bone2', b'bone22', b'bone211']
>>> [child.name for child in skelroot.children]
[b'geom', b'bone1', b'bone21', b'bone2', b'bone22', b'bone211']
```

add_bone(*bone*, *vert_weights*)

Add bone with given vertex weights. After adding all bones, the geometry skinning information should be set from the current position of the bones using the L{update_bind_position} function.

Parameters

- **bone** – The bone NiNode block.
- **vert_weights** – A dictionary mapping each influenced vertex index to a vertex weight.

flatten_skin()

Reposition all bone blocks and geometry block in the tree to be direct children of the skeleton root.

Returns list of all used bones by the skin.

get_skin_deformation()

Returns a list of vertices and normals in their final position after skinning, in geometry space.

get_skin_partition()

Return the skin partition block.

get_vertex_weights()

Get vertex weights in a convenient format: list bone and weight per vertex.

is_skin()

Returns True if geometry is skinned.

send_bones_to_bind_position()

Send all bones to their bind position.

@deprecated: Use L{NifFormat.NiNode.send_bones_to_bind_position} instead of this function.

set_skin_partition(*skinpart*)

Set skin partition block.

update_bind_position()

Make current position of the bones the bind position for this geometry.

Sets the NiSkinData overall transform to the inverse of the geometry transform relative to the skeleton root, and sets the NiSkinData of each bone to the geometry transform relative to the skeleton root times the inverse of the bone transform relative to the skeleton root.

class NiGeometryData(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif._NiGeometryData`, `object`

```
>>> from pyffi.formats.nif import NifFormat
>>> geomdata = NifFormat.NiGeometryData()
```

(continues on next page)

(continued from previous page)

```
>>> geomdata.num_vertices = 3
>>> geomdata.has_vertices = True
>>> geomdata.has_normals = True
>>> geomdata.has_vertex_colors = True
>>> geomdata.num_uv_sets = 2
>>> geomdata.vertices.update_size()
>>> geomdata.normals.update_size()
>>> geomdata.vertex_colors.update_size()
>>> geomdata.uv_sets.update_size()
>>> geomdata.vertices[0].x = 1
>>> geomdata.vertices[0].y = 2
>>> geomdata.vertices[0].z = 3
>>> geomdata.vertices[1].x = 4
>>> geomdata.vertices[1].y = 5
>>> geomdata.vertices[1].z = 6
>>> geomdata.vertices[2].x = 1.200001
>>> geomdata.vertices[2].y = 3.400001
>>> geomdata.vertices[2].z = 5.600001
>>> geomdata.normals[0].x = 0
>>> geomdata.normals[0].y = 0
>>> geomdata.normals[0].z = 1
>>> geomdata.normals[1].x = 0
>>> geomdata.normals[1].y = 1
>>> geomdata.normals[1].z = 0
>>> geomdata.normals[2].x = 1
>>> geomdata.normals[2].y = 0
>>> geomdata.normals[2].z = 0
>>> geomdata.vertex_colors[1].r = 0.310001
>>> geomdata.vertex_colors[1].g = 0.320001
>>> geomdata.vertex_colors[1].b = 0.330001
>>> geomdata.vertex_colors[1].a = 0.340001
>>> geomdata.uv_sets[0][0].u = 0.990001
>>> geomdata.uv_sets[0][0].v = 0.980001
>>> geomdata.uv_sets[0][2].u = 0.970001
>>> geomdata.uv_sets[0][2].v = 0.960001
>>> geomdata.uv_sets[1][0].v = 0.910001
>>> geomdata.uv_sets[1][0].v = 0.920001
>>> geomdata.uv_sets[1][2].v = 0.930001
>>> geomdata.uv_sets[1][2].v = 0.940001
>>> for h in geomdata.get_vertex_hash_generator():
...     print(h)
(1000, 2000, 3000, 0, 0, 1000, 99000, 98000, 0, 92000, 0, 0, 0, 0)
(4000, 5000, 6000, 0, 1000, 0, 0, 0, 0, 0, 310, 320, 330, 340)
(1200, 3400, 5600, 1000, 0, 0, 97000, 96000, 0, 94000, 0, 0, 0, 0)
```

apply_scale(*scale*)

Apply scale factor on data.

get_vertex_hash_generator(*vertexprecision=3*, *normalprecision=3*, *uvprecision=5*, *vcolprecision=3*)

Generator which produces a tuple of integers for each (vertex, normal, uv, vcol), to ease detection of duplicate vertices. The precision parameters denote number of significant digits behind the comma.

Default for uvprecision should really be high because for very large models the uv coordinates can be very close together.

For vertexprecision, 3 seems usually enough (maybe we'll have to increase this at some point).

Parameters

- **vertexprecision** (*float*) – Precision to be used for vertices.
- **normalprecision** (*float*) – Precision to be used for normals.
- **uvprecision** (*float*) – Precision to be used for uvs.
- **vcolprecision** (*float*) – Precision to be used for vertex colors.

Returns A generator yielding a hash value for each vertex.

update_center_radius()
Recalculate center and radius of the data.

class NiGravity(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiParticleModifier

A particle modifier; applies a gravitational field on the particles.

property direction

property force

property position

property type

property unknown_float_1

class NiImage(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

property file_name

property image_data

property unknown_float

property unknown_int

property use_external

class NiInstancingMeshModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiMeshModifier

class NiIntegerExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Extra integer data.

property integer_data

class NiIntegersExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Integers data.

property data

property num_integers

class NiInterpController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

A controller capable of interpolation?

class NiInterpolator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Interpolator objects - function unknown.

class NiKeyBasedInterpolator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiInterpolator

Interpolator objects that use keys?

class NiKeyframeController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiSingleInterpController

A time controller object for animation key frames.

property data

class NiKeyframeData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._NiKeyframeData, object

apply_scale(*scale*)
Apply scale factor on data.

class NiLODData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Abstract class used for different types of LOD selections.

class NiLODNode(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiSwitchNode

Level of detail selector. Links to different levels of detail of the same model, used to switch a geometry at a specified distance.

property lod_center

property lod_level_data

property lod_levels

property num_lod_levels

class NiLight(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiDynamicEffect

Light source.

property ambient_color

property diffuse_color

property dimmer

property specular_color

class NiLightColorController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPoint3InterpController

Light color animation controller.

class NiLightDimmerController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiFloatInterpController

Unknown controller.

class NiLightIntensityController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiFloatInterpController

Unknown controller

class NiLines(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTriBasedGeom

Wireframe geometry.

class NiLinesData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiGeometryData

Wireframe geometry data.

property lines

class NiLookAtController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

Unknown. Start time is 3.4e+38 and stop time is -3.4e+38.

property look_at_node

property unknown_1

class NiLookAtInterpolator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiInterpolator

Unknown.

property look_at

property rotation

property scale

property target

property translation

property unknown_link_1

property unknown_link_2

property unknown_link_3

property unknown_short

class NiMaterialColorController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._NiMaterialColorController, object

get_target_color()
Get target color (works for all nif versions).

set_target_color(*target_color*)
Set target color (works for all nif versions).

class NiMaterialProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._NiMaterialProperty, object

is_interchangeable(*other*)
Are the two material blocks interchangeable?

class NiMesh(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiRenderObject

property bound

property datas

property instancing_enabled

property modifiers

property num_datas

property num_modifiers

property num_submeshes

property primitive_type

property unknown_100

property unknown_101

property unknown_102

property unknown_103

property unknown_200

property unknown_201

property unknown_250

property unknown_251

property unknown_300

property unknown_301

property unknown_302

property unknown_303

property unknown_350

property unknown_351

property unknown_400

property unknown_51

property unknown_52

property unknown_53

property unknown_54

property unknown_55

property unknown_56

class NiMeshHWInstance(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif.NiObject`

class NiMeshModifier(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif.NiObject`

Base class for mesh modifiers.

property complete_points

property num_complete_points

property num_submit_points

property submit_points

class NiMeshPSysData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysData

Particle meshes data.

property num_unknown_ints_1

property unknown_byte_3

property unknown_int_2

property unknown_ints_1

property unknown_node

class NiMeshParticleSystem(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiParticleSystem

Particle system.

class NiMorphController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiInterpController

Unknown! Used by Daoc->'healing.nif'.

class NiMorphData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._NiMorphData, object

apply_scale(*scale*)
Apply scale factor on data.

class NiMorphMeshModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiMeshModifier

Performs linear-weighted blending between a set of target data streams.

property elements

property flags

property num_elements

property num_targets

class NiMorphWeightsController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiInterpController

property interpolators

property num_interpolators

property num_targets

property target_names

property unknown_2

class NiMorpherController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiInterpController

Unknown! Used by Daoc.

property data

class NiMultiTargetTransformController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiInterpController

Unknown.

property extra_targets

property num_extra_targets

class NiMultiTextureProperty(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif.NiProperty

(note: not quite complete yet... but already reads most of the DAoC ones)

property flags

property texture_elements

property unknown_int

class NiNode(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif._NiNode, object

```
>>> from pyffi.formats.nif import NifFormat
>>> x = NifFormat.NiNode()
>>> y = NifFormat.NiNode()
>>> z = NifFormat.NiNode()
>>> x.num_children =1
>>> x.children.update_size()
>>> y in x.children
False
>>> x.children[0] = y
>>> y in x.children
True
>>> x.add_child(z, front = True)
>>> x.add_child(y)
>>> x.num_children
2
>>> x.children[0] is z
True
>>> x.remove_child(y)
>>> y in x.children
False
>>> x.num_children
1
>>> e = NifFormat.NiSpotLight()
>>> x.add_effect(e)
>>> x.num_effects
1
>>> e in x.effects
True
```

```
>>> from pyffi.formats.nif import NifFormat
>>> node = NifFormat.NiNode()
>>> child1 = NifFormat.NiNode()
>>> child1.name = "hello"
>>> child_2 = NifFormat.NiNode()
>>> child_2.name = "world"
>>> node.get_children()
[]
>>> node.set_children([child1, child_2])
>>> [child.name for child in node.get_children()]
[b'hello', b'world']
>>> [child.name for child in node.children]
[b'hello', b'world']
```

(continues on next page)

(continued from previous page)

```
>>> node.set_children([])
>>> node.get_children()
[]
>>> # now set them the other way around
>>> node.set_children([child_2, child1])
>>> [child.name for child in node.get_children()]
[b'world', b'hello']
>>> [child.name for child in node.children]
[b'world', b'hello']
>>> node.remove_child(child_2)
>>> [child.name for child in node.children]
[b'hello']
>>> node.add_child(child_2)
>>> [child.name for child in node.children]
[b'hello', b'world']
```

```
>>> from pyffi.formats.nif import NifFormat
>>> node = NifFormat.NiNode()
>>> effect1 = NifFormat.NiSpotLight()
>>> effect1.name = "hello"
>>> effect2 = NifFormat.NiSpotLight()
>>> effect2.name = "world"
>>> node.get_effects()
[]
>>> node.set_effects([effect1, effect2])
>>> [effect.name for effect in node.get_effects()]
[b'hello', b'world']
>>> [effect.name for effect in node.effects]
[b'hello', b'world']
>>> node.set_effects([])
>>> node.get_effects()
[]
>>> # now set them the other way around
>>> node.set_effects([effect2, effect1])
>>> [effect.name for effect in node.get_effects()]
[b'world', b'hello']
>>> [effect.name for effect in node.effects]
[b'world', b'hello']
>>> node.remove_effect(effect2)
>>> [effect.name for effect in node.effects]
[b'hello']
>>> node.add_effect(effect2)
>>> [effect.name for effect in node.effects]
[b'hello', b'world']
```

add_child(*child*, *front=False*)

Add block to child list.

Parameters **child** (*L{NifFormat.NiAVObject}*) – The child to add.

Keyword Arguments **front** – Whether to add to the front or to the end of the list (default is at end).

add_effect(*effect*)

Add an effect to the list of effects.

Parameters **effect** (*L{NifFormat.NiDynamicEffect}*) – The effect to add.

get_children()

Return a list of the children of the block.

Returns The list of children.
Return type `list` of L{NifFormat.NiAVObject}

get_effects()
Return a list of the effects of the block.
Returns The list of effects.
Return type `list` of L{NifFormat.NiDynamicEffect}

get_skinned_geometries()
This function yields all skinned geometries which have self as skeleton root.

merge_external_skeleton_root(*skelroot*)
Attach skinned geometry to self (which will be the new skeleton root of the nif at the given skeleton root). Use this function if you move a skinned geometry from one nif into a new NIF file. The bone links will be updated to point to the tree at self, instead of to the external tree.

merge_skeleton_roots()
This function will look for other geometries whose skeleton root is a (possibly indirect) child of this node. It will then reparent those geometries to this node. For example, it will unify the skeleton roots in Morrowind's cliffracer.nif file, or of the (official) body skins. This makes it much easier to import skeletons in for instance Blender: there will be only one skeleton root for each bone, over all geometries.

The merge fails for those geometries whose global skin data transform does not match the inverse geometry transform relative to the skeleton root (the maths does not work out in this case!)

Returns list of all new blocks that have been reparented (and added to the skeleton root children list), and a list of blocks for which the merge failed.

remove_child(*child*)
Remove a block from the child list.
Parameters **child** (*L{NifFormat.NiAVObject}*) – The child to remove.

remove_effect(*effect*)
Remove a block from the effect list.
Parameters **effect** (*L{NifFormat.NiDynamicEffect}*) – The effect to remove.

send_bones_to_bind_position()
This function will send all bones of geometries of this skeleton root to their bind position. For best results, call L{send_geometries_to_bind_position} first.
Returns A number quantifying the remaining difference between bind positions.
Return type `float`

send_detached_geometries_to_node_position()
Some nifs (in particular in Morrowind) have geometries that are skinned but that do not share bones. In such cases, send_geometries_to_bind_position cannot reposition them. This function will send such geometries to the position of their root node.

Examples of such nifs are the official Morrowind skins (after merging skeleton roots).

Returns list of detached geometries that have been moved.

send_geometries_to_bind_position()
Call this on the skeleton root of geometries. This function will transform the geometries, such that all skin data transforms coincide, or at least coincide partially.
Returns A number quantifying the remaining difference between bind positions.
Return type `float`

set_children(*childlist*)
Set the list of children from the given list (destroys existing list).

Parameters **childlist** (list of L{NifFormat.NiAVObject}) – The list of child blocks to set.

set_effects(*effectlist*)
Set the list of effects from the given list (destroys existing list).
Parameters **effectlist** (list of L{NifFormat.NiDynamicEffect}) – The list of effect blocks to set.

class NiObject(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._NiObject, object

apply_scale(*scale*)
Scale data in this block. This implementation does nothing. Override this method if it contains geometry data that can be scaled.

find(*block_name=None*, *block_type=None*)

find_chain(*block*, *block_type=None*)
Finds a chain of blocks going from C{self} to C{block}. If found, self is the first element and block is the last element. If no branch found, returns an empty list. Does not check whether there is more than one branch; if so, the first one found is returned.
Parameters
- **block** – The block to find a chain to.
- **block_type** – The type that blocks should have in this chain.

is_interchangeable(*other*)
Are the two blocks interchangeable?

@todo: Rely on AnyType, SimpleType, ComplexType, etc. implementation.

tree(*block_type=None*, *follow_all=True*, *unique=False*)
A generator for parsing all blocks in the tree (starting from and including C{self}).
Parameters
- **block_type** – If not None, yield only blocks of the type C{block_type}.
- **follow_all** – If C{block_type} is not None, then if this is True the function will parse the whole tree. Otherwise, the function will not follow branches that start by a non-C{block_type} block.
- **unique** – Whether the generator can return the same block twice or not.

class NiObjectNET(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._NiObjectNET, object

add_controller(*ctrlblock*)
Add block to controller chain and set target of controller to self.

add_extra_data(*extrablock*)
Add block to extra data list and extra data chain. It is good practice to ensure that the extra data has empty next_extra_data field when adding it to avoid loops in the hierarchy.

add_integer_extra_data(*name*, *value*)
Add a particular extra integer data block.

get_controllers()
Get a list of all controllers.

get_extra_datas()
Get a list of all extra data blocks.

remove_extra_data(*extrablock*)
Remove block from extra data list and extra data chain.

```
>>> from pyffi.formats.nif import NifFormat
>>> block = NifFormat.NiNode()
>>> block.num_extra_data_list = 3
>>> block.extra_data_list.update_size()
>>> extrablock = NifFormat.NiStringExtraData()
>>> block.extra_data_list[1] = extrablock
>>> block.remove_extra_data(extrablock)
>>> [extra for extra in block.extra_data_list]
[None, None]
```

set_extra_datas(*extralist*)

Set all extra data blocks from given list (erases existing data).

```
>>> from pyffi.formats.nif import NifFormat
>>> node = NifFormat.NiNode()
>>> extra1 = NifFormat.NiExtraData()
>>> extra1.name = "hello"
>>> extra2 = NifFormat.NiExtraData()
>>> extra2.name = "world"
>>> node.get_extra_datas()
[]
>>> node.set_extra_datas([extra1, extra2])
>>> [extra.name for extra in node.get_extra_datas()]
[b'hello', b'world']
>>> [extra.name for extra in node.extra_data_list]
[b'hello', b'world']
>>> node.extra_data is extra1
True
>>> extra1.next_extra_data is extra2
True
>>> extra2.next_extra_data is None
True
>>> node.set_extra_datas([])
>>> node.get_extra_datas()
[]
>>> # now set them the other way around
>>> node.set_extra_datas([extra2, extra1])
>>> [extra.name for extra in node.get_extra_datas()]
[b'world', b'hello']
>>> [extra.name for extra in node.extra_data_list]
[b'world', b'hello']
>>> node.extra_data is extra2
True
>>> extra2.next_extra_data is extra1
True
>>> extra1.next_extra_data is None
True
```

Parameters **extralist** (`list` of L{NifFormat.NiExtraData}) – List of extra data blocks to add.

class NiPSBombForce(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif.NiObject`

property name

property unknown_1

property unknown_10

property unknown_2

property unknown_3

property unknown_4

property unknown_5

property unknown_6

property unknown_7

property unknown_8

property unknown_9

class NiPSBoundUpdater(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

property unknown_1

property unknown_2

class NiPSBoxEmitter(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

property name

property unknown_1

property unknown_10

property unknown_11

property unknown_12

property unknown_13

property unknown_14

property unknown_15

property unknown_16

property unknown_17

property unknown_18

property unknown_19

property unknown_2

property unknown_20

property unknown_21

property unknown_22

property unknown_23

property unknown_24

property unknown_25

property unknown_26

property unknown_27

property unknown_28

property unknown_29

property unknown_3

property unknown_30

property unknown_31

property unknown_32

property unknown_33

property unknown_34

property unknown_35

property unknown_36

property unknown_37

property unknown_38

property unknown_39

property unknown_4

property unknown_40

property unknown_41

property unknown_42

property unknown_43

property unknown_44

property unknown_45

property unknown_46

property unknown_47

property unknown_48

property unknown_5

property unknown_6

property unknown_7

property unknown_8

property unknown_9

class NiPSCylinderEmitter(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif.NiPSSphereEmitter`

property unknown_23

class NiPSDragForce(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif.NiObject`

property unknown_1

property unknown_10

property unknown_2

property unknown_3

property unknown_4

property unknown_5

property unknown_6

property unknown_7

property unknown_8

property unknown_9

class NiPSEmitParticlesCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysEmitterCtlr

class NiPSEmitterDeclinationCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierCtlr

class NiPSEmitterDeclinationVarCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSEmitterDeclinationCtlr

class NiPSEmitterLifeSpanCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierCtlr

class NiPSEmitterPlanarAngleCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierCtlr

class NiPSEmitterPlanarAngleVarCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSEmitterPlanarAngleCtlr

class NiPSEmitterRadiusCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

property interpolator

property unknown_2

class NiPSEmitterRotAngleCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierCtlr

class NiPSEmitterRotAngleVarCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSEmitterRotAngleCtlr

class NiPSEmitterRotSpeedCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierCtlr

class NiPSEmitterRotSpeedVarCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSEmitterRotSpeedCtlr

class NiPSEmitterSpeedCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

property interpolator

property unknown_3

class NiPSFacingQuadGenerator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

property unknown_1

property unknown_10

property unknown_11

property unknown_12

property unknown_2

property unknown_3

property unknown_4

property unknown_5

property unknown_6

property unknown_7

property unknown_8

property unknown_9

class NiPSForceActiveCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

property interpolator

property unknown_2

class NiPSGravityForce(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

property unknown_1

property unknown_10

property unknown_11

property unknown_12

property unknown_13

property unknown_14

property unknown_15

property unknown_16

property unknown_17

property unknown_18

property unknown_19

property unknown_2

property unknown_20

property unknown_21

property unknown_22

property unknown_23

property unknown_24

property unknown_25

property unknown_26

property unknown_27

property unknown_28

property unknown_29

property unknown_3

property unknown_30

property unknown_31

property unknown_32

property unknown_33

property unknown_34

property unknown_35

property unknown_36

property unknown_4

property unknown_5

property unknown_6

property unknown_7

property unknown_8

property unknown_9

class NiPSGravityStrengthCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

property unknown_2

property unknown_3

class NiPSMeshEmitter(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

property name

property unknown_1

property unknown_10

property unknown_11

property unknown_12

property unknown_13

property unknown_14

property unknown_15

property unknown_16

property unknown_17

property unknown_18

property unknown_19

property unknown_2

property unknown_20

property unknown_21

property unknown_22

property unknown_23

property unknown_24

property unknown_25

property unknown_26

property unknown_27

property unknown_28

property unknown_3

property unknown_4

property unknown_5

property unknown_6

property unknown_7

property unknown_8

property unknown_9

class NiPSMeshParticleSystem(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif.NiPSParticleSystem

property unknown_23

property unknown_24

property unknown_25

property unknown_26

class NiPSParticleSystem(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif.NiAVObject

property emitter

property generator

property simulator

property unknown_10

property unknown_11

property unknown_12

property unknown_15

property unknown_16

property unknown_17

property unknown_19

property unknown_20

property unknown_21

property unknown_22

property unknown_27

property unknown_28

property unknown_29

property unknown_3

property unknown_30

property unknown_31

property unknown_32

property unknown_33

property unknown_34

property unknown_35

property unknown_36

property unknown_37

property unknown_38

property unknown_39

property unknown_4

property unknown_5

property unknown_6

property unknown_7

property unknown_8

property unknown_9

class NiPSPlanarCollider(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

property name

property unknown_byte_4

property unknown_floats_5

property unknown_int_1

property unknown_int_2

property unknown_link_6

property unknown_short_3

class NiPSResetOnLoopCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

class NiPSSimulator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiMeshModifier

The mesh modifier that performs all particle system simulation.

property num_simulation_steps

property simulation_steps

class NiPSSimulatorCollidersStep(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSSimulatorStep

Encapsulates a floodgate kernel that simulates particle colliders.

property colliders

property num_colliders

class NiPSSimulatorFinalStep(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSSimulatorStep

Encapsulates a floodgate kernel that updates particle positions and ages. As indicated by its name, this step should be attached last in the NiPSSimulator mesh modifier.

class NiPSSimulatorForcesStep(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSSimulatorStep

Encapsulates a floodgate kernel that simulates particle forces.

property forces

property num_forces

class NiPSSimulatorGeneralStep(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSSimulatorStep

Encapsulates a floodgate kernel that updates particle size, colors, and rotations.

property color_keys

property color_loop_behavior

property grow_generation

property grow_time

property num_color_keys

property num_rotation_keys

property num_size_keys

property rotation_keys

property rotation_loop_behavior

property shrink_generation

property shrink_time

property size_keys

property size_loop_behavior

property unknown_1

property unknown_2

property unknown_3

class NiPSSimulatorMeshAlignStep(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSSimulatorStep

Encapsulates a floodgate kernel that updates mesh particle alignment and transforms.

property num_rotation_keys

property rotation_keys

property rotation_loop_behavior

class NiPSSimulatorStep(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Abstract base class for a single step in the particle system simulation process. It has no seralized data.

class NiPSSpawner(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

class NiPSSphereEmitter(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

property name

property unknown_10

property unknown_11

property unknown_12

property unknown_13

property unknown_14

property unknown_15

property unknown_16

property unknown_17

property unknown_18

property unknown_19

property unknown_2

property unknown_20

property unknown_21

property unknown_22

property unknown_3

property unknown_4

property unknown_5

property unknown_6

property unknown_7

property unknown_8

property unknown_9

class NiPSSphericalCollider(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif.NiObject`

property unknown_1

property unknown_2

property unknown_3

property unknown_4

property unknown_5

property unknown_6

property unknown_7

class NiPSysAgeDeathModifier(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif.NiPSysModifier`

Unknown particle modifier.

property spawn_modifier

property spawn_on_death

class NiPSysAirFieldAirFrictionCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierFloatCtlr

Particle system controller for air field air friction.

class NiPSysAirFieldInheritVelocityCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierFloatCtlr

Particle system controller for air field inherit velocity.

class NiPSysAirFieldModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysFieldModifier

Particle system modifier, used for controlling the particle velocity in a field like wind.

property direction

property unknown_boolean_1

property unknown_boolean_2

property unknown_boolean_3

property unknown_float_2

property unknown_float_3

property unknown_float_4

class NiPSysAirFieldSpreadCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierFloatCtlr

Particle system controller for air field spread.

class NiPSysBombModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

Particle modifier that uses a NiNode to use as a "Bomb Object" to alter the path of particles.

property bomb_axis

property bomb_object

property decay

property decay_type

property delta_v

property symmetry_type

class NiPSysBoundUpdateModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

Unknown particle system modifier.

property update_skip

class NiPSysBoxEmitter(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysVolumeEmitter

Particle emitter that uses points within a defined Box shape to emit from..

property depth

property height

property width

class NiPSysCollider(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Particle system collider.

property bounce

property collider_object

property die_on_collide

property next_collider

property parent

property spawn_modifier

property spawn_on_collide

class NiPSysColliderManager(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

Particle modifier that adds a defined shape to act as a collision object for particles to interact with.

property collider

class NiPSysColorModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

Particle modifier that adds keyframe data to modify color/alpha values of particles over time.

property data

class NiPSysCylinderEmitter(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysVolumeEmitter

Particle emitter that uses points within a defined Cylinder shape to emit from.

property height

property radius

class NiPSysData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiRotatingParticlesData

Particle system data.

property aspect_ratio

property has_subtexture_offset_u_vs

property has_unknown_floats_3

property num_subtexture_offset_u_vs

property particle_descriptions

property subtexture_offset_u_vs

property unknown_byte_4

property unknown_floats_3

property unknown_int_4

property unknown_int_5

property unknown_int_6

property unknown_short_1

property unknown_short_2

property unknown_short_3

class NiPSysDragFieldModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysFieldModifier

Particle system modifier, used for controlling the particle velocity in drag space warp.

property direction

property use_direction

class NiPSysDragModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

Unknown.

property drag_axis

property parent

property percentage

property range

property range_falloff

class NiPSysEmitter(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

A particle emitter?

property declination

property declination_variation

property initial_color

property initial_radius

property life_span

property life_span_variation

property planar_angle

property planar_angle_variation

property radius_variation

property speed

property speed_variation

class NiPSysEmitterCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierCtlr

Particle system emitter controller.

property data

property visibility_interpolator

class NiPSysEmitterCtlrData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Particle system emitter controller data.

property float_keys

property num_visibility_keys

property visibility_keys

class NiPSysEmitterDeclinationCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierFloatCtlr

Unknown.

class NiPSysEmitterDeclinationVarCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierFloatCtlr

Unknown.

class NiPSysEmitterInitialRadiusCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierFloatCtlr

Unknown.

class NiPSysEmitterLifeSpanCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierFloatCtlr

Unknown.

class NiPSysEmitterPlanarAngleCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierFloatCtlr

Particle system controller for emitter planar angle.

class NiPSysEmitterPlanarAngleVarCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierFloatCtlr

Particle system controller for emitter planar angle variation.

class NiPSysEmitterSpeedCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierFloatCtlr

Unknown.

class NiPSysFieldAttenuationCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierFloatCtlr

Particle system controller for force field attenuation.

class NiPSysFieldMagnitudeCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierFloatCtlr

Particle system controller for force field magnitude.

class NiPSysFieldMaxDistanceCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierFloatCtlr

Particle system controller for force field maximum distance.

class NiPSysFieldModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

Base for all force field particle modifiers.

property attenuation

property field_object

property magnitude

property max_distance

property use_max_distance

class NiPSysGravityFieldModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysFieldModifier

Particle system modifier, used for controlling the particle velocity in gravity field.

property direction

class NiPSysGravityModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

Adds gravity to a particle system, when linked to a NiNode to use as a Gravity Object.

property decay

property force_type

property gravity_axis

property gravity_object

property strength

property turbulence

property turbulence_scale

property unknown_byte

class NiPSysGravityStrengthCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierFloatCtlr

Unknown.

class NiPSysGrowFadeModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

Particle modifier that controls the time it takes to grow a particle from Size=0 to the specified Size in the emitter, and then back to 0. This modifer has no control over alpha settings.

property base_scale

property fade_generation

property fade_time

property grow_generation

property grow_time

class NiPSysInitialRotAngleCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierFloatCtlr

Particle system controller for emitter initial rotation angle.

class NiPSysInitialRotAngleVarCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierFloatCtlr

Particle system controller for emitter initial rotation angle variation.

class NiPSysInitialRotSpeedCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierFloatCtlr

Particle system controller for emitter initial rotation speed.

class NiPSysInitialRotSpeedVarCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierFloatCtlr

Particle system controller for emitter initial rotation speed variation.

class NiPSysMeshEmitter(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysEmitter

Particle emitter that uses points on a specified mesh to emit from.

property emission_axis

property emission_type

property emitter_meshes

property initial_velocity_type

property num_emitter_meshes

class NiPSysMeshUpdateModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

Unknown.

property meshes

property num_meshes

class NiPSysModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Generic particle system modifier object.

property active

property name

property order

property target

class NiPSysModifierActiveCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierBoolCtlr

Unknown.

property data

class NiPSysModifierBoolCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierCtlr

A particle system modifier controller that deals with boolean data?

class NiPSysModifierCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiSingleInterpController

A particle system modifier controller.

property modifier_name

class NiPSysModifierFloatCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifierCtlr

A particle system modifier controller that deals with floating point data?

property data

class NiPSysPlanarCollider(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysCollider

Particle Collider object which particles will interact with.

property height

property width

property x_axis

property y_axis

class NiPSysPositionModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

Unknown particle system modifier.

class NiPSysRadialFieldModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysFieldModifier

Particle system modifier, used for controlling the particle velocity in force field.

property radial_type

class NiPSysResetOnLoopCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

Unknown.

class NiPSysRotationModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

Particle modifier that adds rotations to particles.

property initial_axis

property initial_rotation_angle

property initial_rotation_angle_variation

property initial_rotation_speed

property initial_rotation_speed_variation

property random_initial_axis

property random_rot_speed_sign

class NiPSysSpawnModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysModifier

Unknown particle modifier.

property life_span

property life_span_variation

property max_num_to_spawn

property min_num_to_spawn

property num_spawn_generations

property percentage_spawned

property spawn_dir_chaos

property spawn_speed_chaos

property unknown_int

class NiPSysSphereEmitter(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysVolumeEmitter

Particle emitter that uses points within a sphere shape to emit from.

property radius

class NiPSysSphericalCollider(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysCollider

Particle Collider object which particles will interact with.

property radius

class NiPSysTrailEmitter(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysEmitter

Guild 2-Specific node

property unknown_float_1

property unknown_float_2

property unknown_float_3

property unknown_float_4

property unknown_float_5

property unknown_float_6

property unknown_float_7

property unknown_int_1

property unknown_int_2

property unknown_int_3

property unknown_int_4

class NiPSysTurbulenceFieldModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysFieldModifier

Particle system modifier, used for controlling the particle velocity in drag space warp.

property frequency

class NiPSysUpdateCtlr(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

Particle system controller, used for ???.

class NiPSysVolumeEmitter(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysEmitter

An emitter that emits meshes?

property emitter_object

class NiPSysVortexFieldModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPSysFieldModifier

Particle system modifier, used for controlling the particle velocity in force field.

property direction

class NiPalette(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

A color palette.

property num_entries

property palette

property unknown_byte

class NiParticleBomb(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiParticleModifier

A particle modifier.

property decay

property decay_type

property delta_v

property direction

property duration

property position

property start

property symmetry_type

class NiParticleColorModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiParticleModifier

Unknown.

property color_data

class NiParticleGrowFade(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiParticleModifier

This particle system modifier controls the particle size. If it is present the particles start with size 0.0 . Then they grow to their original size and stay there until they fade to zero size again at the end of their lifetime cycle.

property fade

property grow

class NiParticleMeshModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiParticleModifier

Unknown.

property num_particle_meshes

property particle_meshes

class NiParticleMeshes(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiParticles

Mesh particle node?

class NiParticleMeshesData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiRotatingParticlesData

Particle meshes data.

property unknown_link_2

class NiParticleModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

A particle system modifier.

property controller

property next_modifier

class NiParticleRotation(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiParticleModifier

Unknown.

property initial_axis

property random_initial_axis

property rotation_speed

class NiParticleSystem(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiParticles

A particle system.

property modifiers

property num_modifiers

property unknown_int_1

property unknown_short_2

property unknown_short_3

property world_space

class NiParticleSystemController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

A generic particle system time controller object.

property color_data

property emit_flags

property emit_rate

property emit_start_time

property emit_stop_time

property emitter

property horizontal_angle

property horizontal_direction

property lifetime

property lifetime_random

property num_particles

property num_valid

property old_emit_rate

property old_speed

property particle_extra

property particle_lifetime

property particle_link

property particle_timestamp

property particle_unknown_short

property particle_unknown_vector

property particle_velocity

property particle_vertex_id

property particles

property size

property speed

property speed_random

property start_random

property trailer

property unknown_byte

property unknown_color

property unknown_float_1

property unknown_float_13

property unknown_floats_2

property unknown_int_1

property unknown_int_2

property unknown_link

property unknown_link_2

property unknown_normal

property unknown_short_2

property unknown_short_3

property vertical_angle

property vertical_direction

class NiParticles(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif.NiGeometry`

Generic particle system node.

class NiParticlesData(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif.NiGeometryData`

Generic rotating particles data object.

property has_radii

property has_rotation_angles

property has_rotation_axes

property has_rotations

property has_sizes

property has_uv_quadrants

property num_active

property num_particles

property num_uv_quadrants

property particle_radius

property radii

property rotation_angles

property rotation_axes

property rotations

property sizes

property unknown_byte_1

property unknown_byte_2

property unknown_link

property uv_quadrants

class NiPathController(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif.NiTimeController`

Time controller for a path.

property float_data

property pos_data

property unknown_float_2

property unknown_float_3

property unknown_int_1

property unknown_short

property unknown_short_2

class NiPathInterpolator(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif.NiKeyBasedInterpolator`

Unknown interpolator.

property float_data

property pos_data

property unknown_float_1

property unknown_float_2

property unknown_int

property unknown_short

property unknown_short_2

class NiPersistentSrcTextureRendererData(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif.ATextureRenderData

property num_faces

property num_pixels

property pixel_data

property unknown_int_6

property unknown_int_7

class NiPhysXActorDesc(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif.NiObject

Unknown PhysX node.

property shape_description

property unknown_byte_1

property unknown_byte_2

property unknown_int_1

property unknown_int_2

property unknown_int_4

property unknown_int_5

property unknown_int_6

property unknown_quat_1

property unknown_quat_2

property unknown_quat_3

property unknown_ref_0

property unknown_ref_1

property unknown_ref_2

property unknown_refs_3

class NiPhysXBodyDesc(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif.NiObject

Unknown PhysX node.

property unknown_bytes

class NiPhysXD6JointDesc(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif.NiObject

Unknown PhysX node.

property unknown_bytes

class NiPhysXKinematicSrc(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif.NiObject

Unknown PhysX node.

property unknown_bytes

class NiPhysXMaterialDesc(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Unknown node.

property unknown_byte_1

property unknown_byte_2

property unknown_int

class NiPhysXMeshDesc(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Unknown PhysX node.

property num_vertices

property unknown_byte_1

property unknown_byte_2

property unknown_bytes_0

property unknown_bytes_1

property unknown_bytes_2

property unknown_bytes_3

property unknown_float_1

property unknown_float_2

property unknown_int_1

property unknown_int_2

property unknown_int_4

property unknown_ints_1

property unknown_short_1

property unknown_short_2

property unknown_shorts_1

property vertices

class NiPhysXProp(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObjectNET

Unknown PhysX node.

property num_dests

property prop_description

property transform_dests

property unknown_byte

property unknown_float_1

property unknown_int

property unknown_int_1

property unknown_refs_1

class NiPhysXPropDesc(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Unknown PhysX node.

property actor_descs

property joint_descs

property material_descs

property num_dests

property num_joints

property num_materials

property unknown_byte_6

property unknown_int_1

property unknown_int_2

property unknown_int_3

property unknown_int_5

property unknown_string_4

class NiPhysXShapeDesc(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Unknown PhysX node.

property mesh_description

property unknown_float_1

property unknown_float_2

property unknown_float_3

property unknown_int_1

property unknown_int_2

property unknown_int_3

property unknown_int_4

property unknown_int_5

property unknown_int_7

property unknown_int_8

property unknown_quat_1

property unknown_quat_2

property unknown_quat_3

property unknown_short_1

property unknown_short_2

class NiPhysXTransformDest(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Unknown PhysX node.

property node

property unknown_byte_1

property unknown_byte_2

class NiPixelData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.ATextureRenderData

A texture.

property num_faces

property num_pixels

property pixel_data

class NiPlanarCollider(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiParticleModifier

Unknown.

property unknown_float_1

property unknown_float_10

property unknown_float_11

property unknown_float_12

property unknown_float_13

property unknown_float_14

property unknown_float_15

property unknown_float_16

property unknown_float_2

property unknown_float_3

property unknown_float_4

property unknown_float_5

property unknown_float_6

property unknown_float_7

property unknown_float_8

property unknown_float_9

property unknown_short

property unknown_short_2

class NiPoint3InterpController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiSingleInterpController

A controller that interpolates point 3 data?

property data

property target_color

class NiPoint3Interpolator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiKeyBasedInterpolator

Unknown.

property data

property point_3_value

class NiPointLight(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiLight

A point light.

property constant_attenuation

property linear_attenuation

property quadratic_attenuation

class NiPortal(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiAVObject

A Portal

property num_vertices

property target

property unknown_flags

property unknown_short_2

property vertices

class NiPosData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Position data.

property data

class NiProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObjectNET

A generic property object.

class NiRangeLODData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiLODData

Describes levels of detail based on distance of object from camera.

property lod_center

property lod_levels

property num_lod_levels

class NiRawImageData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Raw image data.

property height

property image_type

property rgb_image_data

property rgba_image_data

property width

class NiRenderObject(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiAVObject

An object that can be rendered.

property active_material

property material_data

property material_needs_update_default

property num_materials

class NiRollController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiSingleInterpController

Unknown.

property data

class NiRoom(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

Grouping node for nodes in a Portal

property in_portals

property items

property num_in_portals

property num_items

property num_portals_2

property num_walls

property portals_2

property wall_plane

class NiRoomGroup(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

Grouping node for nodes in a Portal

property num_rooms

property rooms

property shell_link

class NiRotatingParticles(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiParticles

Unknown.

class NiRotatingParticlesData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiParticlesData

Rotating particles data object.

property has_rotations_2

property rotations_2

class NiScreenElements(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTriShape

Two dimensional screen elements.

class NiScreenElementsData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTriShapeData

Two dimensional screen elements.

property max_polygons

property num_polygons

property polygon_indices

property polygons

property unknown_u_short_1

property unknown_u_short_2

property unknown_u_short_3

property used_triangle_points

property used_vertices

class NiScreenLODData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiLODData

Describes levels of detail based on size of object on screen?

property bound_center

property bound_radius

property proportion_count

property proportion_levels

property world_center

property world_radius

class NiSequence(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Root node used in some Empire Earth II .kf files (version 4.2.2.0).

property controlled_blocks

property name

property num_controlled_blocks

property text_keys

property text_keys_name

property unknown_int_1

property unknown_int_4

property unknown_int_5

class NiSequenceData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

class NiSequenceStreamHelper(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObjectNET

Keyframe animation root node, in .kf files.

class NiShadeProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiProperty

Determines whether flat shading or smooth shading is used on a shape.

property flags

class NiShadowGenerator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

property name

property num_unknown_links_1

property target

property unknown_flags

property unknown_links_1

property unkown_byte_5

property unkown_byte_9

property unkown_float_4

property unkown_int_2

property unkown_int_6

property unkown_int_7

property unkown_int_8

class NiSingleInterpController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiInterpController

A controller referring to a single interpolator.

property interpolator

class NiSkinData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._NiSkinData, object

apply_scale(*scale*)
Apply scale factor on data.

```
>>> from pyffi.formats.nif import NifFormat
>>> id44 = NifFormat.Matrix44()
>>> id44.set_identity()
>>> skelroot = NifFormat.NiNode()
>>> skelroot.name = 'Scene Root'
>>> skelroot.set_transform(id44)
>>> bone1 = NifFormat.NiNode()
>>> bone1.name = 'bone1'
>>> bone1.set_transform(id44)
>>> bone1.translation.x = 10
>>> skelroot.add_child(bone1)
>>> geom = NifFormat.NiTriShape()
>>> geom.set_transform(id44)
```

(continues on next page)

(continued from previous page)

```
>>> skelroot.add_child(geom)
>>> skininst = NifFormat.NiSkinInstance()
>>> geom.skin_instance = skininst
>>> skininst.skeleton_root = skelroot
>>> skindata = NifFormat.NiSkinData()
>>> skininst.data = skindata
>>> skindata.set_transform(id44)
>>> geom.add_bone(bone1, {})
>>> geom.update_bind_position()
>>> bone1.translation.x
10.0
>>> skindata.bone_list[0].skin_transform.translation.x
-10.0
>>> import pyffi.spells.nif.fix
>>> import pyffi.spells.nif
>>> data = NifFormat.Data()
>>> data.roots = [skelroot]
>>> toaster = pyffi.spells.nif.NifToaster()
>>> toaster.scale = 0.1
>>> pyffi.spells.nif.fix.SpellScale(data=data, toaster=toaster).recurse()
pyffi.toaster:INFO:--- fix_scale ---
pyffi.toaster:INFO:  scaling by factor 0.100000
pyffi.toaster:INFO:  ~~~ NiNode [Scene Root] ~~~
pyffi.toaster:INFO:    ~~~ NiNode [bone1] ~~~
pyffi.toaster:INFO:    ~~~ NiTriShape [] ~~~
pyffi.toaster:INFO:      ~~~ NiSkinInstance [] ~~~
pyffi.toaster:INFO:        ~~~ NiSkinData [] ~~~
>>> bone1.translation.x
1.0
>>> skindata.bone_list[0].skin_transform.translation.x
-1.0
```

get_transform()
Return scale, rotation, and translation into a single 4x4 matrix.

set_transform(*mat*)
Set rotation, transform, and velocity.

class NiSkinInstance(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Skinning instance.

property bones

property data

property num_bones

property skeleton_root

property skin_partition

class NiSkinPartition(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Skinning data, optimized for hardware skinning. The mesh is partitioned in submeshes such that each vertex of a submesh is influenced only by a limited and fixed number of bones.

property num_skin_partition_blocks

property skin_partition_blocks

class NiSkinningLODController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

class NiSkinningMeshModifier(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiMeshModifier

property bone_bounds

property bone_transforms

property bones

property flags

property num_bones

property skeleton_root

property skeleton_transform

class NiSortAdjustNode(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

Unknown node. Found in Loki.

property sorting_mode

property unknown_int_2

class NiSourceCubeMap(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiSourceTexture

Unknown node. Found in Emerge Demo.

class NiSourceTexture(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTexture

Describes texture source and properties.

property alpha_format

property direct_render

property file_name

property is_static

property persist_render_data

property pixel_data

property pixel_layout

property unknown_byte

property unknown_link

property use_external

property use_mipmaps

class NiSpecularProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiProperty

Gives specularity to a shape. Flags 0x0001.

property flags

class NiSphericalCollider(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiParticleModifier

Unknown.

property unknown_float_1

property unknown_float_2

property unknown_float_3

property unknown_float_4

property unknown_float_5

property unknown_short_1

property unknown_short_2

class NiSpotLight(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiPointLight

A spot.

property cutoff_angle

property exponent

property unknown_float

class NiStencilProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiProperty

Allows control of stencil testing.

property draw_mode

property fail_action

property flags

property pass_action

property stencil_enabled

property stencil_function

property stencil_mask

property stencil_ref

property z_fail_action

class NiStringExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Apparently commands for an optimizer instructing it to keep things it would normally discard. Also refers to NiNode objects (through their name) in animation .kf files.

property bytes_remaining

property string_data

class NiStringPalette(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

List of 0x00-seperated strings, which are names of controlled objects and controller types. Used in .kf files in conjunction with NiControllerSequence.

property palette

class NiStringsExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

List of strings; for example, a list of all bone names.

property data

property num_strings

class NiSwitchNode(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiNode

A node used to switch between branches, such as for LOD levels?

property unknown_flags_1

property unknown_int_1

class NiTextKeyExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Extra data, used to name different animation sequences.

property num_text_keys

property text_keys

property unknown_int_1

class NiTexture(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObjectNET

A texture.

class NiTextureEffect(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiDynamicEffect

Enables environment mapping. Should be in both the children list and effects list of the NiTriShape object. For Morrowind: the bump map can be used to bump the environment map (note that the bump map is ignored if no NiTextureEffect object is present).

property clipping_plane

property coordinate_generation_type

property image

property model_projection_matrix

property model_projection_transform

property ps_2_k

property ps_2_l

property source_texture

property texture_clamping

property texture_filtering

property texture_type

property unknown

property unknown_float

property unknown_short

property unknown_vector

class NiTextureModeProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiProperty

Unknown

property ps_2_k

property ps_2_l

property unknown_ints

property unknown_short

class NiTextureProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiProperty

property flags

property image

property unknown_ints_1

property unknown_ints_2

class NiTextureTransformController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiFloatInterpController

Texture transformation controller. The target texture slot should have "Has Texture Transform" enabled.

property data

property operation

property texture_slot

property unknown_2

class NiTexturingProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiProperty

Describes an object's textures.

property apply_mode

property base_texture

property bump_map_luma_offset

property bump_map_luma_scale

property bump_map_matrix

property bump_map_texture

property dark_texture

property decal_0_texture

property decal_1_texture

property decal_2_texture

property decal_3_texture

property detail_texture

property flags

property gloss_texture

property glow_texture

property has_base_texture

property has_bump_map_texture

property has_dark_texture

property has_decal_0_texture

property has_decal_1_texture

property has_decal_2_texture

property has_decal_3_texture

property has_detail_texture

property has_gloss_texture

property has_glow_texture

property has_normal_texture

property has_unknown_2_texture

property normal_texture

property num_shader_textures

property shader_textures

property texture_count

property unknown_2_float

property unknown_2_texture

class NiTimeController(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif.NiObject`

A generic time controller object.

property flags

property frequency

property next_controller

property phase

property start_time

property stop_time

property target

property unknown_integer

class NiTransformController(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif.NiKeyframeController`

NiTransformController replaces the NiKeyframeController.

class NiTransformData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiKeyframeData

Mesh animation keyframe data.

class NiTransformEvaluator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

class NiTransformInterpolator(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._NiTransformInterpolator, object

apply_scale(*scale*)
Apply scale factor <scale> on data.

class NiTransparentProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiProperty

Unknown

property unknown

class NiTriBasedGeom(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._NiTriBasedGeom, object

get_interchangeable_tri_shape(*triangles=None*)
Returns a NiTriShape block that is geometrically interchangeable. If you do not want to set the triangles from the original shape, use the triangles argument.

get_interchangeable_tri_strips(*strips=None*)
Returns a NiTriStrips block that is geometrically interchangeable. If you do not want to set the strips from the original shape, use the strips argument.

get_tangent_space()
Return iterator over normal, tangent, bitangent vectors. If the block has no tangent space, then returns None.

update_skin_center_radius()
Update centers and radii of all skin data fields.

update_skin_partition(*maxbonesperpartition=4*, *maxbonespervertex=4*, *verbose=0*, *stripify=True*, *stitchstrips=False*, *padbones=False*, *triangles=None*, *trianglepartmap=None*, *maximize_bone_sharing=False*)
Recalculate skin partition data.

Deprecated Do not use the verbose argument.

Parameters

- **maxbonesperpartition** – Maximum number of bones in each partition. The num_bones field will not exceed this number.
- **maxbonespervertex** – Maximum number of bones per vertex. The num_weights_per_vertex field will be exactly equal to this number.
- **verbose** – Ignored, and deprecated. Set pyffi's log level instead.
- **stripify** – If true, stripify the partitions, otherwise use triangles.
- **stitchstrips** – If stripify is true, then set this to true to stitch the strips.
- **padbones** – Enforces the numbones field to be equal to maxbonesperpartition. Also ensures that the bone indices are unique and sorted, per vertex. Raises an exception if maxbonespervertex is not equal to maxbonesperpartition (in that case bone indices cannot be unique and sorted). This options is required for Freedom Force vs. the 3rd Reich skin partitions.
- **triangles** – The triangles of the partition (if not specified, then this defaults to C{self.data.get_triangles()}.

- **trianglepartmap** – Maps each triangle to a partition index. Faces with different indices will never appear in the same partition. If the skin instance is a BSDismemberSkinInstance, then these indices are used as body part types, and the partitions in the BSDismemberSkinInstance are updated accordingly. Note that the faces are counted relative to L{triangles}.
- **maximize_bone_sharing** – Maximize bone sharing between partitions. This option is useful for Fallout 3.

update_tangent_space(*as_extra=None*, *vertexprecision=3*, *normalprecision=3*)

Recalculate tangent space data.

Parameters **as_extra** – Whether to store the tangent space data as extra data (as in Oblivion) or not (as in Fallout 3). If not set, switches to Oblivion if an extra data block is found, otherwise does default. Set it to override this detection (for example when using this function to create tangent space data) and force behaviour.

class NiTriBasedGeomData(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif._NiTriBasedGeomData`, `object`

get_triangle_indices(*triangles*)

Yield list of triangle indices (relative to self.get_triangles()) of given triangles. Degenerate triangles in the list are assigned index `None`.

```
>>> from pyffi.formats.nif import NifFormat
>>> geomdata = NifFormat.NiTriShapeData()
>>> geomdata.set_triangles([(0,1,2),(1,2,3),(2,3,4)])
>>> list(geomdata.get_triangle_indices([(1,2,3)]))
[1]
>>> list(geomdata.get_triangle_indices([(3,1,2)]))
[1]
>>> list(geomdata.get_triangle_indices([(2,3,1)]))
[1]
>>> list(geomdata.get_triangle_indices([(1,2,0),(4,2,3)]))
[0, 2]
>>> list(geomdata.get_triangle_indices([(0,0,0),(4,2,3)]))
[None, 2]
>>> list(geomdata.get_triangle_indices([(0,3,4),(4,2,3)]))
Traceback (most recent call last):
    ...
ValueError: ...
```

Parameters **triangles** (*iterator or list of tuples of three ints*) – An iterable of triangles to check.

is_interchangeable(*other*)

Heuristically checks if two NiTriBasedGeomData blocks describe the same geometry, that is, if they can be used interchangeably in a NIF file without affecting the rendering. The check is not fool proof but has shown to work in most practical cases.

Parameters **other** (L{NifFormat.NiTriBasedGeomData} (if it has another type then the function will always return `False`)) – Another geometry data block.

Returns `True` if the geometries are equivalent, `False` otherwise.

class NiTriShape(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif.NiTriBasedGeom`

A shape node that refers to singular triangle data.

class NiTriShapeData(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.formats.nif._NiTriShapeData`, `object`

Example usage:

```
>>> from pyffi.formats.nif import NifFormat
>>> block = NifFormat.NiTriShapeData()
>>> block.set_triangles([(0,1,2),(2,1,3),(2,3,4)])
>>> block.get_strips()
[[0, 1, 2, 3, 4]]
>>> block.get_triangles()
[(0, 1, 2), (2, 1, 3), (2, 3, 4)]
>>> block.set_strips([[1,0,1,2,3,4]])
>>> block.get_strips() # stripifier keeps geometry but nothing else
[[0, 2, 1, 3], [2, 4, 3]]
>>> block.get_triangles()
[(0, 2, 1), (1, 2, 3), (2, 4, 3)]
```

get_strips()

get_triangles()

set_strips(*strips*)

set_triangles(*triangles*, *stitchstrips=False*)

class NiTriShapeSkinController(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif.NiTimeController

Old version of skinning instance.

property bone_data

property bones

property num_bones

property vertex_counts

class NiTriStrips(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif.NiTriBasedGeom

A shape node that refers to data organized into strips of triangles

class NiTriStripsData(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.formats.nif._NiTriStripsData, object

Example usage:

```
>>> from pyffi.formats.nif import NifFormat
>>> block = NifFormat.NiTriStripsData()
>>> block.set_triangles([(0,1,2),(2,1,3),(2,3,4)])
>>> block.get_strips()
[[0, 1, 2, 3, 4]]
>>> block.get_triangles()
[(0, 1, 2), (1, 3, 2), (2, 3, 4)]
>>> block.set_strips([[1,0,1,2,3,4]])
>>> block.get_strips()
[[1, 0, 1, 2, 3, 4]]
>>> block.get_triangles()
[(0, 2, 1), (1, 2, 3), (2, 4, 3)]
```

get_strips()

get_triangles()

set_strips(*strips*)

set_triangles(*triangles*, *stitchstrips=False*)

class NiUVController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

Time controller for texture coordinates.

property data

property unknown_short

class NiUVData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Texture coordinate data.

property uv_groups

class NiVectorExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Extra vector data.

property unknown_float

property vector_data

class NiVertWeightsExtraData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiExtraData

Not used in skinning. Unsure of use - perhaps for morphing animation or gravity.

property num_bytes

property num_vertices

property weight

class NiVertexColorProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiProperty

Property of vertex colors. This object is referred to by the root object of the NIF file whenever some NiTriShapeData object has vertex colors with non-default settings; if not present, vertex colors have vertex_mode=2 and lighting_mode=1.

property flags

property lighting_mode

property vertex_mode

class NiVisController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiBoolInterpController

Time controller for visibility.

property data

class NiVisData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Visibility data for a controller.

property keys

property num_keys

class NiWireframeProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiProperty

Unknown.

property flags

class NiZBufferProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiProperty

This Property controls the Z buffer (OpenGL: depth buffer).

property flags

property function

exception NifError
Bases: Exception

Standard nif exception class.

class NodeGroup(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A group of NiNodes references.

property nodes

property num_nodes

class OblivionHavokMaterial(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

A material, used by havok shape objects in Oblivion.

HAV_MAT_CHAIN = 13

HAV_MAT_CHAIN_STAIRS = 28

HAV_MAT_CLOTH = 1

HAV_MAT_CLOTH_STAIRS = 16

HAV_MAT_DIRT = 2

HAV_MAT_DIRT_STAIRS = 17

HAV_MAT_ELEVATOR = 30

HAV_MAT_GLASS = 3

HAV_MAT_GLASS_STAIRS = 18

HAV_MAT_GRASS = 4

HAV_MAT_GRASS_STAIRS = 19

HAV_MAT_HEAVY_METAL = 11

HAV_MAT_HEAVY_METAL_STAIRS = 26

HAV_MAT_HEAVY_STONE = 10

HAV_MAT_HEAVY_STONE_STAIRS = 25

HAV_MAT_HEAVY_WOOD = 12

HAV_MAT_HEAVY_WOOD_STAIRS = 27

HAV_MAT_METAL = 5

HAV_MAT_METAL_STAIRS = 20

HAV_MAT_ORGANIC = 6

HAV_MAT_ORGANIC_STAIRS = 21

HAV_MAT_RUBBER = 31

HAV_MAT_SKIN = 7

HAV_MAT_SKIN_STAIRS = 22

HAV_MAT_SNOW = 14

HAV_MAT_SNOW_STAIRS = 29

HAV_MAT_STONE = 0

HAV_MAT_STONE_STAIRS = 15

HAV_MAT_WATER = 8

HAV_MAT_WATER_STAIRS = 23

HAV_MAT_WOOD = 9

HAV_MAT_WOOD_STAIRS = 24

class OblivionLayer(***kwargs*)

Bases: `pyffi.object_models.xml.enum.EnumBase`

Sets mesh color in Oblivion Construction Set. Anything higher than 57 is also null.

ANIM_STATIC = 2

AVOID_BOX = 21

BACK_WEAPON = 53

BACK_WEAPON2 = 54

BIPED = 8

BODY = 34

CAMERA_PICK = 24

CHAR_CONTROLLER = 20

CLOUD_TRAP = 16

CLUTTER = 4

CUSTOM_PICK_1 = 28

CUSTOM_PICK_2 = 29

DROPPING_PICK = 31

GROUND = 17

HEAD = 33

ITEM_PICK = 25

LINE_OF_SIGHT = 26

L_CALF = 41

L_FOOT = 42

L_FOREARM = 38

```
L_HAND = 39
L_THIGH = 40
L_UPPER_ARM = 37
NONCOLLIDABLE = 15
NULL = 57
OTHER = 32
PATH_PICK = 27
PONYTAIL = 55
PORTAL = 18
PROJECTILE = 6
PROPS = 10
QUIVER = 52
R_CALF = 47
R_FOOT = 48
R_FOREARM = 44
R_HAND = 45
R_THIGH = 46
R_UPPER_ARM = 43
SHIELD = 51
SIDE_WEAPON = 50
SPELL = 7
SPELL_EXPLOSION = 30
SPINE1 = 35
SPINE2 = 36
STAIRS = 19
STATIC = 1
TAIL = 49
TERRAIN = 13
TRANSPARENT = 3
TRAP = 14
TREES = 9
TRIGGER = 12
UNIDENTIFIED = 0
UNKNOWN1 = 22
UNKNOWN2 = 23
WATER = 11
```

WEAPON = 5

WING = 56

class OblivionSubShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Havok Information for packed TriStrip shapes.

property havok_col_filter

property material

property num_vertices

class OldSkinData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Used to store skin weights in NiTriShapeSkinController.

property unknown_vector

property vertex_index

property vertex_weight

class PSLoopBehavior(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

PS_LOOP_AGESCALE = 2

PS_LOOP_CLAMP_BIRTH = 0

PS_LOOP_CLAMP_DEATH = 1

PS_LOOP_LOOP = 3

PS_LOOP_REFLECT = 4

class Particle(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

particle array entry

property lifespan

property lifetime

property timestamp

property unknown_short

property unknown_vector

property velocity

property vertex_id

class ParticleDesc(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Particle Description.

property translation

property unknown_float_1

property unknown_float_2

property unknown_float_3

property unknown_floats_1

property unknown_int_1

class PixelFormat(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

Specifies the pixel format used by the NiPixelData object to store a texture.

PX_FMT_DXT1 = 4

PX_FMT_DXT5 = 5

PX_FMT_DXT5_ALT = 6

PX_FMT_PAL8 = 2

PX_FMT_RGB8 = 0

PX_FMT_RGBA8 = 1

class PixelLayout(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

An unsigned 32-bit integer, describing the color depth of a texture.

PIX_LAY_BUMPMAP = 4

PIX_LAY_COMPRESSED = 3

PIX_LAY_DEFAULT = 6

PIX_LAY_HIGH_COLOR_16 = 1

PIX_LAY_PALETTISED = 0

PIX_LAY_PALETTISED_4 = 5

PIX_LAY_TRUE_COLOR_32 = 2

class Polygon(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Two dimensional screen elements.

property num_triangles

property num_vertices

property triangle_offset

property vertex_offset

class PrismaticDescriptor(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

property friction

property max_distance

property min_distance

property pivot_a

property pivot_b

property plane_a

property plane_b

property rotation_a

property rotation_b

property rotation_matrix_a

property sliding_a

property sliding_b

property unknown_byte_1

class PropagationMode(***kwargs*)

Bases: pyffi.object_models.xml.enum.EnumBase

PROPAGATE_ALWAYS = 2

PROPAGATE_NEVER = 3

PROPAGATE_ON_FAILURE = 1

PROPAGATE_ON_SUCCESS = 0

class Ptr(***kwargs*)

Bases: pyffi.formats.nif.Ref

A weak reference to another block, used to point up the hierarchy tree. The reference is not returned by the L{get_refs} function to avoid infinite recursion.

get_hash(*data=None*)

Returns a hash value (an immutable object) that can be used to identify the object uniquely.

get_refs(*data=None*)

Return all references (excluding weak pointers) used by this object.

get_value()

Return object value.

replace_global_node(*oldbranch*, *newbranch*, *edge_filter=(True*, *True)*)

```
>>> from pyffi.formats.nif import NifFormat
>>> x = NifFormat.NiNode()
>>> y = NifFormat.NiNode()
>>> z = NifFormat.NiNode()
>>> x.add_child(y)
>>> x.children[0] is y
True
>>> x.children[0] is z
False
>>> x.replace_global_node(y, z)
>>> x.children[0] is y
False
>>> x.children[0] is z
True
>>> x.replace_global_node(z, None)
>>> x.children[0] is None
True
```

set_value(*value*)

Set object value.

class QTransform(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.object_models.xml.struct_.StructBase

property rotation

property scale

property translation

class QuatKey(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A special version of the key type used for quaternions. Never has tangents.

property tbc

property time

property value

class Quaternion(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A quaternion.

property w

property x

property y

property z

class QuaternionXYZW(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A quaternion as it appears in the havok objects.

property w

property x

property y

property z

RE_FILENAME = re.compile('^.*\\.(nif|kf|kfa|nifcache|jmi|texcache|pcpatch|nft|item|nif

class RagdollDescriptor(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._RagdollDescriptor, object

update_a_b(*transform*)
Update B pivot and axes from A using the given transform.

class Ref(***kwargs*)
Bases: pyffi.object_models.xml.basic.BasicBase

Reference to another block.

fix_links(*data*)
Fix block links.

get_detail_display()
Return an object that can be used to display the instance.

get_hash(*data=None*)
Returns a hash value (an immutable object) that can be used to identify the object uniquely.

get_links(*data=None*)
Return all links referred to in this object.

get_refs(*data=None*)
Return all references (excluding weak pointers) used by this object.

get_size(*data=None*)
Returns size of the object in bytes.

get_value()
Return object value.

read(*stream*, *data*)
Read object from file.

replace_global_node(*oldbranch*, *newbranch*, *edge_filter=(True, True)*)

```
>>> from pyffi.formats.nif import NifFormat
>>> x = NifFormat.NiNode()
>>> y = NifFormat.NiNode()
>>> z = NifFormat.NiNode()
>>> x.add_child(y)
>>> x.children[0] is y
True
>>> x.children[0] is z
False
>>> x.replace_global_node(y, z)
>>> x.children[0] is y
False
>>> x.children[0] is z
True
>>> x.replace_global_node(z, None)
>>> x.children[0] is None
True
```

set_value(*value*)
Set object value.

write(*stream*, *data*)
Write block reference.

class Region(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.object_models.xml.struct_.StructBase`

A range of indices, which make up a region (such as a submesh).

property num_indices

property start_index

class RootCollisionNode(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.formats.nif.NiNode`

Morrowind-specific node for collision mesh.

class SemanticData(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.object_models.xml.struct_.StructBase`

property index

property name

class ShaderTexDesc(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.object_models.xml.struct_.StructBase`

An extended texture description for shader textures.

property is_used

property map_index

property texture_data

class ShortString(***kwargs*)

Bases: pyffi.object_models.xml.basic.BasicBase

Another type for strings.

get_hash(*data=None*)

Returns a hash value (an immutable object) that can be used to identify the object uniquely.

get_size(*data=None*)

Returns size of the object in bytes.

get_value()

Return object value.

read(*stream*, *data*)

Read object from file.

set_value(*value*)

Set object value.

write(*stream*, *data*)

Write object to file.

class SizedString(***kwargs*)

Bases: pyffi.object_models.xml.basic.BasicBase, pyffi.object_models.editable.EditableLineEdit

Basic type for strings. The type starts with an unsigned int which describes the length of the string.

```
>>> from tempfile import TemporaryFile
>>> f = TemporaryFile()
>>> from pyffi.object_models import FileFormat
>>> data = FileFormat.Data()
>>> s = SizedString()
>>> if f.write('\x07\x00\x00\x00abcdefg'.encode("ascii")): pass # ignore␣
↪result for py3k
>>> if f.seek(0): pass # ignore result for py3k
>>> s.read(f, data)
>>> str(s)
'abcdefg'
>>> if f.seek(0): pass # ignore result for py3k
>>> s.set_value('Hi There')
>>> s.write(f, data)
>>> if f.seek(0): pass # ignore result for py3k
>>> m = SizedString()
>>> m.read(f, data)
>>> str(m)
'Hi There'
```

get_hash(*data=None*)

Return a hash value for this string.

Returns An immutable object that can be used as a hash.

get_size(*data=None*)

Return number of bytes this type occupies in a file.

Returns Number of bytes.

get_value()
Return the string.
Returns The stored string.

read(*stream*, *data*)
Read string from stream.
Parameters **stream** (*file*) – The stream to read from.

set_value(*value*)
Set string to C{value}.
Parameters **value** (*str*) – The value to assign.

write(*stream*, *data*)
Write string to stream.
Parameters **stream** (*file*) – The stream to write to.

class SkinData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._SkinData, object

get_transform()
Return scale, rotation, and translation into a single 4x4 matrix.

set_transform(*mat*)
Set rotation, transform, and velocity.

class SkinPartition(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._SkinPartition, object

get_mapped_triangles()
Get list of triangles of this partition (mapping into the geometry data vertex list).

get_triangles()
Get list of triangles of this partition.

class SkinShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Reference to shape and skin instance.

property shape

property skin_instance

class SkinShapeGroup(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Unknown.

property link_pairs

property num_link_pairs

class SkinTransform(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._SkinTransform, object

get_transform()
Return scale, rotation, and translation into a single 4x4 matrix.

set_transform(*mat*)
Set rotation, transform, and velocity.

class SkinWeight(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A weighted vertex.

property index

property weight

class SkyObjectType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

Sets what sky function this object fulfills in BSSkyShaderProperty or SkyShaderProperty.

BSSM_SKY = 2

BSSM_SKY_CLOUDS = 3

BSSM_SKY_MOON_STARS_MASK = 7

BSSM_SKY_STARS = 5

BSSM_SKY_SUNGLARE = 1

BSSM_SKY_TEXTURE = 0

class SkyShaderProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.BSShaderLightingProperty

Bethesda-specific node? Found in Fallout3

property file_name

property sky_object_type

class SkyrimHavokMaterial(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

A material, used by havok shape objects in Skyrim.

MAT_BARREL = 732141076

MAT_BOTTLE = 493553910

MAT_BROKEN_STONE = 131151687

MAT_CLOTH = 3839073443

MAT_DIRT = 3106094762

MAT_DRAGON = 2518321175

MAT_GLASS = 3739830338

MAT_GRASS = 1848600814

MAT_GRAVEL = 428587608

MAT_HEAVY_METAL = 2229413539

MAT_HEAVY_STONE = 1570821952

MAT_HEAVY_WOOD = 3070783559

MAT_ICE = 873356572

MAT_LIGHT_WOOD = 365420259

MAT_MATERIAL_ARMOR_HEAVY = 3708432437

MAT_MATERIAL_ARMOR_LIGHT = 3424720541

MAT_MATERIAL_ARROW = 3725505938

MAT_MATERIAL_AXE_1HAND = 1305674443

```
MAT_MATERIAL_BASKET = 790784366
MAT_MATERIAL_BLADE_1HAND = 1060167844
MAT_MATERIAL_BLADE_1HAND_SMALL = 2617944780
MAT_MATERIAL_BLADE_2HAND = 2022742644
MAT_MATERIAL_BLUNT_2HAND = 3969592277
MAT_MATERIAL_BONE = 3049421844
MAT_MATERIAL_BOOK = 1264672850
MAT_MATERIAL_BOTTLE_SMALL = 2025794648
MAT_MATERIAL_BOULDER_LARGE = 1885326971
MAT_MATERIAL_BOULDER_MEDIUM = 4283869410
MAT_MATERIAL_BOULDER_SMALL = 1550912982
MAT_MATERIAL_BOWS_STAVES = 1607128641
MAT_MATERIAL_CARPET = 1286705471
MAT_MATERIAL_CERAMIC_MEDIUM = 781661019
MAT_MATERIAL_CHAIN = 3074114406
MAT_MATERIAL_CHAIN_METAL = 438912228
MAT_MATERIAL_COIN = 3589100606
MAT_MATERIAL_SHIELD_HEAVY = 3702389584
MAT_MATERIAL_SHIELD_LIGHT = 3448167928
MAT_MATERIAL_SKIN_LARGE = 2965929619
MAT_MATERIAL_SKIN_SMALL = 2632367422
MAT_MATERIAL_STONE_AS_STAIRS = 1886078335
MAT_MATERIAL_WOOD_AS_STAIRS = 1803571212
MAT_MUD = 1486385281
MAT_ORGANIC = 2974920155
MAT_SAND = 2168343821
MAT_SKIN = 591247106
MAT_SNOW = 398949039
MAT_SOLID_METAL = 1288358971
MAT_STAIRS_BROKEN_STONE = 2892392795
MAT_STAIRS_SNOW = 1560365355
MAT_STAIRS_STONE = 899511101
MAT_STAIRS_WOOD = 1461712277
MAT_STONE = 3741512247
MAT_UNKNOWN_1028101969 = 1028101969
MAT_UNKNOWN_1440721808 = 1440721808
```

MAT_UNKNOWN_1574477864 = 1574477864

MAT_UNKNOWN_1591009235 = 1591009235

MAT_WATER = 1024582599

MAT_WOOD = 500811281

class SkyrimLayer(***kwargs*)

Bases: pyffi.object_models.xml.enum.EnumBase

Physical purpose of collision object? The setting affects object's havok behavior in game. Anything higher than 47 is also null.

ACOUSTIC_SPACE = 21

ACTORZONE = 22

ANIMSTATIC = 2

AVOIDBOX = 34

BIPED = 8

BIPED_NO_CC = 33

CAMERAPICK = 39

CAMERASHPERE = 36

CHARCONTROLLER = 30

CLOUD_TRAP = 16

CLUTTER = 4

COLLISIONBOX = 35

CONEPROJECTILE = 38

CUSTOMPICK1 = 43

CUSTOMPICK2 = 44

DEADBIP = 32

DEBRIS_LARGE = 20

DEBRIS_SMALL = 19

DOORDETECTION = 37

DROPPINGPICK = 46

GASTRAP = 24

GROUND = 17

INVISIBLE_WALL = 27

ITEMPICK = 40

LINEOFSIGHT = 41

NONCOLLIDABLE = 15

NULL = 47

PATHPICK = 42

PORTAL = 18

PROJECTILE = 6

PROJECTILEZONE = 23

PROPS = 10

SHELLCASING = 25

SPELL = 7

SPELLEXPLOSION = 45

STAIRHELPER = 31

STATIC = 1

TERRAIN = 13

TRANSPARENT = 3

TRANSPARENT_SMALL = 26

TRANSPARENT_SMALL_ANIM = 28

TRAP = 14

TREES = 9

TRIGGER = 12

UNIDENTIFIED = 0

WARD = 29

WATER = 11

WEAPON = 5

class SkyrimShaderPropertyFlags1 (*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.object_models.xml.bit_struct.BitStructBase`

Skyrim Shader Property Flags 1

property slsf_1_cast_shadows

property slsf_1_decal

property slsf_1_dynamic_decal

property slsf_1_environment_mapping

property slsf_1_external_emittance

property slsf_1_eye_environment_mapping

property slsf_1_face_gen_rgb_tint

property slsf_1_facegen_detail_map

property slsf_1_fire_refraction

property slsf_1_greyscale_to_palette_alpha

property slsf_1_greyscale_to_palette_color

property slsf_1_hair_soft_lighting

property slsf_1_landscape

property slsf_1_localmap_hide_secret

property slsf_1_model_space_normals

property slsf_1_multiple_textures

property slsf_1_non_projective_shadows

property slsf_1_own_emit

property slsf_1_parallax

property slsf_1_parallax_occlusion

property slsf_1_projected_uv

property slsf_1_recieve_shadows

property slsf_1_refraction

property slsf_1_remappable_textures

property slsf_1_screendoor_alpha_fade

property slsf_1_skinned

property slsf_1_soft_effect

property slsf_1_specular

property slsf_1_temp_refraction

property slsf_1_use_falloff

property slsf_1_vertex_alpha

property slsf_1_z_buffer_test

class SkyrimShaderPropertyFlags2 (*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.object_models.xml.bit_struct.BitStructBase`

Skyrim Shader Property Flags 2

property slsf_2_anisotropic_lighting

property slsf_2_assume_shadowmask

property slsf_2_back_lighting

property slsf_2_billboard

property slsf_2_cloud_lod

property slsf_2_double_sided

property slsf_2_effect_lighting

property slsf_2_env_map_light_fade

property slsf_2_fit_slope

property slsf_2_glow_map

property slsf_2_hd_lod_objects

property slsf_2_hide_on_local_map

property slsf_2_lod_landscape

property slsf_2_lod_objects

property slsf_2_multi_index_snow

property slsf_2_multi_layer_parallax

property slsf_2_no_fade

property slsf_2_no_lod_land_blend

property slsf_2_no_transparency_multisampling

property slsf_2_packed_tangent

property slsf_2_premult_alpha

property slsf_2_rim_lighting

property slsf_2_soft_lighting

property slsf_2_tree_anim

property slsf_2_uniform_scale

property slsf_2_unused_01

property slsf_2_unused_02

property slsf_2_vertex_colors

property slsf_2_vertex_lighting

property slsf_2_weapon_blood

property slsf_2_wireframe

property slsf_2_z_buffer_write

class SkyrimWaterShaderFlags(*template=None*, *argument=None*, *parent=None*)

Bases: `pyffi.object_models.xml.bit_struct.BitStructBase`

Skyrim water shader property flags

property swsf_1_bypass_refraction_map

property swsf_1_enabled

property swsf_1_highlight_layer_toggle

property swsf_1_unknown_0

property swsf_1_unknown_3

property swsf_1_unknown_4

property swsf_1_unknown_5

property swsf_1_water_toggle

class SolverDeactivation(***kwargs*)

Bases: `pyffi.object_models.xml.enum.EnumBase`

A list of possible solver deactivation settings. This value defines how the solver deactivates objects. The solver works on a per object basis. Note: Solver deactivation does not save CPU, but reduces creeping of movable objects in a pile quite dramatically.

SOLVER_DEACTIVATION_HIGH = 4

SOLVER_DEACTIVATION_INVALID = 0

SOLVER_DEACTIVATION_LOW = 2

SOLVER_DEACTIVATION_MAX = 5

SOLVER_DEACTIVATION_MEDIUM = 3

SOLVER_DEACTIVATION_OFF = 1

class SortingMode(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

SORTING_INHERIT = 0

SORTING_OFF = 1

class SphereBV(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A sphere.

property center

property radius

class StencilAction(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

This enum defines the various actions used in conjunction with the stencil buffer. For a detailed description of the individual options please refer to the OpenGL docs.

ACTION_DECREMENT = 4

ACTION_INCREMENT = 3

ACTION_INVERT = 5

ACTION_KEEP = 0

ACTION_REPLACE = 2

ACTION_ZERO = 1

class StencilCompareMode(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

This enum contains the options for doing stencil buffer tests.

TEST_ALWAYS = 7

TEST_EQUAL = 2

TEST_GREATER = 4

TEST_GREATER_EQUAL = 6

TEST_LESS = 1

TEST_LESS_EQUAL = 3

TEST_NEVER = 0

TEST_NOT_EQUAL = 5

class StiffSpringDescriptor(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

property length

property pivot_a

property pivot_b

StringIndex
alias of pyffi.object_models.common.UInt

class StringOffset(***kwargs*)
Bases: pyffi.object_models.common.Int

This is just an integer with -1 as default value.

class StringPalette(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._StringPalette, object

add_string(*text*)
Adds string to palette (will recycle existing strings if possible) and return offset to the string in the palette.

```
>>> from pyffi.formats.nif import NifFormat
>>> pal = NifFormat.StringPalette()
>>> pal.add_string("abc")
0
>>> pal.add_string("abc")
0
>>> pal.add_string("def")
4
>>> pal.add_string("")
-1
>>> print(pal.get_string(4).decode("ascii"))
def
```

clear()
Clear all strings in the palette.

```
>>> from pyffi.formats.nif import NifFormat
>>> pal = NifFormat.StringPalette()
>>> pal.add_string("abc")
0
>>> pal.add_string("def")
4
>>> # pal.palette.decode("ascii") needs lstrip magic for py3k
>>> print(repr(pal.palette.decode("ascii")).lstrip("u"))
'abc\x00def\x00'
>>> pal.clear()
>>> # pal.palette.decode("ascii") needs lstrip magic for py3k
>>> print(repr(pal.palette.decode("ascii")).lstrip("u"))
''
```

get_all_strings()
Return a list of all strings.

```
>>> from pyffi.formats.nif import NifFormat
>>> pal = NifFormat.StringPalette()
>>> pal.add_string("abc")
0
>>> pal.add_string("def")
4
>>> for x in pal.get_all_strings():
...     print(x.decode("ascii"))
abc
def
>>> # pal.palette.decode("ascii") needs lstrip magic for py3k
```

(continues on next page)

(continued from previous page)

```
>>> print(repr(pal.palette.decode("ascii")).lstrip("u"))
'abc\x00def\x00'
```

get_string(*offset*)

Return string at given offset.

```
>>> from pyffi.formats.nif import NifFormat
>>> pal = NifFormat.StringPalette()
>>> pal.add_string("abc")
0
>>> pal.add_string("def")
4
>>> print(pal.get_string(0).decode("ascii"))
abc
>>> print(pal.get_string(4).decode("ascii"))
def
>>> pal.get_string(5)
pyffi.nif.stringpalette:WARNING:StringPalette: no string starts at offset
↪5 (string is b'ef', preceeding character is b'd')
b'ef'
>>> pal.get_string(100)
Traceback (most recent call last):
    ...
ValueError: ...
```

class SubConstraint(*template=None*, *argument=None*, *parent=None*)

Bases: pyffi.object_models.xml.struct_.StructBase

property ball_and_socket

property entities

property hinge

property limited_hinge

property num_entities

property priority

property prismatic

property ragdoll

property stiff_spring

property type

class SymmetryType(***kwargs*)

Bases: pyffi.object_models.xml.enum.EnumBase

Determines symetry type used by NiPSysBombModifier.

CYLINDRICAL_SYMMETRY = 1

PLANAR_SYMMETRY = 2

SPHERICAL_SYMMETRY = 0

class SyncPoint(***kwargs*)

Bases: pyffi.object_models.xml.enum.EnumBase

Specifies the time when an application must syncronize for some reason.

SYNC_ANY = 32768

SYNC_PHYSICS_COMPLETED = 32864

SYNC_PHYSICS_SIMULATE = 32848

SYNC_POST_UPDATE = 32800

SYNC_REFLECTIONS = 32880

SYNC_RENDER = 32832

SYNC_UPDATE = 32784

SYNC_VISIBLE = 32816

class TBC(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Tension, bias, continuity.

property b

property c

property t

class TallGrassShaderProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.BSShaderProperty

Bethesda-specific node.

property file_name

class TargetColor(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

Used by NiPoint3InterpControllers to select which type of color in the controlled object that will be animated.

TC_AMBIENT = 0

TC_DIFFUSE = 1

TC_SELF_ILLUM = 3

TC_SPECULAR = 2

class TexClampMode(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

Specifies the availiable texture clamp modes. That is, the behavior of pixels outside the range of the texture.

CLAMP_S_CLAMP_T = 0

CLAMP_S_WRAP_T = 1

WRAP_S_CLAMP_T = 2

WRAP_S_WRAP_T = 3

class TexCoord(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._TexCoord, object

as_list()

normalize()

class TexDesc(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Texture description.

property center_offset

property clamp_mode

property filter_mode

property flags

property has_texture_transform

property ps_2_k

property ps_2_l

property source

property tiling

property transform_type

property translation

property unknown_1

property unknown_short

property uv_set

property w_rotation

class TexFilterMode(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

Specifies the availiable texture filter modes. That is, the way pixels within a texture are blended together when textures are displayed on the screen at a size other than their original dimentions.

FILTER_BILERP = 1

FILTER_BILERP_MIPNEAREST = 5

FILTER_NEAREST = 0

FILTER_NEAREST_MIPLERP = 4

FILTER_NEAREST_MIPNEAREST = 3

FILTER_TRILERP = 2

class TexSource(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A texture source.

property file_name

property pixel_data

property unknown_byte

property unknown_link

property use_external

class TexTransform(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

Determines how a NiTextureTransformController animates the UV coordinates.

TT_ROTATE = 2

TT_SCALE_U = 3

TT_SCALE_V = 4

TT_TRANSLATE_U = 0

TT_TRANSLATE_V = 1

class TexType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

The type of texture.

BASE_MAP = 0

BUMP_MAP = 5

DARK_MAP = 1

DECAL_0_MAP = 8

DECAL_1_MAP = 9

DECAL_2_MAP = 10

DECAL_3_MAP = 11

DETAIL_MAP = 2

GLOSS_MAP = 3

GLOW_MAP = 4

NORMAL_MAP = 6

UNKNOWN2_MAP = 7

class TileShaderProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.BSShaderLightingProperty

Bethesda-specific node.

property file_name

class Triangle(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

List of three vertex indices.

property v_1

property v_2

property v_3

class UnionBV(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

property bounding_volumes

property num_bv

class Vector3(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._Vector3, object

as_list()

as_tuple()

assign(*vec*)
Set this vector to values from another object that supports iteration or x,y,z properties

crossproduct(*x*)

get_copy()

norm(*sqrt=<built-in function sqrt>*)

normalize(*ignore_error=False*, *sqrt=<built-in function sqrt>*)

normalized(*ignore_error=False*)

class Vector4(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._Vector4, object

```
>>> from pyffi.formats.nif import NifFormat
>>> vec = NifFormat.Vector4()
>>> vec.x = 1.0
>>> vec.y = 2.0
>>> vec.z = 3.0
>>> vec.w = 4.0
>>> print(vec)
[  1.000  2.000  3.000  4.000 ]
>>> vec.as_list()
[1.0, 2.0, 3.0, 4.0]
>>> vec.as_tuple()
(1.0, 2.0, 3.0, 4.0)
>>> print(vec.get_vector_3())
[  1.000  2.000  3.000 ]
>>> vec2 = NifFormat.Vector4()
>>> vec == vec2
False
>>> vec2.x = 1.0
>>> vec2.y = 2.0
>>> vec2.z = 3.0
>>> vec2.w = 4.0
>>> vec == vec2
True
```

as_list()

as_tuple()

get_copy()

get_vector_3()

class VelocityType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

Controls the way the a particle mesh emitter determines the starting speed and direction of the particles that are emitted.

VELOCITY_USE_DIRECTION = 2

VELOCITY_USE_NORMALS = 0

VELOCITY_USE_RANDOM = 1

class VertMode(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

An unsigned 32-bit integer, which describes how to apply vertex colors.

VERT_MODE_SRC_AMB_DIF = 2

VERT_MODE_SRC_EMISSIVE = 1

VERT_MODE_SRC_IGNORE = 0

class VolumetricFogShaderProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.BSShaderProperty

Bethesda-specific node.

class WaterShaderProperty(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.BSShaderProperty

Bethesda-specific node? Found in Fallout3

class ZCompareMode(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

This enum contains the options for doing z buffer tests.

ZCOMP_ALWAYS = 0

ZCOMP_EQUAL = 2

ZCOMP_GREATER = 4

ZCOMP_GREATER_EQUAL = 6

ZCOMP_LESS = 1

ZCOMP_LESS_EQUAL = 3

ZCOMP_NEVER = 7

ZCOMP_NOT_EQUAL = 5

class bhkAabbPhantom(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkShapePhantom

Bethesda-specific node.

property unknown_ints_1

class bhkBallAndSocketConstraint(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkConstraint

A Ball and Socket Constraint.

property ball_and_socket

class bhkBallSocketConstraintChain(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkSerializable

A Ball and Socket Constraint chain.

property floats_1

property links

property links_2

property num_floats

property num_links

property num_links_2

property unknown_float_1

property unknown_float_2

property unknown_int_1

property unknown_int_2

property unknown_int_3

class bhkBlendCollisionObject(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkCollisionObject

Unknown.

property unknown_float_1

property unknown_float_2

class bhkBlendController(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiTimeController

Unknown. Is apparently only used in skeleton.nif files.

property unknown_int

class bhkBoxShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._bhkBoxShape, object

apply_scale(*scale*)
Apply scale factor C{scale} on data.

get_mass_center_inertia(*density=1*, *solid=True*)
Return mass, center, and inertia tensor.

class bhkBreakableConstraint(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkConstraint

A breakable constraint.

property remove_if_broken

property sub_constraint

property threshold

class bhkBvTreeShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkShape

A tree-like Havok data structure stored in an assembly-like binary code?

class bhkCMSDBigTris(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Triangle indices used in pair with "Big Verts" in a bhkCompressedMeshShapeData.

property triangle_1

property triangle_2

property triangle_3

property unknown_int_1

property unknown_short_1

class bhkCMSDChunk(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Defines subshape chunks in bhkCompressedMeshShapeData

property indices

property indices_2

property material_index

property num_indices

property num_indices_2

property num_strips

property num_vertices

property strips

property transform_index

property translation

property unknown_short_1

property vertices

class bhkCMSDMaterial(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Per-chunk material, used in bhkCompressedMeshShapeData

property byte_set_to_0

property layer

property material

property short_set_to_0

class bhkCMSDTransform(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A set of transformation data: translation and rotation

property rotation

property translation

class bhkCapsuleShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._bhkCapsuleShape, object

apply_scale(*scale*)
Apply scale factor <scale> on data.

get_mass_center_inertia(*density=1*, *solid=True*)
Return mass, center, and inertia tensor.

class bhkCollisionObject(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkNiCollisionObject

Havok related collision object?

class bhkCompressedMeshShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkShape

Compressed collision mesh.

property data

property radius

property scale

property target

property unknown_4_bytes

property unknown_float_1

property unknown_float_3

property unknown_float_4

property unknown_float_5

property unknown_floats_1

property unknown_int_1

class bhkCompressedMeshShapeData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkRefObject

A compressed mesh shape for collision in Skyrim.

property big_tris

property big_verts

property bits_per_index

property bits_per_w_index

property bounds_max

property bounds_min

property chunk_materials

property chunk_transforms

property chunks

property error

property mask_index

property mask_w_index

property num_big_tris

property num_big_verts

property num_chunks

property num_materials

property num_transforms

property unknown_byte_1

property unknown_byte_2

property unknown_int_12

property unknown_int_3

property unknown_int_4

property unknown_int_5

property unknown_int_6

class bhkConstraint(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._bhkConstraint, object

get_transform_a_b(*parent*)
Returns the transform of the first entity relative to the second entity. Root is simply a nif block that is a common parent to both blocks.

class bhkConvexListShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkShape

A havok shape. A list of convex shapes.

Do not put a bhkPackedNiTriStripsShape in the Sub Shapes. Use a separate collision nodes without a list shape for those.

Also, shapes collected in a bhkListShape may not have the correct walking noise, so only use it for non-walkable objects.

property material

property num_sub_shapes

property sub_shapes

property unknown_byte_1

property unknown_float_1

property unknown_floats

class bhkConvexShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkSphereRepShape

A havok shape.

class bhkConvexTransformShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkTransformShape

A convex transformed shape?

class bhkConvexVerticesShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._bhkConvexVerticesShape, object

apply_scale(*scale*)
Apply scale factor on data.

get_mass_center_inertia(*density=1*, *solid=True*)
Return mass, center, and inertia tensor.

class bhkEntity(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkWorldObject

A havok node, describes physical properties.

class bhkHingeConstraint(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkConstraint

A hinge constraint.

property hinge

class bhkLimitedHingeConstraint(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._bhkLimitedHingeConstraint, object

apply_scale(*scale*)
Scale data.

update_a_b(*parent*)
Update the B data from the A data. The parent argument is simply a common parent to the entities.

class bhkLiquidAction(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkSerializable

Bethesda-specific node.

property unknown_float_1

property unknown_float_2

property unknown_float_3

property unknown_float_4

property unknown_int_1

property unknown_int_2

property unknown_int_3

class bhkListShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._bhkListShape, object

add_shape(*shape*, *front=False*)
Add shape to list.

get_mass_center_inertia(*density=1*, *solid=True*)
Return center of gravity and area.

remove_shape(*shape*)
Remove a shape from the shape list.

class bhkMalleableConstraint(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._bhkMalleableConstraint, object

apply_scale(*scale*)
Scale data.

update_a_b(*parent*)
Update the B data from the A data.

class bhkMeshShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkShape

property num_strips_data

property num_unknown_floats

property strips_data

property unknown_1

property unknown_2

property unknown_floats

class bhkMoppBvTreeShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._bhkMoppBvTreeShape, object

get_mass_center_inertia(*density=1*, *solid=True*)
Return mass, center of gravity, and inertia tensor.

mopp_from_tree(*tree*)

parse_mopp(*start=0*, *depth=0*, *toffset=0*, *verbose=False*)
The mopp data is printed to the debug channel while parsed. Returns list of indices into mopp data of the bytes processed and a list of triangle indices encountered.

The verbose argument is ignored (and is deprecated).

split_triangles(*ts*, *bbox*, *dir=0*)
Direction 0=X, 1=Y, 2=Z

update_mopp()
Update the MOPP data, scale, and origin, and welding info.

@deprecated: use update_mopp_welding instead

update_mopp_welding()
Update the MOPP data, scale, and origin, and welding info.

update_origin_scale()
Update scale and origin.

class bhkMultiSphereShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._bhkMultiSphereShape, object

get_mass_center_inertia(*density=1*, *solid=True*)
Return center of gravity and area.

class bhkNiCollisionObject(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiCollisionObject

Havok related collision object?

property body

property flags

class bhkNiTriStripsShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._bhkNiTriStripsShape, object

get_interchangeable_packed_shape()
Returns a bhkPackedNiTriStripsShape block that is geometrically interchangeable.

get_mass_center_inertia(*density=1*, *solid=True*)
Return mass, center, and inertia tensor.

class bhkOrientHingedBodyAction(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkSerializable

Bethesda-Specific node.

property unknown_ints_1

class bhkPCollisionObject(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkNiCollisionObject

Unknown.

class bhkPackedNiTriStripsShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._bhkPackedNiTriStripsShape, object

add_shape(*triangles*, *normals*, *vertices*, *layer=0*, *material=0*)
Pack the given geometry.

get_mass_center_inertia(*density=1*, *solid=True*)
Return mass, center, and inertia tensor.

get_sub_shapes()
Return sub shapes (works for both Oblivion and Fallout 3).

get_triangle_hash_generator()
Generator which produces a tuple of integers, or None in degenerate case, for each triangle to ease detection of duplicate triangles.

```
>>> shape = NifFormat.bhkPackedNiTriStripsShape()
>>> data = NifFormat.hkPackedNiTriStripsData()
>>> shape.data = data
>>> data.num_triangles = 6
>>> data.triangles.update_size()
>>> data.triangles[0].triangle.v_1 = 0
>>> data.triangles[0].triangle.v_2 = 1
>>> data.triangles[0].triangle.v_3 = 2
>>> data.triangles[1].triangle.v_1 = 2
>>> data.triangles[1].triangle.v_2 = 1
>>> data.triangles[1].triangle.v_3 = 3
>>> data.triangles[2].triangle.v_1 = 3
>>> data.triangles[2].triangle.v_2 = 2
>>> data.triangles[2].triangle.v_3 = 1
>>> data.triangles[3].triangle.v_1 = 3
>>> data.triangles[3].triangle.v_2 = 1
>>> data.triangles[3].triangle.v_3 = 2
>>> data.triangles[4].triangle.v_1 = 0
>>> data.triangles[4].triangle.v_2 = 0
>>> data.triangles[4].triangle.v_3 = 3
>>> data.triangles[5].triangle.v_1 = 1
>>> data.triangles[5].triangle.v_2 = 3
>>> data.triangles[5].triangle.v_3 = 4
>>> list(shape.get_triangle_hash_generator())
[(0, 1, 2), (1, 3, 2), (1, 3, 2), (1, 2, 3), None, (1, 3, 4)]
```

Returns A generator yielding a hash value for each triangle.

get_vertex_hash_generator(*vertexprecision=3*, *subshape_index=None*)
Generator which produces a tuple of integers for each vertex to ease detection of duplicate/close enough to remove vertices. The precision parameter denote number of significant digits behind the comma.

For vertexprecision, 3 seems usually enough (maybe we'll have to increase this at some point).

```
>>> shape = NifFormat.bhkPackedNiTriStripsShape()
>>> data = NifFormat.hkPackedNiTriStripsData()
>>> shape.data = data
>>> shape.num_sub_shapes = 2
>>> shape.sub_shapes.update_size()
>>> data.num_vertices = 3
>>> shape.sub_shapes[0].num_vertices = 2
>>> shape.sub_shapes[1].num_vertices = 1
>>> data.vertices.update_size()
>>> data.vertices[0].x = 0.0
>>> data.vertices[0].y = 0.1
```

(continues on next page)

(continued from previous page)

```
>>> data.vertices[0].z = 0.2
>>> data.vertices[1].x = 1.0
>>> data.vertices[1].y = 1.1
>>> data.vertices[1].z = 1.2
>>> data.vertices[2].x = 2.0
>>> data.vertices[2].y = 2.1
>>> data.vertices[2].z = 2.2
>>> list(shape.get_vertex_hash_generator())
[(0, (0, 100, 200)), (0, (1000, 1100, 1200)), (1, (2000, 2100, 2200))]
>>> list(shape.get_vertex_hash_generator(subshape_index=0))
[(0, 100, 200), (1000, 1100, 1200)]
>>> list(shape.get_vertex_hash_generator(subshape_index=1))
[(2000, 2100, 2200)]
```

Parameters **vertexprecision** (*float*) – Precision to be used for vertices.
Returns A generator yielding a hash value for each vertex.

class bhkPhantom(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkWorldObject

Havok object that do not react with other objects when they collide (causing deflection, etc.) but still trigger collision notifications to the game. Possible uses are traps, portals, AI fields, etc.

class bhkPrismaticConstraint(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkConstraint

A prismatic constraint.

property prismatic

class bhkRDTConstraint(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

property entity_a

property entity_b

property malleable_constraint

property priority

property ragdoll

property type

property unknown_int

class bhkRDTMalleableConstraint(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A malleable constraint.

property damping

property entity_a

property entity_b

property hinge

property limited_hinge

property priority

property ragdoll

property type

property unknown_int

class bhkRagdollConstraint(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._bhkRagdollConstraint, object

apply_scale(*scale*)
Scale data.

update_a_b(*parent*)
Update the B data from the A data.

class bhkRagdollTemplate(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Found in Fallout 3, more ragdoll info? (meshesragdollconstraint*.rdt)

property bones

property name

property num_bones

class bhkRagdollTemplateData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.NiObject

Data for bhkRagdollTemplate

property constraint

property flag_or_num_constraints

property friction

property mass

property name

property radius

property restitution

property unknown_int

class bhkRefObject(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._bhkRefObject, object

get_shape_mass_center_inertia(*density=1*, *solid=True*)
Return mass, center of gravity, and inertia tensor of this object's shape, if self.shape is not None.

If self.shape is None, then returns zeros for everything.

class bhkRigidBody(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._bhkRigidBody, object

apply_scale(*scale*)
Apply scale factor <scale> on data.

update_mass_center_inertia(*density=1*, *solid=True*, *mass=None*)
Look at all the objects under this rigid body and update the mass, center of gravity, and inertia tensor accordingly. If the C{mass} parameter is given then the C{density} argument is ignored.

class bhkRigidBodyT(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkRigidBody

Unknown.

class bhkSPCollisionObject(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkPCollisionObject

Unknown.

class bhkSerializable(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkRefObject

Havok objects that can be saved and loaded from disk?

class bhkShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkSerializable

A Havok Shape?

class bhkShapeCollection(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkShape

Havok collision object that uses multiple shapes?

class bhkShapePhantom(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkPhantom

A Havok phantom that uses a Havok shape object for its collision volume instead of just a bounding box.

class bhkSimpleShapePhantom(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkShapePhantom

Unknown shape.

property unknown_float

property unknown_floats_2

property unkown_floats

class bhkSphereRepShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkShape

A havok shape, perhaps with a bounding sphere for quick rejection in addition to more detailed shape data?

property material

property radius

class bhkSphereShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._bhkSphereShape, object

apply_scale(*scale*)
Apply scale factor <scale> on data.

get_mass_center_inertia(*density=1*, *solid=True*)
Return mass, center, and inertia tensor.

class bhkStiffSpringConstraint(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkConstraint

A spring constraint.

property stiff_spring

class bhkTransformShape(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._bhkTransformShape, object

apply_scale(*scale*)
Apply scale factor <scale> on data.

get_mass_center_inertia(*density=1*, *solid=True*)
Return mass, center, and inertia tensor.

class bhkWorldObject(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif.bhkSerializable

Havok objects that have a position in the world?

property havok_col_filter

property shape

class bool(***kwargs*)
Bases: pyffi.object_models.xml.basic.BasicBase, pyffi.object_models.editable.EditableBoolComboBox

Basic implementation of a 32-bit (8-bit for versions > 4.0.0.2) boolean type.

```
>>> i = NifFormat.bool()
>>> i.set_value('false')
>>> i.get_value()
False
>>> i.set_value('true')
>>> i.get_value()
True
```

get_hash(*data=None*)
Returns a hash value (an immutable object) that can be used to identify the object uniquely.

get_size(*data=None*)
Returns size of the object in bytes.

get_value()
Return object value.

read(*stream*, *data*)
Read object from file.

set_value(*value*)
Set object value.

write(*stream*, *data*)
Write object to file.

byte
alias of pyffi.object_models.common.UByte

char
alias of pyffi.object_models.common.Char

float
alias of pyffi.object_models.common.Float

games = {'?': [167772419], 'Atlantica': [335675400], 'Axis and Allies': [167837696]

class hkConstraintType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

The type of constraint.

BallAndSocket = 0

Hinge = 1

Prismatic = 6

Ragdoll = 7

StiffSpring = 8

class hkPackedNiTriStripsData(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.nif._hkPackedNiTriStripsData, object

apply_scale(*scale*)
Apply scale factor on data.

class hkResponseType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

RESPONSE_INVALID = 0

RESPONSE_NONE = 3

RESPONSE_REPORTING = 2

RESPONSE_SIMPLE_CONTACT = 1

class hkTriangle(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A triangle with extra data used for physics.

property normal

property triangle

property welding_info

int
alias of pyffi.object_models.common.Int

class physXMaterialRef(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

property material_desc

property number

property unknown_byte_1

short
alias of pyffi.object_models.common.Short

class string(***kwargs*)
Bases: pyffi.object_models.common.SizedString

get_hash(*data=None*)
Return a hash value for this string.
Returns An immutable object that can be used as a hash.

get_size(*data=None*)
Return number of bytes this type occupies in a file.
Returns Number of bytes.

get_strings(*data*)
Return all strings used by this object.

read(*stream*, *data*)
Read string from stream.
Parameters **stream** (*file*) – The stream to read from.

write(*stream*, *data*)
Write string to stream.
Parameters **stream** (*file*) – The stream to write to.

uint
alias of pyffi.object_models.common.UInt

ulittle32
alias of pyffi.object_models.common.ULittle32

ushort
alias of pyffi.object_models.common.UShort

static version_number(*version_str*)
Converts version string into an integer.

Parameters **version_str** (*str*) – The version string.

Returns A version integer.

```
>>> hex(NifFormat.version_number('3.14.15.29'))
'0x30e0f1d'
>>> hex(NifFormat.version_number('1.2'))
'0x1020000'
>>> hex(NifFormat.version_number('3.03'))
'0x3000300'
>>> hex(NifFormat.version_number('NS'))
'0xa010000'
```

versions = {'10.0.1.0': 167772416, '10.0.1.2': 167772418, '10.0.1.3': 167772419, '1

xml_alias = []

xml_bit_struct = [<bit_struct 'BSSegmentFlags'>, <bit_struct 'FurnitureEntryPoints'>,

xml_enum = [<enum 'AlphaFormat'>, <enum 'ApplyMode'>, <enum 'TexType'>, <enum 'KeyType

xml_file_name = 'nif.xml'

xml_file_path = [None, '/home/docs/checkouts/readthedocs.org/user_builds/pyffi/checkou

xml_struct = [<struct 'Color3'>, <struct 'ByteColor3'>, <struct 'Color4'>, <struct 'By

Regression tests

These tests are used to check for functionality and bugs in the library. They also provide code examples which you may find useful.

Read a NIF file

```
>>> from os.path import dirname
>>> dirpath = __file__
>>> for i in range(4): #recurse up to root repo dir
...     dirpath = dirname(dirpath)
>>> repo_root = dirpath
>>> format_root = os.path.join(repo_root, 'tests', 'spells', 'nif', 'files')
>>> stream = open(os.path.join(format_root, 'test.nif'), 'rb')
>>> data = NifFormat.Data()
```

(continues on next page)

(continued from previous page)

```
>>> # inspect is optional; it will not read the actual blocks
>>> data.inspect(stream)
>>> hex(data.version)
'0x14010003'
>>> data.user_version
0
>>> for blocktype in data.header.block_types:
...     print(blocktype.decode("ascii"))
NiNode
NiTriShape
NiTriShapeData
>>> data.roots # blocks have not been read yet, so this is an empty list
[]
>>> data.read(stream)
>>> for root in data.roots:
...     for block in root.tree():
...         if isinstance(block, NifFormat.NiNode):
...             print(block.name.decode("ascii"))
test
>>> stream.close()
```

Parse all NIF files in a directory tree

```
>>> for stream, data in NifFormat.walkData(format_root):
...     try:
...         # the replace call makes the doctest also pass on windows
...         os_path = stream.name
...         split = (os_path.split(os.sep))[-5:]
...         rejoin = os.path.join(*split).replace(os.sep, "/")
...         print("reading %s" % rejoin)
...         data.read(stream)
...     except Exception:
...         print(
...             "Warning: read failed due corrupt file,"
...             " corrupt format description, or bug.")
reading tests/spells/nif/files/invalid.nif
Warning: read failed due corrupt file, corrupt format description, or bug.
reading tests/spells/nif/files/nds.nif
reading tests/spells/nif/files/neosteam.nif
reading tests/spells/nif/files/test.nif
reading tests/spells/nif/files/test_centerradius.nif
reading tests/spells/nif/files/test_check_tangentspace1.nif
reading tests/spells/nif/files/test_check_tangentspace2.nif
reading tests/spells/nif/files/test_check_tangentspace3.nif
reading tests/spells/nif/files/test_check_tangentspace4.nif
reading tests/spells/nif/files/test_convexverticesshape.nif
reading tests/spells/nif/files/test_dump_tex.nif
reading tests/spells/nif/files/test_fix_clampmaterialalpha.nif
reading tests/spells/nif/files/test_fix_cleanstringpalette.nif
reading tests/spells/nif/files/test_fix_detachhavoktristripsdata.nif
reading tests/spells/nif/files/test_fix_disableparallax.nif
reading tests/spells/nif/files/test_fix_ffvt3rskinpartition.nif
reading tests/spells/nif/files/test_fix_mergeskeletonroots.nif
reading tests/spells/nif/files/test_fix_tangentspace.nif
```

(continues on next page)

(continued from previous page)

```
reading tests/spells/nif/files/test_fix_texturepath.nif
reading tests/spells/nif/files/test_grid_128x128.nif
reading tests/spells/nif/files/test_grid_64x64.nif
reading tests/spells/nif/files/test_mopp.nif
reading tests/spells/nif/files/test_opt_collision_complex_mopp.nif
reading tests/spells/nif/files/test_opt_collision_mopp.nif
reading tests/spells/nif/files/test_opt_collision_packed.nif
reading tests/spells/nif/files/test_opt_collision_to_boxshape.nif
reading tests/spells/nif/files/test_opt_collision_to_boxshape_notabox.nif
reading tests/spells/nif/files/test_opt_collision_unpacked.nif
reading tests/spells/nif/files/test_opt_delunusedbones.nif
```

reading tests/spells/nif/files/test_opt_dupverts.nif reading tests/spells/nif/files/test_opt_emptyproperties.nif reading tests/spells/nif/files/test_opt_grid_layout.nif reading tests/spells/nif/files/test_opt_mergeduplicates.nif reading tests/spells/nif/files/test_opt_vertex_cache.nif reading tests/spells/nif/files/test_opt_zeroscale.nif reading tests/spells/nif/files/test_skincenterradius.nif reading tests/spells/nif/files/test_vertexcolor.nif

Create a NIF model from scratch and write to file

```
>>> root = NifFormat.NiNode()
>>> root.name = 'Scene Root'
>>> blk = NifFormat.NiNode()
>>> root.add_child(blk)
>>> blk.name = 'new block'
>>> blk.scale = 2.4
>>> blk.translation.x = 3.9
>>> blk.rotation.m_11 = 1.0
>>> blk.rotation.m_22 = 1.0
>>> blk.rotation.m_33 = 1.0
>>> ctrl = NifFormat.NiVisController()
>>> ctrl.flags = 0x000c
>>> ctrl.target = blk
>>> blk.add_controller(ctrl)
>>> blk.add_controller(NifFormat.NiAlphaController())
>>> strips = NifFormat.NiTriStrips()
>>> root.add_child(strips, front = True)
>>> strips.name = "hello world"
>>> strips.rotation.m_11 = 1.0
>>> strips.rotation.m_22 = 1.0
>>> strips.rotation.m_33 = 1.0
>>> data = NifFormat.NiTriStripsData()
>>> strips.data = data
>>> data.num_vertices = 5
>>> data.has_vertices = True
>>> data.vertices.update_size()
>>> for i, v in enumerate(data.vertices):
...     v.x = 1.0+i/10.0
...     v.y = 0.2+1.0/(i+1)
...     v.z = 0.03
>>> data.update_center_radius()
>>> data.num_strips = 2
>>> data.strip_lengths.update_size()
>>> data.strip_lengths[0] = 3
>>> data.strip_lengths[1] = 4
>>> data.has_points = True
```

(continues on next page)

(continued from previous page)

```
>>> data.points.update_size()
>>> data.points[0][0] = 0
>>> data.points[0][1] = 1
>>> data.points[0][2] = 2
>>> data.points[1][0] = 1
>>> data.points[1][1] = 2
>>> data.points[1][2] = 3
>>> data.points[1][3] = 4
>>> data.num_uv_sets = 1
>>> data.has_uv = True
>>> data.uv_sets.update_size()
>>> for i, v in enumerate(data.uv_sets[0]):
...     v.u = 1.0-i/10.0
...     v.v = 1.0/(i+1)
>>> data.has_normals = True
>>> data.normals.update_size()
>>> for i, v in enumerate(data.normals):
...     v.x = 0.0
...     v.y = 0.0
...     v.z = 1.0
>>> strips.update_tangent_space()
>>> from tempfile import TemporaryFile
>>> stream = TemporaryFile()
>>> nifdata = NifFormat.Data(version=0x14010003, user_version=10)
>>> nifdata.roots = [root]
>>> nifdata.write(stream)
>>> stream.close()
```

Get list of versions and games

```
>>> for vnum in sorted(NifFormat.versions.values()):
...     print('0x%08X' % vnum)
0x02030000
0x03000000
0x03000300
0x03010000
0x0303000D
0x04000000
0x04000002
0x0401000C
0x04020002
0x04020100
0x04020200
0x0A000100
0x0A000102
0x0A000103
0x0A010000
0x0A010065
0x0A01006A
0x0A020000
0x0A020001
0x0A040001
0x14000004
0x14000005
```

(continues on next page)

(continued from previous page)

```
0x14010003
0x14020007
0x14020008
0x14030001
0x14030002
0x14030003
0x14030006
0x14030009
0x14050000
0x14060000
0x14060500
0x1E000002
0x1E010003
>>> for game, versions in sorted(NifFormat.games.items(), key=lambda x: x[0]):
...     print("%s " % game + " ".join('0x%08X' % vnum for vnum in versions))
? 0x0A000103
Atlantica 0x14020008
Axis and Allies 0x0A010000
Bully SE 0x14030009
Civilization IV 0x04020002 0x04020100 0x04020200 0x0A000100 0x0A010000 0x0A020000␣
↪0x14000004
Culpa Innata 0x04020200
Dark Age of Camelot 0x02030000 0x03000300 0x03010000 0x0401000C 0x04020100 0x04020200␣
↪0x0A010000
Divinity 2 0x14030009
Emerge 0x14020007 0x14020008 0x14030001 0x14030002 0x14030003 0x14030006 0x1E000002
Empire Earth II 0x04020200 0x0A010000
Empire Earth III 0x14020007 0x14020008
Entropia Universe 0x0A010000
Epic Mickey 0x14060500
Fallout 3 0x14020007
Freedom Force 0x04000000 0x04000002
Freedom Force vs. the 3rd Reich 0x0A010000
Howling Sword 0x14030009
Kohan 2 0x0A010000
KrazyRain 0x14050000 0x14060000
Lazeska 0x14030009
Loki 0x0A020000
Megami Tensei: Imagine 0x14010003
Morrowind 0x04000002
NeoSteam 0x0A010000
Oblivion 0x0303000D 0x0A000100 0x0A000102 0x0A010065 0x0A01006A 0x0A020000 0x14000004␣
↪0x14000005
Prison Tycoon 0x0A020000
Pro Cycling Manager 0x0A020000
Red Ocean 0x0A020000
Rocksmith 0x1E010003
Rocksmith 2014 0x1E010003
Sid Meier's Railroads 0x14000004
Skyrim 0x14020007
Star Trek: Bridge Commander 0x03000000 0x03010000
The Guild 2 0x0A010000
Warhammer 0x14030009
Wildlife Park 2 0x0A010000 0x0A020000
Worldshift 0x0A020001 0x0A040001
Zoo Tycoon 2 0x0A000100
```

Reading an unsupported NIF file

```
>>> file = os.path.join(format_root, 'invalid.nif')
>>> stream = open(file, 'rb')
>>> data = NifFormat.Data()
>>> data.inspect(stream) # the file seems ok on inspection
>>> data.read(stream)
Traceback (most recent call last):
    ...
ValueError: ...
>>> stream.close()
```

Template types

```
>>> block = NifFormat.NiTextKeyExtraData()
>>> block.num_text_keys = 1
>>> block.text_keys.update_size()
>>> block.text_keys[0].time = 1.0
>>> block.text_keys[0].value = 'hi'
```

Links

```
>>> NifFormat.NiNode._has_links
True
>>> NifFormat.NiBone._has_links
True
>>> skelroot = NifFormat.NiNode()
>>> geom = NifFormat.NiTriShape()
>>> geom.skin_instance = NifFormat.NiSkinInstance()
>>> geom.skin_instance.skeleton_root = skelroot
>>> [block.__class__.__name__ for block in geom.get_refs()]
['NiSkinInstance']
>>> [block.__class__.__name__ for block in geom.get_links()]
['NiSkinInstance']
>>> [block.__class__.__name__ for block in geom.skin_instance.get_refs()]
[]
>>> [block.__class__.__name__ for block in geom.skin_instance.get_links()]
['NiNode']
```

Strings

```
>>> extra = NifFormat.NiTextKeyExtraData()
>>> extra.num_text_keys = 2
>>> extra.text_keys.update_size()
>>> extra.text_keys[0].time = 0.0
>>> extra.text_keys[0].value = "start"
>>> extra.text_keys[1].time = 2.0
>>> extra.text_keys[1].value = "end"
>>> for extrastr in extra.get_strings(None):
...     print(extrastr.decode("ascii"))
```

(continues on next page)

(continued from previous page)

```
start
end
```

pyffi.formats.tga — Targa (.tga)

Implementation

class pyffi.formats.tga.**TgaFormat**
Bases: pyffi.object_models.xml.FileFormat

This class implements the TGA format.

class ColorMapType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

An unsigned 8-bit integer, describing the color map type.

class Data
Bases: pyffi.object_models.Data

get_global_child_nodes(*edge_filter=(True, True)*)
Generator which yields all children of this item in the global view, of given edge type (default is edges of type 0).

Override this method.
Returns Generator for global node children.

inspect(*stream*)
Quick heuristic check if stream contains Targa data, by looking at the first 18 bytes.
Parameters **stream** (*file*) – The stream to inspect.

read(*stream*)
Read a tga file.
Parameters **stream** (file) – The stream from which to read.

write(*stream*)
Write a tga file.
Parameters **stream** (file) – The stream to write to.

class FooterString(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.basic.BasicBase

The Targa footer signature.

get_hash(*data=None*)
Return a hash value for the signature.
Returns An immutable object that can be used as a hash.

get_size(*data=None*)
Return number of bytes that the signature occupies in a file.
Returns Number of bytes.

get_value()
Get signature.
Returns The signature.

read(*stream*, *data*)
Read signature from stream.

Parameters **stream** (*file*) – The stream to read from.

set_value(*value*)
Set signature.
Parameters **value** (*str*) – The value to assign.

write(*stream*, *data*)
Write signature to stream.
Parameters **stream** (*file*) – The stream to read from.

class Image
Bases: pyffi.utils.graph.GlobalNode

get_detail_child_names(*edge_filter=(True*, *True)*)
Generator which yields all child names of this item in the detail view.

Override this method if the node has children.
Returns Generator for detail tree child names.
Return type generator yielding strs

get_detail_child_nodes(*edge_filter=(True*, *True)*)
Generator which yields all children of this item in the detail view (by default, all acyclic and active ones).

Override this method if the node has children.
Parameters **edge_filter** (EdgeFilter or type(None)) – The edge type to include.
Returns Generator for detail tree child nodes.
Return type generator yielding DetailNodes

class ImageType(***kwargs*)
Bases: pyffi.object_models.xml.enum.EnumBase

An unsigned 8-bit integer, describing the image type.

PixelData
alias of pyffi.object_models.common.UndecodedData

byte
alias of pyffi.object_models.common.Byte

char
alias of pyffi.object_models.common.Char

float
alias of pyffi.object_models.common.Float

int
alias of pyffi.object_models.common.Int

short
alias of pyffi.object_models.common.Short

ubyte
alias of pyffi.object_models.common.UByte

uint
alias of pyffi.object_models.common.UInt

ushort
alias of pyffi.object_models.common.UShort

Regression tests

Read a TGA file

```
>>> # check and read tga file
>>> import os
>>> from os.path import dirname
>>> dirpath = __file__
>>> for i in range(4): #recurse up to root repo dir
...     dirpath = dirname(dirpath)
>>> repo_root = dirpath
>>> format_root = os.path.join(repo_root, 'tests', 'formats', 'tga')
>>> file = os.path.join(format_root, 'test.tga').replace("\\", "/")
>>> stream = open(file, 'rb')
>>> data = TgaFormat.Data()
>>> data.inspect(stream)
>>> data.read(stream)
>>> stream.close()
>>> data.header.width
60
>>> data.header.height
20
```

Parse all TGA files in a directory tree

```
>>> for stream, data in TgaFormat.walkData(format_root):
...     try:
...         # the replace call makes the doctest also pass on windows
...         os_path = stream.name
...         split = (os_path.split(os.sep))[-4:]
...         rejoin = os.path.join(*split).replace("\\", "/")
...         print("reading %s" % rejoin)
...     except Exception:
...         print(
...             "Warning: read failed due corrupt file,"
...             " corrupt format description, or bug.")
reading tests/formats/tga/test.tga
reading tests/formats/tga/test_footer.tga
```

Create a TGA file from scratch and write to file

```
>>> data = TgaFormat.Data()
>>> from tempfile import TemporaryFile
>>> stream = TemporaryFile()
>>> data.write(stream)
>>> stream.close()
```

pyffi.formats.tri — TRI (.tri)

A .tri file contains facial expression data, that is, morphs for dynamic expressions such as smile, frown, and so on.

Implementation

class pyffi.formats.tri.**TriFormat**
Bases: pyffi.object_models.xml.FileFormat

This class implements the TRI format.

Data
alias of *Header*

class FileSignature(***kwargs*)
Bases: pyffi.object_models.xml.basic.BasicBase

Basic type which implements the header of a TRI file.

get_detail_display()
Return an object that can be used to display the instance.

get_hash(*data=None*)
Return a hash value for this value.
Returns An immutable object that can be used as a hash.

get_size(*data=None*)
Return number of bytes the header string occupies in a file.
Returns Number of bytes.

read(*stream*, *data*)
Read header string from stream and check it.
Parameters **stream** (*file*) – The stream to read from.

write(*stream*, *data*)
Write the header string to stream.
Parameters **stream** (*file*) – The stream to write to.

class FileVersion(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.basic.BasicBase

get_detail_display()
Return an object that can be used to display the instance.

get_hash(*data=None*)
Returns a hash value (an immutable object) that can be used to identify the object uniquely.

get_size(*data=None*)
Returns size of the object in bytes.

get_value()
Return object value.

read(*stream*, *data*)
Read object from file.

set_value(*value*)
Set object value.

write(*stream*, *data*)
Write object to file.

class Header(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.tri._Header, pyffi.object_models.Data

A class to contain the actual tri data.

add_modifier(*name=None*, *relative_vertices=None*)
Add a modifier.

add_morph(*name=None*, *relative_vertices=None*)
Add a morph.

get_global_child_nodes(*edge_filter=(True, True)*)
Generator which yields all children of this item in the global view, of given edge type (default is edges of type 0).

Override this method.
Returns Generator for global node children.

inspect(*stream*)
Quickly checks if stream contains TRI data, and reads everything up to the arrays.
Parameters **stream** (*file*) – The stream to inspect.

inspect_quick(*stream*)
Quickly checks if stream contains TRI data, by looking at the first 8 bytes. Reads the signature and the version.
Parameters **stream** (*file*) – The stream to inspect.

read(*stream*)
Read a tri file.
Parameters **stream** (file) – The stream from which to read.

write(*stream*)
Write a tri file.
Parameters **stream** (file) – The stream to which to write.

class ModifierRecord(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

A modifier replaces the vertices from the base model (Header.vertices) with those in Header.modifier_vertices. Note that Header.modifier_vertices counts up from the first modifier onwards. For example, if you were to take the 4th vertex to be modified in the 2nd modifier and the size of the 1st modifier was 10 vertices, then you would need Header.modifier_vertices[14]. Therefore, Header.num_modifier_vertices = sum(modifier.num_vertices_to_modify for modifier in Header.modifiers).

class MorphRecord(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.formats.tri._MorphRecord, object

```
>>> # create morph with 3 vertices.
>>> morph = TriFormat.MorphRecord(argument=3)
>>> morph.set_relative_vertices(
...     [(3, 5, 2), (1, 3, 2), (-9, 3, -1)])
>>> # scale should be 9/32768.0 = 0.0002746...
>>> morph.scale
0.0002746...
>>> for vert in morph.get_relative_vertices():
...     print([int(1000 * x + 0.5) for x in vert])
[3000, 5000, 2000]
[1000, 3000, 2000]
[-8999, 3000, -999]
```

apply_scale(*scale*)
Apply scale factor to data.

```
>>> # create morph with 3 vertices.
>>> morph = TriFormat.MorphRecord(argument=3)
>>> morph.set_relative_vertices(
...     [(3, 5, 2), (1, 3, 2), (-9, 3, -1)])
>>> morph.apply_scale(2)
>>> for vert in morph.get_relative_vertices():
...     print([int(1000 * x + 0.5) for x in vert])
[6000, 10000, 4000]
[2000, 6000, 4000]
[-17999, 6000, -1999]
```

class QuadFace(*template=None*, *argument=None*, *parent=None*)
Bases: pyffi.object_models.xml.struct_.StructBase

Not used by the Oblivion engine as it's tri based.

class SizedStringZ(***kwargs*)
Bases: pyffi.object_models.common.SizedString

get_size(*data=None*)
Return number of bytes this type occupies in a file.
Returns Number of bytes.

read(*stream*, *data*)
Read string from stream.
Parameters **stream** (*file*) – The stream to read from.

write(*stream*, *data*)
Write string to stream.
Parameters **stream** (*file*) – The stream to write to.

byte
alias of pyffi.object_models.common.Byte

char
alias of pyffi.object_models.common.Char

float
alias of pyffi.object_models.common.Float

int
alias of pyffi.object_models.common.Int

short
alias of pyffi.object_models.common.Short

ubyte
alias of pyffi.object_models.common.UByte

uint
alias of pyffi.object_models.common.UInt

ushort
alias of pyffi.object_models.common.UShort

static version_number(*version_str*)
Converts version string into an integer.

Parameters **version_str** (*str*) – The version string.

Returns A version integer.

```
>>> TriFormat.version_number('003')
3
>>> TriFormat.version_number('XXX')
-1
```

Regression tests

Read a TRI file

```
>>> # check and read tri file
>>> from os.path import dirname
>>> dirpath = __file__
>>> for i in range(4): #recurse up to root repo dir
...     dirpath = dirname(dirpath)
>>> repo_root = dirpath
>>> format_root = os.path.join(repo_root, 'tests', 'formats', 'tri')
>>> file = os.path.join(format_root, 'mmouthxivilai.tri')
>>> stream = open(file, 'rb')
>>> data = TriFormat.Data()
>>> data.inspect(stream)
>>> # do some stuff with header?
>>> data.num_vertices
89
>>> data.num_tri_faces
215
>>> data.num_quad_faces
0
>>> data.num_uvs
89
>>> data.num_morphs
18
>>> data.read(stream)
>>> print([str(morph.name.decode("ascii")) for morph in data.morphs])
['Fear', 'Surprise', 'Aah', 'BigAah', 'BMP', 'ChJSh', 'DST', 'Eee', 'Eh', 'FV', 'I',
↪'K', 'N', 'Oh', 'OohQ', 'R', 'Th', 'W']
```

Parse all TRI files in a directory tree

```
>>> for stream, data in TriFormat.walkData(format_root):
...     try:
...         # the replace call makes the doctest also pass on windows
...         os_path = stream.name
...         split = (os_path.split(os.sep))[-4:]
...         rejoin = os.path.join(*split).replace(os.sep, "/")
...         print("reading %s" % rejoin)
...     except Exception:
...         print(
...             "Warning: read failed due corrupt file,"
...             " corrupt format description, or bug.")
reading tests/formats/tri/mmouthxivilai.tri
```

Create an TRI file from scratch and write to file

```
>>> data = TriFormat.Data()
>>> from tempfile import TemporaryFile
>>> stream = TemporaryFile()
>>> data.write(stream)
```

Adding new formats

This section tries to explain how you can implement your own format in pyffi.

Getting Started

Note that the files which make up the following example can all be found in the examples/simple directory of the source distribution of pyffi.

Suppose you have a simple file format, which consists of an integer, followed by a list of integers as many as described by the first integer. We start by creating an XML file, call it `simple.xml`, which describes this format in a way that pyffi can understand:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE fileformat>
<fileformat version="1.0">
    <basic name="Int">A signed 32-bit integer.</basic>
    <struct name="Example">
        <add name="Num Integers" type="Int">
            Number of integers that follow.
        </add>
        <add name="Integers" type="Int" arr1="Num Integers">
            A list of integers.
        </add>
    </struct>
</fileformat>
```

What pyffi does is convert this simple XML description into Python classes which automatically can read and write the structure you've just described. Say this is the contents of `simple.py`:

```
import os
import pyffi.object_models.xml
import pyffi.object_models.common

class SimpleFormat(pyffi.object_models.xml.FileFormat):
    xml_file_name = 'simple.xml'
    xml_file_path = [ os.path.dirname(__file__) ]

    # basic types

    Int = pyffi.object_models.common.Int

    # extensions of generated types

    class Data(pyffi.object_models.xml.FileFormat.Data):
        def __init__(self):
            self.example = SimpleFormat.Example()
```

(continues on next page)

(continued from previous page)

```
        def read(self, stream):
            self.example.read(stream, self)

        def write(self, stream):
            self.example.write(stream, self)

    class Example:
        def addInteger(self, x):
            self.numIntegers += 1
            self.integers.update_size()
            self.integers[self.numIntegers-1] = x
```

What happens in this piece of code?

- The `pyffi.object_models.xml.FileFormat` base class triggers the transformation of xml into Python classes; how these classes can be used will be explained further.
- The `xml_file_name` class attribute provides the name of the xml file that describes the structures we wish to generate. The `xml_file_path` attribute gives a list of locations of where to look for this file; in our case we have simply chosen to put `simple.xml` in the same directory as `simple.py`.
- The `SimpleFormat.Example` class provides the generated class with a function `addInteger()` in addition to the attributes `numIntegers` and `integers` which have been created from the XML.
- Finally, the `pyffi.object_models.common` module implements the most common basic types, such as integers, characters, and floats. In the above example we have taken advantage of `pyffi.object_models.common.Int`, which defines a signed 32-bit integer, exactly the type we need.

Reading and Writing Files

To read the contents of a file of the format described by simple.xml:

```
from simple import SimpleFormat
x = SimpleFormat.Data()
f = open('somefile.simple', 'rb')
x.read(f)
f.close()
print(x.example)
```

Or, to create a new file in this format:

```
from simple import SimpleFormat
x = SimpleFormat.Data()
x.example.num_integers = 5
x.example.integers.update_size()
x.example.integers[0] = 3
x.example.integers[1] = 1
x.example.integers[2] = 4
x.example.integers[3] = 1
x.example.integers[4] = 5
f = open('pi.simple', 'wb')
x.write(f)
f.close()
```

Further References

With the above simple example in mind, you may wish to browse through the source code of *pyffi.formats.cgf* or *pyffi.formats.nif* to see how pyffi works for more complex file formats.

pyffi.spells — High level file operations

Note: This module is based on wz's NifTester module, although nothing of wz's original code is left in this module.

A *toaster*, implemented by subclasses of the abstract *Toaster* class, walks over all files in a folder, and applies one or more transformations on each file. Such transformations are called *spell*s, and are implemented by subclasses of the abstract *Spell* class.

A *spell* can also run independently of a *toaster* and be applied on a branch directly. The recommended way of doing this is via the *Spell.recurse()* method.

Supported spells

For format specific spells, refer to the corresponding module.

pyffi.spells.cgf — Crytek Geometry/Animation (.cgf/.cga) spells

Todo: Write documentation.

pyffi.spells.dds — DirectDraw Surface spells

There are no spells yet.

pyffi.spells.kfm — NetImmerse/Gamebryo Keyframe Motion (.kfm) spells

pyffi.spells.nif — NetImmerse/Gamebryo File/Keyframe (.nif/.kf/.kfa) spells

Module which contains all spells that check something in a NIF file.

Spells for dumping particular blocks from nifs.

pyffi.spells.nif.fix — spells to fix errors

Module which contains all spells that fix something in a nif.

Implementation

class pyffi.spells.nif.fix.**SpellDelTangentSpace**(*toaster=None*, *data=None*, *stream=None*)

Bases: pyffi.spells.nif.NifSpell

Delete tangentspace if it is present.

branchentry(*branch*)

Cast the spell on the given branch. First called with branch equal to data's children, then the grandchildren, and so on. The default implementation simply returns True.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (GlobalNode) – The branch to cast the spell on.

Returns True if the children must be processed, False otherwise.

Return type bool

branchinspect(*branch*)

Like _branchinspect(), but for customization: can be overridden to perform an extra inspection (the default implementation always returns True).

Parameters **branch** (GlobalNode) – The branch to check.

Returns True if the branch must be processed, False otherwise.

Return type bool

datainspect()

This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns True if the file must be processed, False otherwise.

Return type bool

class pyffi.spells.nif.fix.**SpellAddTangentSpace**(*toaster=None*, *data=None*, *stream=None*)

Bases: pyffi.spells.nif.NifSpell

Add tangentspace if none is present.

branchentry(*branch*)

Cast the spell on the given branch. First called with branch equal to data's children, then the grandchildren, and so on. The default implementation simply returns True.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (GlobalNode) – The branch to cast the spell on.

Returns True if the children must be processed, False otherwise.

Return type bool

branchinspect(*branch*)
Like `_branchinspect()`, but for customization: can be overridden to perform an extra inspection (the default implementation always returns `True`).

Parameters **branch** (`GlobalNode`) – The branch to check.

Returns `True` if the branch must be processed, `False` otherwise.

Return type `bool`

datainspect()
This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns `True` if the file must be processed, `False` otherwise.

Return type `bool`

class pyffi.spells.nif.fix.**SpellFFVT3RSkinPartition**(*toaster=None*, *data=None*, *stream=None*)
Bases: `pyffi.spells.nif.NifSpell`

Create or update skin partition, with settings that work for Freedom Force vs. The 3rd Reich.

branchentry(*branch*)
Cast the spell on the given branch. First called with branch equal to `data`'s children, then the grandchildren, and so on. The default implementation simply returns `True`.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (`GlobalNode`) – The branch to cast the spell on.

Returns `True` if the children must be processed, `False` otherwise.

Return type `bool`

branchinspect(*branch*)
Like `_branchinspect()`, but for customization: can be overridden to perform an extra inspection (the default implementation always returns `True`).

Parameters **branch** (`GlobalNode`) – The branch to check.

Returns `True` if the branch must be processed, `False` otherwise.

Return type `bool`

datainspect()
This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns `True` if the file must be processed, `False` otherwise.

Return type `bool`

class pyffi.spells.nif.fix.**SpellFixTexturePath**(*toaster=None*, *data=None*, *stream=None*)
Bases: `pyffi.spells.nif.fix.SpellParseTexturePath`

Fix the texture path. Transforms 0x0a into n and 0x0d into r. This fixes a bug in nifs saved with older versions of nifskope. Also transforms / into . This fixes problems when packing files into a bsa archive. Also if the version is 20.0.0.4 or higher it will check for bad texture path form of e.g. c:program filesbethsoftobtexturesfilepath.dds and replace it with e.g. texturesfilepath.dds.

substitute(*old_path*)
Helper function to allow subclasses of this spell to change part of the path with minimum of code. This implementation returns path unmodified.

class pyffi.spells.nif.fix.**SpellDetachHavokTriStripsData**(**args*, ***kwargs*)
Bases: pyffi.spells.nif.NifSpell

For NiTriStrips if their NiTriStripsData also occurs in a bhkNiTriStripsShape, make deep copy of data in havok. This is mainly useful as a preperation for other spells that act on NiTriStripsData, to ensure that the havok data remains untouched.

branchentry(*branch*)
Cast the spell on the given branch. First called with branch equal to data's children, then the grandchildren, and so on. The default implementation simply returns True.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (GlobalNode) – The branch to cast the spell on.

Returns True if the children must be processed, False otherwise.

Return type bool

branchinspect(*branch*)
Like _branchinspect(), but for customization: can be overridden to perform an extra inspection (the default implementation always returns True).

Parameters **branch** (GlobalNode) – The branch to check.

Returns True if the branch must be processed, False otherwise.

Return type bool

dataentry()
Called before all blocks are recursed. The default implementation simply returns True. You can access the data via data, and unlike in the *datainspect()* method, the full file has been processed at this stage.

Typically, you will override this function to perform a global operation on the file data.

Returns True if the children must be processed, False otherwise.

Return type bool

datainspect()
This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns True if the file must be processed, False otherwise.

Return type bool

class pyffi.spells.nif.fix.**SpellClampMaterialAlpha**(*toaster=None*, *data=None*, *stream=None*)
Bases: pyffi.spells.nif.NifSpell

Clamp corrupted material alpha values.

branchentry(*branch*)
Cast the spell on the given branch. First called with branch equal to data's children, then the grandchildren, and so on. The default implementation simply returns True.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (GlobalNode) – The branch to cast the spell on.

Returns True if the children must be processed, False otherwise.

Return type bool

branchinspect(*branch*)

Like _branchinspect(), but for customization: can be overridden to perform an extra inspection (the default implementation always returns True).

Parameters **branch** (GlobalNode) – The branch to check.

Returns True if the branch must be processed, False otherwise.

Return type bool

datainspect()

This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns True if the file must be processed, False otherwise.

Return type bool

class pyffi.spells.nif.fix.**SpellSendGeometriesToBindPosition**(*toaster=None*, *data=None*, *stream=None*)

Bases: pyffi.spells.nif.SpellVisitSkeletonRoots

Transform skinned geometries so similar bones have the same bone data, and hence, the same bind position, over all geometries.

skelrootentry(*branch*)

Do something with a skeleton root. Return value is ignored.

class pyffi.spells.nif.fix.**SpellSendDetachedGeometriesToNodePosition**(*toaster=None*, *data=None*, *stream=None*)

Bases: pyffi.spells.nif.SpellVisitSkeletonRoots

Transform geometries so each set of geometries that shares bones is aligned with the transform of the root bone of that set.

skelrootentry(*branch*)

Do something with a skeleton root. Return value is ignored.

class pyffi.spells.nif.fix.**SpellSendBonesToBindPosition**(*toaster=None*, *data=None*, *stream=None*)

Bases: pyffi.spells.nif.SpellVisitSkeletonRoots

Transform bones so bone data agrees with bone transforms, and hence, all bones are in bind position.

skelrootentry(*branch*)

Do something with a skeleton root. Return value is ignored.

class pyffi.spells.nif.fix.**SpellMergeSkeletonRoots**(*toaster=None*, *data=None*, *stream=None*)

Bases: pyffi.spells.nif.NifSpell

Merges skeleton roots in the NIF file so that no skeleton root has another skeleton root as child. Warns if merge is impossible (this happens if the global skin data of the geometry is not the unit transform).

branchentry(*branch*)
Cast the spell on the given branch. First called with branch equal to `data`'s children, then the grandchildren, and so on. The default implementation simply returns `True`.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (`GlobalNode`) – The branch to cast the spell on.

Returns `True` if the children must be processed, `False` otherwise.

Return type `bool`

branchinspect(*branch*)
Like `_branchinspect()`, but for customization: can be overridden to perform an extra inspection (the default implementation always returns `True`).

Parameters **branch** (`GlobalNode`) – The branch to check.

Returns `True` if the branch must be processed, `False` otherwise.

Return type `bool`

dataentry()
Called before all blocks are recursed. The default implementation simply returns `True`. You can access the data via `data`, and unlike in the *datainspect()* method, the full file has been processed at this stage.

Typically, you will override this function to perform a global operation on the file data.

Returns `True` if the children must be processed, `False` otherwise.

Return type `bool`

datainspect()
This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns `True` if the file must be processed, `False` otherwise.

Return type `bool`

class pyffi.spells.nif.fix.**SpellApplySkinDeformation**(*toaster=None*, *data=None*, *stream=None*)
Bases: `pyffi.spells.nif.NifSpell`

Apply skin deformation to nif.

class pyffi.spells.nif.fix.**SpellScale**(*toaster=None*, *data=None*, *stream=None*)
Bases: `pyffi.spells.nif.NifSpell`

Scale a model.

branchentry(*branch*)
Cast the spell on the given branch. First called with branch equal to `data`'s children, then the grandchildren, and so on. The default implementation simply returns `True`.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (`GlobalNode`) – The branch to cast the spell on.

Returns `True` if the children must be processed, `False` otherwise.

Return type `bool`

branchinspect(*branch*)

Like `_branchinspect()`, but for customization: can be overridden to perform an extra inspection (the default implementation always returns `True`).

Parameters **branch** (`GlobalNode`) – The branch to check.

Returns `True` if the branch must be processed, `False` otherwise.

Return type `bool`

dataentry()

Called before all blocks are recursed. The default implementation simply returns `True`. You can access the data via `data`, and unlike in the `datainspect()` method, the full file has been processed at this stage.

Typically, you will override this function to perform a global operation on the file data.

Returns `True` if the children must be processed, `False` otherwise.

Return type `bool`

classmethod toastentry(*toaster*)

Called just before the toaster starts processing all files. If it returns `False`, then the spell is not used. The default implementation simply returns `True`.

For example, if the spell only acts on a particular block type, but that block type is excluded, then you can use this function to flag that this spell can be skipped. You can also use this function to initialize statistics data to be aggregated from files, to initialize a log file, and so.

Parameters **toaster** (`Toaster`) – The toaster this spell is called from.

Returns `True` if the spell applies, `False` otherwise.

Return type `bool`

class pyffi.spells.nif.fix.**SpellFixCenterRadius**(*toaster=None*, *data=None*, *stream=None*)

Bases: `pyffi.spells.nif.check.SpellCheckCenterRadius`

Recalculate geometry centers and radii.

class pyffi.spells.nif.fix.**SpellFixSkinCenterRadius**(*toaster=None*, *data=None*, *stream=None*)

Bases: `pyffi.spells.nif.check.SpellCheckSkinCenterRadius`

Recalculate skin centers and radii.

class pyffi.spells.nif.fix.**SpellFixMopp**(*toaster=None*, *data=None*, *stream=None*)

Bases: `pyffi.spells.nif.check.SpellCheckMopp`

Recalculate mopp data from collision geometry.

branchentry(*branch*)

Cast the spell on the given branch. First called with branch equal to `data`'s children, then the grandchildren, and so on. The default implementation simply returns `True`.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (`GlobalNode`) – The branch to cast the spell on.

Returns `True` if the children must be processed, `False` otherwise.

Return type `bool`

class pyffi.spells.nif.fix.**SpellCleanStringPalette**(*toaster=None*, *data=None*, *stream=None*)

Bases: pyffi.spells.nif.NifSpell

Remove unused strings from string palette.

branchentry(*branch*)

Parses string palette of either a single controller sequence, or of all controller sequences in a controller manager.

```
>>> seq = NifFormat.NiControllerSequence()
>>> seq.string_palette = NifFormat.NiStringPalette()
>>> block = seq.add_controlled_block()
>>> block.string_palette = seq.string_palette
>>> block.set_variable_1("there")
>>> block.set_node_name("hello")
>>> block.string_palette.palette.add_string("test")
12
>>> seq.string_palette.palette.get_all_strings()
[b'there', b'hello', b'test']
>>> SpellCleanStringPalette().branchentry(seq)
pyffi.toaster:INFO:parsing string palette
False
>>> seq.string_palette.palette.get_all_strings()
[b'hello', b'there']
>>> block.get_variable_1()
b'there'
>>> block.get_node_name()
b'hello'
```

branchinspect(*branch*)

Like _branchinspect(), but for customization: can be overridden to perform an extra inspection (the default implementation always returns True).

Parameters **branch** (GlobalNode) – The branch to check.

Returns True if the branch must be processed, False otherwise.

Return type bool

datainspect()

This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns True if the file must be processed, False otherwise.

Return type bool

substitute(*old_string*)

Helper function to substitute strings in the string palette, to allow subclasses of this spell can modify the strings. This implementation returns string unmodified.

class pyffi.spells.nif.fix.**SpellDelUnusedRoots**(*toaster=None*, *data=None*, *stream=None*)

Bases: pyffi.spells.nif.NifSpell

Remove root branches that shouldn't be root branches and are unused in the file such as NiProperty branches that are not properly parented.

dataentry()

Called before all blocks are recursed. The default implementation simply returns True. You can access

the data via `data`, and unlike in the *`datainspect()`* method, the full file has been processed at this stage.

Typically, you will override this function to perform a global operation on the file data.

Returns `True` if the children must be processed, `False` otherwise.

Return type `bool`

datainspect()
This is called after *`pyffi.object_models.FileFormat.Data.inspect()`* has been called, and before *`pyffi.object_models.FileFormat.Data.read()`* is called. Override this function for customization.

Returns `True` if the file must be processed, `False` otherwise.

Return type `bool`

class pyffi.spells.nif.fix.**SpellFixEmptySkeletonRoots**(*toaster=None*, *data=None*, *stream=None*)
Bases: `pyffi.spells.nif.NifSpell`

Fix empty skeleton roots in an as sane as possible way.

branchentry(*branch*)
Cast the spell on the given branch. First called with branch equal to `data`'s children, then the grandchildren, and so on. The default implementation simply returns `True`.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (`GlobalNode`) – The branch to cast the spell on.

Returns `True` if the children must be processed, `False` otherwise.

Return type `bool`

branchinspect(*branch*)
Like `_branchinspect()`, but for customization: can be overridden to perform an extra inspection (the default implementation always returns `True`).

Parameters **branch** (`GlobalNode`) – The branch to check.

Returns `True` if the branch must be processed, `False` otherwise.

Return type `bool`

dataentry()
Called before all blocks are recursed. The default implementation simply returns `True`. You can access the data via `data`, and unlike in the *`datainspect()`* method, the full file has been processed at this stage.

Typically, you will override this function to perform a global operation on the file data.

Returns `True` if the children must be processed, `False` otherwise.

Return type `bool`

datainspect()
This is called after *`pyffi.object_models.FileFormat.Data.inspect()`* has been called, and before *`pyffi.object_models.FileFormat.Data.read()`* is called. Override this function for customization.

Returns `True` if the file must be processed, `False` otherwise.

Return type `bool`

Regression tests

Spells for optimizing NIF files.

class pyffi.spells.nif.optimize.**SpellCleanRefLists**(*toaster=None*, *data=None*, *stream=None*)

Bases: pyffi.spells.nif.NifSpell

Remove empty and duplicate entries in reference lists.

branchentry(*branch*)

Cast the spell on the given branch. First called with branch equal to data's children, then the grandchildren, and so on. The default implementation simply returns True.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (GlobalNode) – The branch to cast the spell on.

Returns True if the children must be processed, False otherwise.

Return type bool

branchinspect(*branch*)

Like _branchinspect(), but for customization: can be overridden to perform an extra inspection (the default implementation always returns True).

Parameters **branch** (GlobalNode) – The branch to check.

Returns True if the branch must be processed, False otherwise.

Return type bool

cleanreflist(*reflist*, *category*)

Return a cleaned copy of the given list of references.

dataentry()

Called before all blocks are recursed. The default implementation simply returns True. You can access the data via data, and unlike in the *datainspect()* method, the full file has been processed at this stage.

Typically, you will override this function to perform a global operation on the file data.

Returns True if the children must be processed, False otherwise.

Return type bool

datainspect()

This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns True if the file must be processed, False otherwise.

Return type bool

class pyffi.spells.nif.optimize.**SpellMergeDuplicates**(**args*, ***kwargs*)

Bases: pyffi.spells.nif.NifSpell

Remove duplicate branches.

branchentry(*branch*)

Cast the spell on the given branch. First called with branch equal to data's children, then the grandchildren, and so on. The default implementation simply returns True.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (GlobalNode) – The branch to cast the spell on.

Returns True if the children must be processed, False otherwise.

Return type bool

branchinspect(*branch*)

Like _branchinspect(), but for customization: can be overridden to perform an extra inspection (the default implementation always returns True).

Parameters **branch** (GlobalNode) – The branch to check.

Returns True if the branch must be processed, False otherwise.

Return type bool

datainspect()

This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns True if the file must be processed, False otherwise.

Return type bool

class pyffi.spells.nif.optimize.**SpellOptimizeGeometry**(**args*, ***kwargs*)

Bases: pyffi.spells.nif.NifSpell

Optimize all geometries: - remove duplicate vertices - triangulate - recalculate skin partition - recalculate tangent space

branchentry(*branch*)

Optimize a NiTriStrips or NiTriShape block:

- remove duplicate vertices
- retriangulate for vertex cache
- recalculate skin partition
- recalculate tangent space

Todo: Limit the size of shapes (see operation optimization mod for Oblivion!)

---

branchinspect(*branch*)

Like _branchinspect(), but for customization: can be overridden to perform an extra inspection (the default implementation always returns True).

Parameters **branch** (GlobalNode) – The branch to check.

Returns True if the branch must be processed, False otherwise.

Return type bool

datainspect()

This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns True if the file must be processed, False otherwise.

Return type bool

class pyffi.spells.nif.optimize.**SpellOptimize**(*toaster=None*, *data=None*, *stream=None*)

Bases: pyffi.spells.SpellCleanFarNifSpellDelUnusedRootsSpellCleanRefListsSpellDetachHavo

Global fixer and optimizer spell.

class pyffi.spells.nif.optimize.**SpellDelUnusedBones**(*toaster=None*, *data=None*, *stream=None*)

Bases: pyffi.spells.nif.NifSpell

Remove nodes that are not used for anything.

branchentry(*branch*)

Cast the spell on the given branch. First called with branch equal to data's children, then the grandchildren, and so on. The default implementation simply returns True.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (GlobalNode) – The branch to cast the spell on.

Returns True if the children must be processed, False otherwise.

Return type bool

branchinspect(*branch*)

Like _branchinspect(), but for customization: can be overridden to perform an extra inspection (the default implementation always returns True).

Parameters **branch** (GlobalNode) – The branch to check.

Returns True if the branch must be processed, False otherwise.

Return type bool

dataentry()

Called before all blocks are recursed. The default implementation simply returns True. You can access the data via data, and unlike in the *datainspect()* method, the full file has been processed at this stage.

Typically, you will override this function to perform a global operation on the file data.

Returns True if the children must be processed, False otherwise.

Return type bool

datainspect()

This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns True if the file must be processed, False otherwise.

Return type bool

pyffi.spells.nif.modify — spells to make modifications

Module which contains all spells that modify a nif.

class pyffi.spells.nif.modify.**SpellTexturePath**(*toaster=None*, *data=None*, *stream=None*)

Bases: pyffi.spells.nif.fix.SpellParseTexturePath

Changes the texture path while keeping the texture names.

substitute(*old_path*)

Helper function to allow subclasses of this spell to change part of the path with minimum of code. This implementation returns path unmodified.

classmethod toastentry(*toaster*)

Called just before the toaster starts processing all files. If it returns `False`, then the spell is not used. The default implementation simply returns `True`.

For example, if the spell only acts on a particular block type, but that block type is excluded, then you can use this function to flag that this spell can be skipped. You can also use this function to initialize statistics data to be aggregated from files, to initialize a log file, and so.

Parameters **toaster** (`Toaster`) – The toaster this spell is called from.

Returns `True` if the spell applies, `False` otherwise.

Return type `bool`

class pyffi.spells.nif.modify.**SpellSubstituteTexturePath**(*toaster=None*, *data=None*, *stream=None*)

Bases: *pyffi.spells.nif.fix.SpellFixTexturePath*

Runs a regex replacement on texture paths.

substitute(*old_path*)

Returns modified texture path, and reports if path was modified.

classmethod toastentry(*toaster*)

Called just before the toaster starts processing all files. If it returns `False`, then the spell is not used. The default implementation simply returns `True`.

For example, if the spell only acts on a particular block type, but that block type is excluded, then you can use this function to flag that this spell can be skipped. You can also use this function to initialize statistics data to be aggregated from files, to initialize a log file, and so.

Parameters **toaster** (`Toaster`) – The toaster this spell is called from.

Returns `True` if the spell applies, `False` otherwise.

Return type `bool`

class pyffi.spells.nif.modify.**SpellLowResTexturePath**(*toaster=None*, *data=None*, *stream=None*)

Bases: *pyffi.spells.nif.modify.SpellSubstituteTexturePath*

Changes the texture path by replacing 'textures*' with 'textureslowres*' - used mainly for making _far.nifs

substitute(*old_path*)

Returns modified texture path, and reports if path was modified.

classmethod toastentry(*toaster*)

Called just before the toaster starts processing all files. If it returns `False`, then the spell is not used. The default implementation simply returns `True`.

For example, if the spell only acts on a particular block type, but that block type is excluded, then you can use this function to flag that this spell can be skipped. You can also use this function to initialize statistics data to be aggregated from files, to initialize a log file, and so.

Parameters **toaster** (Toaster) – The toaster this spell is called from.

Returns True if the spell applies, False otherwise.

Return type bool

class pyffi.spells.nif.modify.**SpellCollisionType**(*toaster=None*, *data=None*, *stream=None*)

Bases: pyffi.spells.nif.NifSpell

Sets the object collision to be a different type

branchentry(*branch*)

Cast the spell on the given branch. First called with branch equal to data's children, then the grandchildren, and so on. The default implementation simply returns True.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (GlobalNode) – The branch to cast the spell on.

Returns True if the children must be processed, False otherwise.

Return type bool

branchinspect(*branch*)

Like _branchinspect(), but for customization: can be overridden to perform an extra inspection (the default implementation always returns True).

Parameters **branch** (GlobalNode) – The branch to check.

Returns True if the branch must be processed, False otherwise.

Return type bool

datainspect()

This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns True if the file must be processed, False otherwise.

Return type bool

classmethod **toastentry**(*toaster*)

Called just before the toaster starts processing all files. If it returns False, then the spell is not used. The default implementation simply returns True.

For example, if the spell only acts on a particular block type, but that block type is excluded, then you can use this function to flag that this spell can be skipped. You can also use this function to initialize statistics data to be aggregated from files, to initialize a log file, and so.

Parameters **toaster** (Toaster) – The toaster this spell is called from.

Returns True if the spell applies, False otherwise.

Return type bool

class pyffi.spells.nif.modify.**SpellCollisionMaterial**(*toaster=None*, *data=None*, *stream=None*)

Bases: pyffi.spells.nif.NifSpell

Sets the object's collision material to be a different type

branchentry(*branch*)
Cast the spell on the given branch. First called with branch equal to `data`'s children, then the grandchildren, and so on. The default implementation simply returns `True`.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (`GlobalNode`) – The branch to cast the spell on.

Returns `True` if the children must be processed, `False` otherwise.

Return type `bool`

branchinspect(*branch*)
Like `_branchinspect()`, but for customization: can be overridden to perform an extra inspection (the default implementation always returns `True`).

Parameters **branch** (`GlobalNode`) – The branch to check.

Returns `True` if the branch must be processed, `False` otherwise.

Return type `bool`

datainspect()
This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns `True` if the file must be processed, `False` otherwise.

Return type `bool`

classmethod toastentry(*toaster*)
Called just before the toaster starts processing all files. If it returns `False`, then the spell is not used. The default implementation simply returns `True`.

For example, if the spell only acts on a particular block type, but that block type is excluded, then you can use this function to flag that this spell can be skipped. You can also use this function to initialize statistics data to be aggregated from files, to initialize a log file, and so.

Parameters **toaster** (`Toaster`) – The toaster this spell is called from.

Returns `True` if the spell applies, `False` otherwise.

Return type `bool`

class pyffi.spells.nif.modify.**SpellScaleAnimationTime**(*toaster=None*, *data=None*, *stream=None*)
Bases: `pyffi.spells.nif.NifSpell`

Scales the animation time.

branchentry(*branch*)
Cast the spell on the given branch. First called with branch equal to `data`'s children, then the grandchildren, and so on. The default implementation simply returns `True`.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (`GlobalNode`) – The branch to cast the spell on.

Returns `True` if the children must be processed, `False` otherwise.

Return type `bool`

branchinspect(*branch*)
Like `_branchinspect()`, but for customization: can be overridden to perform an extra inspection (the default implementation always returns `True`).

Parameters **branch** (`GlobalNode`) – The branch to check.

Returns `True` if the branch must be processed, `False` otherwise.

Return type `bool`

datainspect()
This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns `True` if the file must be processed, `False` otherwise.

Return type `bool`

classmethod toastentry(*toaster*)
Called just before the toaster starts processing all files. If it returns `False`, then the spell is not used. The default implementation simply returns `True`.

For example, if the spell only acts on a particular block type, but that block type is excluded, then you can use this function to flag that this spell can be skipped. You can also use this function to initialize statistics data to be aggregated from files, to initialize a log file, and so.

Parameters **toaster** (`Toaster`) – The toaster this spell is called from.

Returns `True` if the spell applies, `False` otherwise.

Return type `bool`

class pyffi.spells.nif.modify.**SpellReverseAnimation**(*toaster=None*, *data=None*, *stream=None*)
Bases: `pyffi.spells.nif.NifSpell`

Reverses the animation by reversing datas in relation to the time.

branchentry(*branch*)
Cast the spell on the given branch. First called with branch equal to `data`'s children, then the grandchildren, and so on. The default implementation simply returns `True`.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (`GlobalNode`) – The branch to cast the spell on.

Returns `True` if the children must be processed, `False` otherwise.

Return type `bool`

branchinspect(*branch*)
Like `_branchinspect()`, but for customization: can be overridden to perform an extra inspection (the default implementation always returns `True`).

Parameters **branch** (`GlobalNode`) – The branch to check.

Returns `True` if the branch must be processed, `False` otherwise.

Return type `bool`

datainspect()
This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns `True` if the file must be processed, `False` otherwise.

Return type `bool`

class pyffi.spells.nif.modify.**SpellSubstituteStringPalette**(*toaster=None*, *data=None*, *stream=None*)

Bases: *pyffi.spells.nif.fix.SpellCleanStringPalette*

Substitute strings in a string palette.

substitute(*old_string*)

Returns modified string, and reports if string was modified.

classmethod toastentry(*toaster*)

Called just before the toaster starts processing all files. If it returns `False`, then the spell is not used. The default implementation simply returns `True`.

For example, if the spell only acts on a particular block type, but that block type is excluded, then you can use this function to flag that this spell can be skipped. You can also use this function to initialize statistics data to be aggregated from files, to initialize a log file, and so.

Parameters **toaster** (`Toaster`) – The toaster this spell is called from.

Returns `True` if the spell applies, `False` otherwise.

Return type `bool`

class pyffi.spells.nif.modify.**SpellChangeBonePriorities**(*toaster=None*, *data=None*, *stream=None*)

Bases: `pyffi.spells.nif.NifSpell`

Changes controlled block priorities based on controlled block name.

branchentry(*branch*)

Cast the spell on the given branch. First called with branch equal to `data`'s children, then the grandchildren, and so on. The default implementation simply returns `True`.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (`GlobalNode`) – The branch to cast the spell on.

Returns `True` if the children must be processed, `False` otherwise.

Return type `bool`

branchinspect(*branch*)

Like `_branchinspect()`, but for customization: can be overridden to perform an extra inspection (the default implementation always returns `True`).

Parameters **branch** (`GlobalNode`) – The branch to check.

Returns `True` if the branch must be processed, `False` otherwise.

Return type `bool`

datainspect()

This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns `True` if the file must be processed, `False` otherwise.

Return type `bool`

classmethod toastentry(*toaster*)
Called just before the toaster starts processing all files. If it returns `False`, then the spell is not used. The default implementation simply returns `True`.

For example, if the spell only acts on a particular block type, but that block type is excluded, then you can use this function to flag that this spell can be skipped. You can also use this function to initialize statistics data to be aggregated from files, to initialize a log file, and so.

Parameters **toaster** (`Toaster`) – The toaster this spell is called from.

Returns `True` if the spell applies, `False` otherwise.

Return type `bool`

class pyffi.spells.nif.modify.**SpellSetInterpolatorTransRotScale**(*toaster=None*, *data=None*, *stream=None*)
Bases: `pyffi.spells.nif.NifSpell`

Changes specified bone(s) translations/rotations in their NiTransformInterpolator.

branchentry(*branch*)
Cast the spell on the given branch. First called with branch equal to `data`'s children, then the grandchildren, and so on. The default implementation simply returns `True`.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (`GlobalNode`) – The branch to cast the spell on.

Returns `True` if the children must be processed, `False` otherwise.

Return type `bool`

branchinspect(*branch*)
Like `_branchinspect()`, but for customization: can be overridden to perform an extra inspection (the default implementation always returns `True`).

Parameters **branch** (`GlobalNode`) – The branch to check.

Returns `True` if the branch must be processed, `False` otherwise.

Return type `bool`

datainspect()
This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns `True` if the file must be processed, `False` otherwise.

Return type `bool`

classmethod toastentry(*toaster*)
Called just before the toaster starts processing all files. If it returns `False`, then the spell is not used. The default implementation simply returns `True`.

For example, if the spell only acts on a particular block type, but that block type is excluded, then you can use this function to flag that this spell can be skipped. You can also use this function to initialize statistics data to be aggregated from files, to initialize a log file, and so.

Parameters **toaster** (`Toaster`) – The toaster this spell is called from.

Returns `True` if the spell applies, `False` otherwise.

Return type `bool`

class pyffi.spells.nif.modify.**SpellDelInterpolatorTransformData**(*toaster=None*, *data=None*, *stream=None*)

Bases: pyffi.spells.nif.NifSpell

Deletes the specified bone(s) NiTransformData(s).

branchentry(*branch*)

Cast the spell on the given branch. First called with branch equal to data's children, then the grandchildren, and so on. The default implementation simply returns True.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (GlobalNode) – The branch to cast the spell on.

Returns True if the children must be processed, False otherwise.

Return type bool

branchinspect(*branch*)

Like _branchinspect(), but for customization: can be overridden to perform an extra inspection (the default implementation always returns True).

Parameters **branch** (GlobalNode) – The branch to check.

Returns True if the branch must be processed, False otherwise.

Return type bool

datainspect()

This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns True if the file must be processed, False otherwise.

Return type bool

classmethod **toastentry**(*toaster*)

Called just before the toaster starts processing all files. If it returns False, then the spell is not used. The default implementation simply returns True.

For example, if the spell only acts on a particular block type, but that block type is excluded, then you can use this function to flag that this spell can be skipped. You can also use this function to initialize statistics data to be aggregated from files, to initialize a log file, and so.

Parameters **toaster** (Toaster) – The toaster this spell is called from.

Returns True if the spell applies, False otherwise.

Return type bool

class pyffi.spells.nif.modify.**SpellDelBranches**(*toaster=None*, *data=None*, *stream=None*)

Bases: pyffi.spells.nif.NifSpell

Delete blocks that match the exclude list.

branchentry(*branch*)

Strip branch if it is flagged for deletion.

is_branch_to_be_deleted(*branch*)

Returns True for those branches that must be deleted. The default implementation returns True for

branches that are not admissible as specified by include/exclude options of the toaster. Override in subclasses that must delete specific branches.

class pyffi.spells.nif.modify.**_SpellDelBranchClasses**(*toaster=None*, *data=None*, *stream=None*)

Bases: *pyffi.spells.nif.modify.SpellDelBranches*

Delete blocks that match a given list. Only useful as base class for other spells.

BRANCH_CLASSES_TO_BE_DELETED = ()
List of branch classes that have to be deleted.

datainspect()
This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns `True` if the file must be processed, `False` otherwise.

Return type `bool`

is_branch_to_be_deleted(*branch*)
Returns `True` for those branches that must be deleted. The default implementation returns `True` for branches that are not admissible as specified by include/exclude options of the toaster. Override in subclasses that must delete specific branches.

class pyffi.spells.nif.modify.**SpellDelSkinShapes**(*toaster=None*, *data=None*, *stream=None*)

Bases: *pyffi.spells.nif.modify.SpellDelBranches*

Delete any geometries with a material name of 'skin'

branchinspect(*branch*)
Like `_branchinspect()`, but for customization: can be overridden to perform an extra inspection (the default implementation always returns `True`).

Parameters **branch** (`GlobalNode`) – The branch to check.

Returns `True` if the branch must be processed, `False` otherwise.

Return type `bool`

is_branch_to_be_deleted(*branch*)
Returns `True` for those branches that must be deleted. The default implementation returns `True` for branches that are not admissible as specified by include/exclude options of the toaster. Override in subclasses that must delete specific branches.

class pyffi.spells.nif.modify.**SpellDisableParallax**(*toaster=None*, *data=None*, *stream=None*)

Bases: `pyffi.spells.nif.NifSpell`

Disable parallax shader (for Oblivion, but may work on other nifs too).

branchentry(*branch*)
Cast the spell on the given branch. First called with branch equal to `data`'s children, then the grandchildren, and so on. The default implementation simply returns `True`.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (`GlobalNode`) – The branch to cast the spell on.

Returns `True` if the children must be processed, `False` otherwise.

Return type `bool`

branchinspect(*branch*)
Like `_branchinspect()`, but for customization: can be overridden to perform an extra inspection (the default implementation always returns `True`).

Parameters **branch** (`GlobalNode`) – The branch to check.

Returns `True` if the branch must be processed, `False` otherwise.

Return type `bool`

datainspect()
This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns `True` if the file must be processed, `False` otherwise.

Return type `bool`

class pyffi.spells.nif.modify.**SpellAddStencilProperty**(*toaster=None*, *data=None*, *stream=None*)
Bases: `pyffi.spells.nif.NifSpell`

Adds a NiStencilProperty to each geometry if it is not present.

branchentry(*branch*)
Cast the spell on the given branch. First called with branch equal to `data`'s children, then the grandchildren, and so on. The default implementation simply returns `True`.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (`GlobalNode`) – The branch to cast the spell on.

Returns `True` if the children must be processed, `False` otherwise.

Return type `bool`

branchinspect(*branch*)
Like `_branchinspect()`, but for customization: can be overridden to perform an extra inspection (the default implementation always returns `True`).

Parameters **branch** (`GlobalNode`) – The branch to check.

Returns `True` if the branch must be processed, `False` otherwise.

Return type `bool`

datainspect()
This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns `True` if the file must be processed, `False` otherwise.

Return type `bool`

class pyffi.spells.nif.modify.**SpellDelVertexColor**(*toaster=None*, *data=None*, *stream=None*)
Bases: *pyffi.spells.nif.modify.SpellDelBranches*

Delete vertex color properties and vertex color data.

branchentry(*branch*)
Strip branch if it is flagged for deletion.

branchinspect(*branch*)
Like `_branchinspect()`, but for customization: can be overridden to perform an extra inspection (the default implementation always returns `True`).

Parameters **branch** (`GlobalNode`) – The branch to check.

Returns `True` if the branch must be processed, `False` otherwise.

Return type `bool`

datainspect()
This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns `True` if the file must be processed, `False` otherwise.

Return type `bool`

is_branch_to_be_deleted(*branch*)
Returns `True` for those branches that must be deleted. The default implementation returns `True` for branches that are not admissible as specified by include/exclude options of the toaster. Override in subclasses that must delete specific branches.

class pyffi.spells.nif.modify.**SpellMakeSkinlessNif**(*toaster=None*, *data=None*, *stream=None*)
Bases: `pyffi.spells.SpellDelSkinShapesSpellAddStencilProperty`

Spell to make fleshless CMR (Custom Model Races) clothing/armour type nifs.

class pyffi.spells.nif.modify.**SpellCleanFarNif**(*toaster=None*, *data=None*, *stream=None*)
Bases: `pyffi.spells.SpellDelVertexColorPropertySpellDelAlphaPropertySpellDelSpecularPrope`

Spell to clean _far type nifs (for even more optimizations, combine this with the optimize spell).

datainspect()
Inspect every spell with L{Spell.datainspect} and keep those spells that must be cast.

class pyffi.spells.nif.modify.**SpellMakeFarNif**(*toaster=None*, *data=None*, *stream=None*)
Bases: `pyffi.spells.SpellDelVertexColorPropertySpellDelAlphaPropertySpellDelSpecularPrope`

Spell to make _far type nifs (for even more optimizations, combine this with the optimize spell).

pyffi.spells.tga — Targa spells

There are no spells yet.

Some spells are applicable on every file format, and those are documented here.

class pyffi.spells.**SpellApplyPatch**(*toaster=None*, *data=None*, *stream=None*)
Bases: *pyffi.spells.Spell*

A spell for applying a patch on files.

datainspect()
There is no need to read the whole file, so we apply the patch already at inspection stage, and stop the spell process by returning `False`.

Returns `False`

Return type `bool`

Adding new spells

To create new spells, derive your custom spells from the *Spell* class, and include them in the *Toaster.SPELLS* attribute of your toaster.

class pyffi.spells.**Spell**(*toaster=None*, *data=None*, *stream=None*)
Bases: object

Spell base class. A spell takes a data file and then does something useful with it. The main entry point for spells is *recurse()*, so if you are writing new spells, start with reading the documentation with *recurse()*.

READONLY = True
A bool which determines whether the spell is read only or not. Default value is True. Override this class attribute, and set to False, when subclassing a spell that must write files back to the disk.

SPELLNAME = None
A str describing how to refer to the spell from the command line. Override this class attribute when subclassing.

__init__(*toaster=None*, *data=None*, *stream=None*)
Initialize the spell data.

Parameters

- **data** (*Data*) – The file *data*.
- **stream** (file) – The file *stream*.
- **toaster** (*Toaster*) – The *toaster* this spell is called from (optional).

_branchinspect(*branch*)
Check if spell should be cast on this branch or not, based on exclude and include options passed on the command line. You should not need to override this function: if you need additional checks on whether a branch must be parsed or not, override the *branchinspect()* method.

Parameters **branch** (GlobalNode) – The branch to check.

Returns True if the branch must be processed, False otherwise.

Return type bool

_datainspect()
This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called.

Returns True if the file must be processed, False otherwise.

Return type bool

branchentry(*branch*)
Cast the spell on the given branch. First called with branch equal to *data*'s children, then the grandchildren, and so on. The default implementation simply returns True.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (GlobalNode) – The branch to cast the spell on.

Returns True if the children must be processed, False otherwise.

Return type bool

branchexit(*branch*)
Cast a spell on the given branch, after all its children, grandchildren, have been processed, if *branchentry()* returned True on the given branch.

Typically, you will override this function to perform a particular operation on a block type, but you rely on the fact that the children must have been processed first.

Parameters **branch** (GlobalNode) – The branch to cast the spell on.

branchinspect(*branch*)
Like *_branchinspect()*, but for customization: can be overridden to perform an extra inspection (the default implementation always returns True).

Parameters **branch** (GlobalNode) – The branch to check.

Returns True if the branch must be processed, False otherwise.

Return type bool

data = None
The *Data* instance this spell acts on.

dataentry()
Called before all blocks are recursed. The default implementation simply returns True. You can access the data via *data*, and unlike in the *datainspect()* method, the full file has been processed at this stage.

Typically, you will override this function to perform a global operation on the file data.

Returns True if the children must be processed, False otherwise.

Return type bool

dataexit()
Called after all blocks have been processed, if *dataentry()* returned True.

Typically, you will override this function to perform a final spell operation, such as writing back the file in a special way, or making a summary log.

datainspect()
This is called after *pyffi.object_models.FileFormat.Data.inspect()* has been called, and before *pyffi.object_models.FileFormat.Data.read()* is called. Override this function for customization.

Returns True if the file must be processed, False otherwise.

Return type bool

recurse(*branch=None*)
Helper function which calls *_branchinspect()* and *branchinspect()* on the branch, if both successful then *branchentry()* on the branch, and if this is succesful it calls *recurse()* on the branch's children, and once all children are done, it calls *branchexit()*.

Note that *_branchinspect()* and *branchinspect()* are not called upon first entry of this function, that is, when called with *data* as branch argument. Use *datainspect()* to stop recursion into this branch.

Do not override this function.

Parameters **branch** (GlobalNode) – The branch to start the recursion from, or None to recurse the whole tree.

stream = None
The current file being processed.

classmethod toastentry(*toaster*)
Called just before the toaster starts processing all files. If it returns False, then the spell is not used. The default implementation simply returns True.

For example, if the spell only acts on a particular block type, but that block type is excluded, then you can use this function to flag that this spell can be skipped. You can also use this function to initialize statistics data to be aggregated from files, to initialize a log file, and so.

Parameters **toaster** (*Toaster*) – The toaster this spell is called from.

Returns `True` if the spell applies, `False` otherwise.

Return type `bool`

toaster = None
The *Toaster* instance this spell is called from.

classmethod toastexit(*toaster*)
Called when the toaster has finished processing all files.

Parameters **toaster** (*Toaster*) – The toaster this spell is called from.

Grouping spells together

It is also possible to create composite spells, that is, spells that simply execute other spells. The following functions and classes can be used for this purpose.

pyffi.spells.**SpellGroupParallel**(**args*)
Class factory for grouping spells in parallel.

pyffi.spells.**SpellGroupSeries**(**args*)
Class factory for grouping spells in series.

class pyffi.spells.**SpellGroupBase**(*toaster=None*, *data=None*, *stream=None*)
Bases: *pyffi.spells.Spell*

Base class for grouping spells. This implements all the spell grouping functions that fall outside of the actual recursing (`__init__()`, *toastentry()*, `_datainspect()`, *datainspect()*, and *toastexit()*).

ACTIVESPELLCLASSES = []
List of active spells of this group (not instantiated). This list is automatically built when *toastentry()* is called.

SPELLCLASSES = []
List of *Spell*s of this group (not instantiated).

datainspect()
Inspect every spell with L{Spell.datainspect} and keep those spells that must be cast.

spells = []
List of active spell instances.

classmethod toastentry(*toaster*)
Called just before the toaster starts processing all files. If it returns `False`, then the spell is not used. The default implementation simply returns `True`.

For example, if the spell only acts on a particular block type, but that block type is excluded, then you can use this function to flag that this spell can be skipped. You can also use this function to initialize statistics data to be aggregated from files, to initialize a log file, and so.

Parameters **toaster** (*Toaster*) – The toaster this spell is called from.

Returns `True` if the spell applies, `False` otherwise.

Return type `bool`

classmethod toastexit(*toaster*)
Called when the toaster has finished processing all files.

Parameters **toaster** (*Toaster*) – The toaster this spell is called from.

class pyffi.spells.**SpellGroupParallelBase**(*toaster=None*, *data=None*, *stream=None*)
Bases: *pyffi.spells.SpellGroupBase*

Base class for running spells in parallel (that is, with only a single recursion in the tree).

branchentry(*branch*)
Run all spells.

branchexit(*branch*)
Cast a spell on the given branch, after all its children, grandchildren, have been processed, if *branchentry()* returned `True` on the given branch.

Typically, you will override this function to perform a particular operation on a block type, but you rely on the fact that the children must have been processed first.

Parameters **branch** (`GlobalNode`) – The branch to cast the spell on.

branchinspect(*branch*)
Inspect spells with *Spell.branchinspect()* (not all checks are executed, only keeps going until a spell inspection returns `True`).

property changed
bool(x) -> bool

Returns True when the argument x is true, False otherwise. The builtins True and False are the only two instances of the class bool. The class bool is a subclass of the class int, and cannot be subclassed.

dataentry()
Look into every spell with *Spell.dataentry()*.

dataexit()
Look into every spell with *Spell.dataexit()*.

class pyffi.spells.**SpellGroupSeriesBase**(*toaster=None*, *data=None*, *stream=None*)
Bases: *pyffi.spells.SpellGroupBase*

Base class for running spells in series.

branchentry(*branch*)
Cast the spell on the given branch. First called with branch equal to `data`'s children, then the grandchildren, and so on. The default implementation simply returns `True`.

Typically, you will override this function to perform an operation on a particular block type and/or to stop recursion at particular block types.

Parameters **branch** (`GlobalNode`) – The branch to cast the spell on.

Returns `True` if the children must be processed, `False` otherwise.

Return type `bool`

branchinspect(*branch*)
Like `_branchinspect()`, but for customization: can be overridden to perform an extra inspection (the default implementation always returns `True`).

Parameters **branch** (`GlobalNode`) – The branch to check.

Returns `True` if the branch must be processed, `False` otherwise.

Return type `bool`

property changed
bool(x) -> bool

Returns True when the argument x is true, False otherwise. The builtins True and False are the only two instances of the class bool. The class bool is a subclass of the class int, and cannot be subclassed.

dataentry()
Called before all blocks are recursed. The default implementation simply returns `True`. You can access the data via `data`, and unlike in the `datainspect()` method, the full file has been processed at this stage.

Typically, you will override this function to perform a global operation on the file data.

Returns `True` if the children must be processed, `False` otherwise.

Return type `bool`

dataexit()
Called after all blocks have been processed, if *dataentry()* returned `True`.

Typically, you will override this function to perform a final spell operation, such as writing back the file in a special way, or making a summary log.

recurse(*branch=None*)
Recurse spells in series.

Creating toaster scripts

To create a new toaster script, derive your toaster from the *Toaster* class, and set the *Toaster.FILEFORMAT* attribute of your toaster to the file format class of the files it can toast.

class pyffi.spells.**Toaster**(*spellclass=None*, *options=None*, *spellnames=None*, *logger=None*)
Bases: object

Toaster base class. Toasters run spells on large quantities of files. They load each file and pass the data structure to any number of spells.

ALIASDICT = {}
Dictionary with aliases for spells.

DEFAULT_OPTIONS = {'applypatch': False, 'archives': False, 'arg': '', 'createpatch
List of spell classes of the particular *Toaster* instance.

EXAMPLES = ''
Some examples which describe typical use of the toaster.

FILEFORMAT
alias of *pyffi.object_models.FileFormat*

SPELLS = []
List of all available *Spell* classes.

cli()
Command line interface: initializes spell classes and options from the command line, and run the *toast()* method.

exclude_types = []
Tuple of types corresponding to the exclude key of *options*.

get_toast_head_root_ext(*filename*)
Get the name of where the input file *filename* would be written to by the toaster: head, root, and extension.

Parameters **filename** (str) – The name of the hypothetical file to be toasted.

Returns The head, root, and extension of the destination, or (None, None, None) if --dry-run is specified.

Return type tuple of three strs

get_toast_stream(*filename*, *test_exists=False*)

Calls get_toast_head_root_ext(filename)() to determine the name of the toast file, and return a stream for writing accordingly.

Then return a stream where result can be written to, or in case test_exists is True, test if result would overwrite a file. More specifically, if test_exists is True, then no streams are created, and True is returned if the file already exists, and False is returned otherwise.

include_types = []

Tuple of types corresponding to the include key of *options*.

indent = 0

An int which describes the current indentation level for messages.

inspect_filename(*filename*)

Returns whether to toast a filename or not, based on skip_regexs and only_regexs.

is_admissible_branch_class(*branchtype*)

Helper function which checks whether a given branch type should have spells cast on it or not, based in exclude and include options.

```
>>> from pyffi.formats.nif import NifFormat
>>> class MyToaster(Toaster):
...     FILEFORMAT = NifFormat
>>> toaster = MyToaster() # no include or exclude: all admissible
>>> toaster.is_admissible_branch_class(NifFormat.NiProperty)
True
>>> toaster.is_admissible_branch_class(NifFormat.NiNode)
True
>>> toaster = MyToaster(options={"include": ["NiProperty", "NiNode"], "exclude
↪": ["NiMaterialProperty", "NiLODNode"]})
>>> toaster.is_admissible_branch_class(NifFormat.NiProperty)
True
>>> toaster.is_admissible_branch_class(NifFormat.NiNode)
True
>>> toaster.is_admissible_branch_class(NifFormat.NiAVObject)
False
>>> toaster.is_admissible_branch_class(NifFormat.NiLODNode)
False
>>> toaster.is_admissible_branch_class(NifFormat.NiSwitchNode)
True
>>> toaster.is_admissible_branch_class(NifFormat.NiMaterialProperty)
False
>>> toaster.is_admissible_branch_class(NifFormat.NiAlphaProperty)
True
```

logger = <Logger pyffi.toaster (WARNING)>

A logging.Logger for toaster log messages.

msg(*message*)

Write log message with logger.info(), taking into account *indent*.

Parameters **message** (str) – The message to write.

msgblockbegin(*message*)
Acts like *msg()*, but also increases *indent* after writing the message.

msgblockend(*message=None*)
Acts like *msg()*, but also decreases *indent* before writing the message, but if the message argument is None, then no message is printed.

only_regexs = []
Tuple of regular expressions corresponding to the only key of *options*.

options = {}
The options of the toaster, as dict.

static parse_inifile(*option*, *opt*, *value*, *parser*, *toaster=None*)
Initializes spell classes and options from an ini file.

skip_regexs = []
Tuple of regular expressions corresponding to the skip key of *options*.

spellnames = []
A list of the names of the spells.

toast(*top*)
Walk over all files in a directory tree and cast spells on every file.

Parameters **top** (*str*) – The directory or file to toast.

toast_archives(*top*)
Toast all files in all archives.

top = ''
Name of the top folder to toast.

write(*stream*, *data*)
Writes the data to data and raises an exception if the write fails, but restores file if fails on overwrite.

writepatch(*stream*, *data*)
Creates a binary patch for the updated file.

pyffi.object_models — File format description engines

Warning: The documentation of this module is very incomplete.

This module bundles all file format object models. An object model is a group of classes whose instances can hold the information contained in a file whose format is described in a particular way (xml, xsd, and possibly others).

class pyffi.object_models.**MetaFileFormat**
Bases: type

This metaclass is an abstract base class for transforming a file format description into classes which can be directly used to manipulate files in this format.

A file format is implemented as a particular class (a subclass of *FileFormat*) with class members corresponding to different (bit)struct types, enum types, basic types, and aliases.

static openfile(*filename*, *filepaths=None*, *encoding=None*)
Find *filename* in given *filepaths*, and open it. Raises IOError if file cannot be opened.

Parameters

- **filename** (str) – The file to open.
- **filepaths** (list of strs) – List of paths where to look for the file.

class pyffi.object_models.**FileFormat**

Bases: object

This class is the base class for all file formats. It implements a number of useful functions such as walking over directory trees (*walkData()*) and a default attribute naming function (*name_attribute()*). It also implements the base class for representing file data (*FileFormat.Data*).

ARCHIVE_CLASSES = []

Override this with a list of archive formats that may contain files of the format.

class Data

Bases: pyffi.utils.graph.GlobalNode

Base class for representing data in a particular format. Override this class to implement reading and writing.

inspect(*stream*)

Quickly checks whether the stream appears to contain data of a particular format. Resets stream to original position. Call this function if you simply wish to check that a file is of a particular format without having to parse it completely.

Override this method.

Parameters **stream** (*file*) – The file to inspect.

Returns True if stream is of particular format, False otherwise.

read(*stream*)

Read data of particular format from stream. Override this method.

Parameters **stream** (file) – The file to read from.

user_version = None

User version (additional version field) of the data.

version = None

Version of the data.

write(*stream*)

Write data of particular format to stream. Override this method.

Parameters **stream** (file) – The file to write to.

RE_FILENAME = None

Override this with a regular expression (the result of a re.compile call) for the file extension of the format you are implementing.

classmethod name_attribute(*name*)

Converts an attribute name, as in the description file, into a name usable by python.

Parameters **name** (str) – The attribute name.

Returns Reformatted attribute name, useable by python.

```
>>> FileFormat.name_attribute('tHis is A Silly naME')
't_his_is_a_silly_na_m_e'
>>> FileFormat.name_attribute('Test:Something')
'test_something'
>>> FileFormat.name_attribute('unknown?')
'unknown'
```

classmethod name_class(*name*)

Converts a class name, as in the xsd file, into a name usable by python.

Parameters **name** (*str*) – The class name.

Returns Reformatted class name, useable by python.

```
>>> FileFormat.name_class('this IS a sillyNAME')
'ThisIsASillyNAME'
```

classmethod name_parts(*name*)

Intelligently split a name into parts:

- first, split at non-alphanumeric characters
- next, seperate digits from characters
- finally, if some part has mixed case, it must be camel case so split it further at upper case characters

```
>>> FileFormat.name_parts("hello_world")
['hello', 'world']
>>> FileFormat.name_parts("HELLO_WORLD")
['HELLO', 'WORLD']
>>> FileFormat.name_parts("HelloWorld")
['Hello', 'World']
>>> FileFormat.name_parts("helloWorld")
['hello', 'World']
>>> FileFormat.name_parts("xs:NMTOKEN")
['xs', 'NMTOKEN']
>>> FileFormat.name_parts("xs:NCName")
['xs', 'N', 'C', 'Name']
>>> FileFormat.name_parts('this IS a sillyNAME')
['this', 'IS', 'a', 'silly', 'N', 'A', 'M', 'E']
>>> FileFormat.name_parts('tHis is A Silly naME')
['t', 'His', 'is', 'A', 'Silly', 'na', 'M', 'E']
```

static version_number(*version_str*)

Converts version string into an integer. This default implementation simply returns zero at all times, and works for formats that are not versioned.

Override for versioned formats.

Parameters **version_str** (`str`) – The version string.

Returns A version integer. A negative number denotes an invalid version.

classmethod walk(*top*, *topdown=True*, *mode='rb'*)

A generator which yields all files in directory top whose filename matches the regular expression *RE_FILENAME*. The argument top can also be a file instead of a directory. Errors coming from os.walk are ignored.

Note that the caller is not responsible for closing the stream.

This function is for instance used by *pyffi.spells* to implement modifying a file after reading and parsing.

Parameters

- **top** (`str`) – The top folder.
- **topdown** (`bool`) – Determines whether subdirectories should be iterated over first.
- **mode** (`str`) – The mode in which to open files.

classmethod walkData(*top*, *topdown=True*, *mode='rb'*)

A generator which yields the data of all files in directory top whose filename matches the regular expression

RE_FILENAME. The argument top can also be a file instead of a directory. Errors coming from os.walk are ignored.

Note that the caller is not responsible for closing the stream.

This function is for instance used by *pyffi.spells* to implement modifying a file after reading and parsing.

Parameters

- **top** (str) – The top folder.
- **topdown** (bool) – Determines whether subdirectories should be iterated over first.
- **mode** (str) – The mode in which to open files.

1.7.4 How to contribute

Do you want to fix a bug, improve documentation, or add a new feature?

Get git/msysgit

If you are on windows, you need msysgit. If you are already familiar with subversion, then, in a nutshell, msysgit is for git what TortoiseSVN is for subversion. The main difference is that msysgit is a command line based tool.

For more information about git and github, the github help site is a great start.

Track the source

If you simply want to keep track of the latest source code, start a shell (or, the Git Bash on windows), and type (this is like "svn checkout"):

```
git clone git://github.com/niftools/pyffi.git
```

To synchronize your code, type (this is like "svn update"):

```
git pull
```

Development

Create a fork

- Get a github account.
- Log in on github and fork PyFFI (yes! merging with git is easy so forking is encouraged!).

Use the source

PyFFI is entirely written in pure Python, hence the source code runs as such on any system that runs Python. Edit the code with your favorite editor, and install your version of PyFFI into your Python tree to enable your PyFFI to be used by other applications such as for instance QSkope, or the Blender NIF Scripts. From within your PyFFI git checkout:

```
C:\Python25\python.exe setup.py install
```

or on linux:

```
python setup.py install
```

To build the documentation:

```
cd docs
make html -a
```

PyFFI has an extensive test suite, which you can run via:

```
python rundoctest.py
```

The Blender NIF Scripts test suite provides additional testing for PyFFI. From within your niftools/blender checkout:

```
./install.sh
blender -P runtest_xxx.py
```

To build the source packages and the Windows installer (on a linux system which has both wine and nsis installed):

```
makezip.sh
```

Submit your updates

Simply do a pull request if you want your fork to be merged, and your contributions may be included in the next release!

1.7.5 Authors

Main author

- Amorilia (amorilia@users.sourceforge.net)

Previous Developers

- wz (wz_@users.sourceforge.net)
  - niftester (the current spells module is a revamped version of that)
  - nifvis (which hopefully will be embedded into QSkope one day. . .)
- taarna23 (taarna23@users.sourceforge.net)
  - extraction of DXT compressed embedded textures for the nif format
- tazpn (tazpn@users.sourceforge.net)

  - mopp generator using the Havok SDK
  - suggestions for improving the spells module
- Scanti
  - EGM & TRI format info
- Carver13
  - EGM & TRI format xml

Contributors

- PacificMorrowind (pacmorrowind@sourceforge.net)
  - some nif/kf modifying spells
- Ulrim/Arthmoor
  - optimization kit
- seith (seith@users.sourceforge.net)
  - logo design for the Windows installer
- MorCroft
  - Patch for BSSTextureSet texture path substitution

1.7.6 License

1.7.7 ChangeLog

Release 2.2.3 (Mar 17, 2014)

- Update spell texture_path_substitution for BSTextureSet blocks (fix contributed by MorCroft)
- Updated to latest nif.xml, submodules moved to github.

Release 2.2.2 (Nov 17, 2012)

- Use VERSION file.
- Fix dump_python for correct default values.
- Fix xml for Fallout 3.

Release 2.2.1 (Nov 11, 2012)

- Added Skyrim version detection.
- The check_readwrite spell now handles file size differences due to empty strings.
- Bugfix in nif write method when entities are None (see Skyrim meshes/actors/character/character assets/hair/hairlonghumanm.nif).
- Various fixes for Fallout 3 and Fallout NV nifs.
- New dump_python spell, to generate python code that recreates a nif file.
- Accept string palette strings that do not have a null character preceeding it (reported by Koniption).
- New modify_getbonepriorities and modify_setbonepriorities spells, which work similar to the kfupdater.
- New fix_fallout3stringoffsets spell to 'fix' empty offsets in Oblivion style kf files, if they are to be used in Fallout 3.
- Installer no longer targets Maya and Blender.

Release 2.2.0 (Nov 26, 2011)

- Added PSK and PSA file support (used by Unreal engine).
- A py3k fix (contributed by infectedsoundsystem).
- Updated installer for Blender 2.5x+.

Release 2.1.11 (Nov 26, 2011)

- Explicitly use wine for running mopper on non-win32 platforms (fixes issue on Arch Linux, reported by ifss000f, see issue #3423990).
- Removed skip list from extra fix_texturepath stage in Oblivion optimization kit.
- Various optimizations (contributed by infectedsoundsystem). The optimizer spell now runs a fair bit faster.
- Garbage collection call after each spell has been removed as profiling showed that a lot of time was spent on it. You can still force the old (slow) behaviour by using the new –gccollect command line option or adding "gccollect = True" in your ini file.
- Encoding fix for xml and xsd parsing.
- Merge duplicates after optimizing geometry to work around de-duplication during geometry optimization phase (fixes issue #3425637, reported by chacky2).
- Removed xsd object model and dae format (wasn't functional yet anyway); deferred to py3k.

Release 2.1.10 (Oct 10, 2011)

- Fixed bspline data methods to handle invalid kfs with missing basis data (reported by K'Aviash).
- Fixed mass, center, inertia methods to deal with cases where shape is missing (reported by rlibiez, see niftools issue #3248754).
- Fixed center calculation of bhkListShape collisions, and fixed zero division error when creating very small collision shapes (reported by Koniption, see issues #3334577 and #3308638).
- Fixed shortcut to uninstaller in programs folder (reported by neomonkeus, see niftools issue #3397540).
- Fixed geometry optimizer to handle cases where number of morph vertices does not match number of shape vertices (reported by rlibiez, see issue #3395484).
- Merged ulrim's optimization kit, along with arthmoor's improved ini files.
- Integrated far nif optimization with the general purpose optimize spell (requested by Gratis_monsta, see issue #3177316).
- New shell_optimize.ini for configuring optimization as executed from Windows shell.
- Allow .ini files that do not have a [main] or [options] section.
- Fix Windows shell integration to point to the new shell_optimize.ini file (reported by rlibiez, see issue #3415490).
- Fixed zombie process problem on Windows when a toaster was running with multiple jobs (reported by Alphanos, see issue #3390826).

Release 2.1.9 (Mar 26, 2011)

- Improved documentation of .dir/.img unpack and pack scripts.
- Bugfix in .dir format, so it can now also handle single record images.
- Added new script to make and apply patches (functionality is identical to and OnmyojiOmn's and jonwd7's pyffi patcher scripts, but it is written in Python to work on all platforms).
- New fix_emptyskeletonroots spell (automatically included in the optimize spell) to fix issues with nifs that do not have their NiSkinInstance skeleton root set (fixes #3174085, reported by xjdhdr).

- Fixed logging issue on Windows platform with multithreading enabled (fixes issue #3174339, reported by xjdhdr).
- Fixed QSkope shortcut issue when path contains spaces (reported by Brumbek).
- Added support for BSPackedAdditionalGeometryData (reported by Ghostwalker71, niftools issue #3177847).
- Skip terminal chars in mopper output (fixes issues with running mopper under wine).
- Bugfix in patch_recursive_make/apply scripts for "deep" folder layouts (fixes issue #3193914, reported by xjdhdr).
- Do not pack collisions in OL_CLUTTER (fixes issue #3194017 reported by Gratis_monsta).
- Fixed issue with skipping terminal chars in mopper output on Windows platform (fixes issue #3205569, reported and diagnosed by ulrim).
- Updates for Bully SE format (fixes issue reported by Tosyk).
- Bully SE nif header reading fix for BBonusB.nft.
- Added initial support for Epic Mickey (reported and test files provided by Tosyk).
- Bugfix for NiMesh read and write.
- Updated dump_pixeldata spell to enable it to export Bully SE's nft files.
- Disabled copy in patch_recursive_apply script (see issue #3219744, suggested by ulrim).
- Pass additional arguments of patch_recursive_apply/make to the patch command (see issue #3219744, suggested by ulrim).
- Fix nif optimizer in case it contains tangent space data but no uv data (see issue #3218751, reported by krimhorn).
- Handle removal of redundant triangles when updating dismember skin partitions (see issue #3218751, reported by krimhorn).
- Fix vertex cache optimizer to handle more meshes with more than 255 triangles per vertex (see issue #3218751, reported by krimhorn).
- Skipping meshes that have NiAdditionalGeometryData (until we understand what this data does and how to optimize it).
- Sane default settings for bhkRigidBody unknowns to ensure that constraints behave properly (contributed by Koniption).
- Added ini file to unpack Bully SE .nft files.

Release 2.1.8 (Feb 4, 2011)

- Quickhull bugfix for precision argument in degenerate cases (issue #3163949, fix contributed by liuhuanjim013).
- Fixed issue with detecting box shapes on degenerate collision meshes (fixes issue #3145104, reported by Gratis_monsta).
- Ensure that enum has valid default value.
- Added CStreamableAssetData for Divinity 2 (reported by pertinen, niftools issue #3164929).
- NiPixelData.save_as_dds fourcc flag bugfix.
- Added Rockstar .dir format (used in Bully SE).
- Added Rockstar .dir/.img unpack and pack scripts (only tested for Bully SE).

Release 2.1.7 (Jan 23, 2011)

- Added support for Fallout New Vegas (contributed by throttlekitty and saiden).
- Updated geometry optimizer to keep dismember body parts, for Fallout 3 and Fallout New Vegas (fixes issue #3025691 reported by Chaky).
- Added flag to enable debugging in vertex cache algorithm, to assess how suboptimal the solution is on any particular mesh (testing reveals that it finds the globally optimal solution in more than 99% of all iterations, for typical meshes).
- New check_triangles_atvr spell to find optimal parameters for vertex cache algorithm by simulated annealing.
- Fixed send_geometries_to_bind_position, send_detached_geometries_to_node_position, and send_bones_to_bind_position in case skin instance has empty bone references (fixes issue #3114079, reported by drakonnen).
- Fix for verbose option in multithread mode (reported by Gratis_monsta).
- Fix optimize spell when no vertices are left after removing duplicate vertices and degenerate triangles (reported by Gratis_monsta).
- Fixed tangent space issue along uv seams (reported by Gratis_monsta and Tommy_H, see issue #3120585).
- Log an error instead of raising an exception on invalid enum values (fixes issue #3127161, reported by rlibiez).
- Disabled 2to3 in Windows installer; the Python 3 version of PyFFI will be released separately.

Release 2.1.6 (Nov 13, 2010)

- The optimize spell now includes two new spells: opt_collisiongeometry for optimizing triangle based collisions, and opt_collisionbox for optimizing simple box collisions. This should result in faster loads and probably also a small but noticable performance improvement.
- Fixed opt_collisiongeometry for multimaterial mopps (reported by wildcard_25, see issue #3058096).
- New SpellCleanFarNif spell (suggested by wildcard_25, see issue #3021629).
- Bad normals are now ignored when packing a bhkNiTriStripsShape (fixes issue #3060025, reported by rlibiez).
- The opt_delunusedbones spell no longer removes bones if they have a collision object (fixes issue #3064083, reported by wildcard_25).
- If the jobs option is not specified in the toaster, then the number of processors is used—requires Python 2.6 or higher (suggested by chaky2, see issue #3052715, implements issue #3065503).
- New opt_delzeroscale spell to delete branches with zero scale (suggested by chaky2, see issue #3013004).
- The opt_mergeduplicates spell now ignores (non-special) material names, so identical materials with different names will get merged as well (suggested by chaky2, see issue #3013004).
- New spell to fix subshape counts (see issue #3060025, reported by rlibiez), it is included in the optimize spell.
- New opt_collisionbox spell which automatically converts triangle based box collisions to primitive box collisions, which are much faster in-game (contributed by PacificMorrowind).
- Optimizer spell now uses triangles to represent skin partitions (improves in-game fps).
- Better vertex map calculation when calculating skin partitions (improves in-game fps).
- Optimizer now always triangulates (improves in-game fps). Stripification will be readded later in a modularized version of the optimizer spell, for those that want minimal file size rather than maximal performance).
- Much faster implementation of vertex cache algorithm (now runs in linear time instead of quadratic time).

- Check triangle count when converting to box shape (fixes issue #3091150).
- Bugfix in vertex map reordering (fixes most nifs reported in issue #3071616).
- Bugfix in vertex cache algorithm (fixes a nif reported in issue #3071616).
- Cover degenerate case in ATVR calculation when there are no vertices (fixes a nif reported in issue #3071616).
- Cover degenerate case when calculating cache optimized vertex map (fixes a nif reported in issue #3071616).
- Remove branches if they have no triangles (again fixes a nif reported in issue #3071616).

Release 2.1.5 (Jul 18, 2010)

- Improved interface for TRI files, and a bugfix in TRI file writing.
- Added EGT file support.
- The fix_texturepath spell now also converts double backslash in single backslash (suggested by Baphometal).
- Bugfix in save_as_dds function for newer NiPixelData blocks (reported by norocelmiau, issue #2996800).
- Added support for Laxe Lore nifs (reported by bobsobol, issue #2995866).
- New spells:
  - opt_collisiongeometry: to optimize collision geometry in nifs (contributed by PacificMorrowind).
  - check_materialemissivevalue: checks (and warns) about high values in material emissive settings (contributed by PacificMorrowind).
  - modify_mirroranimation: mirrors an animation (specifically left to right and vice versa) - use it to for example turn a right hand punch anim into a left hand punch anim (contributed by PacificMorrowind).
- Added big-endian support.
- Removed `**kwargs` argument passing for faster and more transparant implementation (reading and writing is now about 8% faster).
- Do not merge BSShaderProperty blocks (reported by Chaky, niftools issue #3009832).
- Installer now recognizes Maya 2011.
- Fixed NiPSysData read and write for Fallout 3 (reported by Chaky, niftools issue #3010861).

Release 2.1.4 (Mar 19, 2010)

- Extra names in oblivion_optimize.ini skip list for known mods (contributed by Tommy_H).
- New spells
  - modify_collisiontomopp
  - opt_reducegeometry
  - opt_packcollision
- Windows right-click optimize method now uses default.ini and oblivion_optimize.ini.
- fix_texturepath now fixes paths that include the whole drive path to just the textures/... path.
- The optimize spell has been fixed to update Fallout 3 style tangent space (fixes issue #2941568, reported by xjdhdr).

Release 2.1.3 (Feb 20, 2010)

- Added toaster option to process files in archives (not yet functional).
- Added toaster option to resume, by skipping existing files in the destination folder.
- Toaster now removes incompletely written files on CTRL-C (to avoid corrupted files).
- Fixed makefarnif spell (now no longer deletes vertex colors).
- New spells
  - fix_delunusedroots
  - modify_bonepriorities
  - modify_interpolatortransrotscale
  - modify_delinterpolatortransformdata
  - opt_delunusedbones
- The niftoaster optimize spell now also includes fix_delunusedroots.
- Removed unused pep8 attribute conversion code.
- Toasters can now be configured from an ini file.
- bhkMalleableHinge update_a_b bugfix (reported by Ghostwalker71).

Release 2.1.2 (Jan 16, 2010)

- Fallout 3 skin partition flag bugfix (reported by Ghostwalker71).
- Fixed bug in optimize spell, when has_vertex_colors was False but vertex color array was present (reported by Baphometal, debugged by PacificMorrowind).
- Initial bsa file support (Morrowind, Oblivion, and Fallout 3).

Release 2.1.1 (Jan 11, 2010)

- Accidently released corrupted nif.xml (affected Fallout 3), so this is just a quick bugfix release including the correct nif.xml.

Release 2.1.0 (Jan 10, 2010)

- Improved windows installer.
- Compatibility fix for Python 2.5 users (reported by mac1415).
- Renamed some internal modules for pep8 compliance.
- All classes and attributes are now in pep8 style. For compatibility, camelCase attributes are generated too (however this will be dropped for py3k).
- Renamed a few niftoaster spells.
  - fix_strip -> modify_delbranches
  - fix_disableparallax -> modify_disableparallax
- New niftoaster spells.

  - fix_cleanstringpalette: removes unused strings from string palette.
  - modify_substitutestringpalette: regular expression substitution of strings in a string palette.
  - modify_scaleanimationtime: numeric scaling of animations.
  - modify_reverseanimation: reverses an animation (ie useful for making only an open animation and then running this to get a close animation).
  - modify_collisionmaterial: sets any collision materials in a nif to specified type.
  - modify_delskinshapes: Delete any geometries with a material name of 'skin'
  - modify_texturepathlowres: Changes the texture path by replacing 'textures/*' with 'textures/lowres/*'. used mainly for making _far.nifs.
  - modify_addstencilprop: Adds a NiStencilProperty to each geometry if it is not present.
  - modify_substitutetexturepath: regular expression substitution of a texture path.
  - modify_makeskinlessnif: Spell to make fleshless CMR (Custom Model Races) clothing/armour type nifs. (runs modify_delskinshapes and modify_addstencilprop)
  - modify_makefarnif: Spell to make _far type nifs.
- Bugfix for niftoaster dump spell.
- New –suffix option for toaster (similar to the already existing –prefix option).
- New –skip and –only toaster options to toast files by regular expression.
- New –jobs toaster option which enables multithreaded toasting.
- New –source-dir and –dest-dir options to save toasted nifs in a given destination folder.
- Added workaround for memory leaks (at the moment requires –jobs >= 2 to be functional).
- The niftoaster opt_geometry spell now always skips NIF files when a similarly named tri or egm file is found.
- Added support for Atlantica nifs.
- Added support for Joymaster Interactive Howling Sword nifs.

Release 2.0.5 (Nov 23, 2009)

- Added regression test and fixed rare bug in stripification (reported by PacificMorrowind, see issue #2889048).
- Improved strip stitching algorithm: *much* more efficient, and now rarely needs more than 2 stitches per strip.
- Improved stripifier algorithm: runs about 30% faster, and usually yields slightly better strips.
- Added new modify_texturepath and modify_collisiontype niftoaster spells (contributed by PacificMorrowind).
- Various fixes and improvements for 20.5.0.0+ nifs.
- Check endian type when processing nifs.
- Source release now includes missing egm.xml and tri.xml files (reported by skomut, fixes issue #2902125).

Release 2.0.4 (Nov 10, 2009)

- Write NaN on float overflow.
- Use pytristrip if it is installed.
- Implemented the FaceGen egm (done) and tri (in progress) file formats with help of Scanti and Carver13.
- The nif dump_pixeldata spell now also dumps NiPersistentSrcTextureRenderData (reported by lusht).
- Set TSpace flags 16 to signal presence of tangent space data (fixes Fallout 3 issue, reported by Miaximus).

Release 2.0.3 (Sep 28, 2009)

- Various bugfixes for the Aion cgf format.
- Updates for nif.xml to support more recent nif versions (20.5.0.0, 20.6.0.0, and 30.0.0.2).

Release 2.0.2 (Aug 12, 2009)

- The source has been updated to be Python 3.x compatible via 2to3.
- New unified installer which works for all versions of Python and Maya at once (at the moment: 2.5, 2.6, 3.0, 3.1) and also for all versions of Maya that use Python 2.5 and 2.6 (2008, 2009, and 2010, including the 64 bit variants).
- Added support for Aion cgf files.
- Added support for NeoSteam header and footer.
- Log warning rather than raising exception on invalid links (fixes issue #2818403 reported by abubakr125).
- Optimizer can now recover from invalid indices in strips (this fixes some nifs mentioned in issue #2795837 by baphometal).
- Skin updater can now recover when some vertices have no weights (this fixes some nifs mentioned in issue #2795837 by baphometal).
- Skip zero weights and add up weights of duplicated bones when calculating vertex weights (this fixes some nifs mentioned in issue #2795837 by baphometal).
- The nif optimizer can now handle NiTriShapeData attached as a NiTriStrips data block (fixes some corrupt nifs provided by baphometal in issue #2795837).
- Optimizer can now recover from NaN values in geometry (sample nifs provided by baphometal).
- Do not attempt to optimize nifs with an insane amount of triangles, but put out a warning instead.
- Log error rather than raising exception when end of NIF file is not reached (fixes issue with sample nif provided by baphometal).

Release 2.0.1 (Jul 22, 2009)

- Added Windows installer for Python 2.6.
- Updated mopper.exe compiled with msvc 2008 sp1 (fixes issue #2802413, reported by pacmorrowind).
- Added pdb session to track cicular references and memory leaks (see issues #2787602 and #2795837 reported by alexkapi12 and xfrancis147).
- Added valgrind script to check memory usage, and to allow keeping track of it between releases (see issues #2787602 and #2795837 reported by alexkapi12 and xfrancis147).
- Removed parenting in xml model from everywhere except Array, and using weakrefs to avoid circular references, which helps with garbage collection. Performance should now be slightly improved.
- Updates to xml object model expression syntax.
  - Support for field qualifier '.'.
  - Support for addition '+'.
- Updates to Targa format.
  - Support for RLE compressed Targa files (test file contributed by Alphax, see issue #2790494).
  - Read Targa footer, if present (test file contributed by Alphax, see issue #2790494).
  - Improved interface: header, image, and footer are now global nodes.
- Updates to xsd object model.
  - Classes and attributes for Collada format are now generated (but not yet functional).

Release 2.0.0 (May 4, 2009)

- Windows installer now detects Maya 2008 and Maya 2009, and their 64 bit variants, and can install itself into every Maya version that is found.
- Updates to the XML object model (affects CGF, DDS, KFM, NIF, and TGA).
  - Class customizers are taken immediately from the format class, and not from separate modules — all code from customization modules has been moved into the main format classes. The result is that parsing is faster by about 50 percent.
  - clsFilePath removed, as it is no longer used.
- Updates and fixes for the KFM format.
  - The Data element inherits from Header, and Header includes also all animations, so it is more straightforward to edit files.
  - The KFM files open again in QSkope.
- Updates for the CGF format.
  - CHUNK_MAP no longer constructed in Data.__init__ but in a metaclass.
  - Deprecated functions in CgfFormat have been removed.
- Updates for the NIF format.
  - Synced nif.xml with nifskope's xml (includes fixes for Lazeska).
  - Removed deprecated scripts (niftexdump, nifdump, ffvt3rskinpartition, nifoptimize).

  - Fixed scaling bug on nifs whose tree has duplicate nodes. Scaling now no longer works recursively, unless you use the scaling spell which handles the duplication correctly.

- Updated module names to follow pep8 naming conventions: all modules have lower case names.

Release 1.2.4 (Apr 21, 2009)

- Documentation is being converted to Sphinx. Currently some parts of the documentation are slightly broken with epydoc. Hopefully the migration will be complete in a month or so, resolving this issue.
- removed deprecated PyFFI.Spells code:
  - old style spells no longer supported
  - almost all old spells have been converted to the new spell system (the few remaining ones will be ported for the next release)
- nif:
  - nif optimizer can be run on folders from the windows context menu (right-click on any folder containing nifs and select "Optimize with PyFFI")
  - synced nif.xml with upstream (adds support for Worldshift, bug fixes)
  - using weak references for Ptr type (this aids garbage collection)
  - added fix_strip niftoaster spell which can remove branches selectively (feature request #2164309)
  - new getTangentSpace function for NiTriBasedGeom (works for both Oblivion and Fallout 3 style tangent spaces)
  - improved mergeSkeletonRoots function (will also merge roots of skins that have no bones in common, this helps a lot with Morrowind imports)
  - new sendDetachedGeometriesToNodePosition function and spell (helps a lot with Morrowind imports)
- tga:
  - added support for color map and image data in the xml
  - uses the new data model
  - works again in QSkope
- xml object model:
  - added support for multiplication and division operators in expressions
- fixes for unicode support (prepares for py3k)

Release 1.2.3 (Apr 2, 2009)

- removed reduce() calls (py3k compatibility)
- started converting print calls (py3k compatibility)
- removed relative imports (py3k compatibility)
- removed BSDiff module (not useful, very slow, use external bsdiff instead)
- nif:
  - fixed the update mopp spell for fallout 3 nifs
  - fixed addShape in bhkPackedNiTriStripsShape for fallout 3 nifs

  - niftoaster sends to stdout instead of stderr so output can be captured (reported by razorwing)

Release 1.2.2 (Feb 15, 2009)

- cgf format:
  - fixed various regression bugs that prevented qskope to run on cgf files
  - updated to use the new data system

Release 1.2.1 (Feb 2, 2009)

- nif format:
  - new addIntegerExtraData function for NiObjectNET

Release 1.2.0 (Jan 25, 2009)

- installer directs to Python 2.5.4 if not installed
- using logging module for log messages
- nif format:
  - swapping tangents and binormals in xml; renaming binormals to bitangents (see http://www.terathon.com/code/tangent.html)
  - updates for Fallout 3 format
  - updated skin partition algorithm to work for Fallout 3
    - new triangles argument
    - new facemap argument to pre-define partitions (they will be split further if needed to meet constraints)
    - sort vertex weight list by weight in skin partitions (except if padbones is true; then sorted by bone index, to keep compatibility with ffvt3r)
    - option to maximize bone sharing
  - mopps take material indices into account and compute welding info (attempt to fix mopp multi-material issues, does not yet seem to work though)
  - support for niftools bitflags by converting it to a bitstruct on the fly
  - better algorithm for sending bones to bind position, including spells for automating this function over a large number of nifs
  - disable fast inverse in bind pos functions to increase numerical precision
  - new algorithm to sync geometry bind poses, along with spell (this fixes many issues with Morrowind imports and a few issues with Fallout 3 imports)
  - more doctests for various functions
  - a few more matrix functions (supNorm, substraction)
- dds format:
  - updated to use the FileFormat.Data method (old inconvenient method removed)
- qskope:

  - refactored the tree model
  - all parenting functions are delegated to seperate DetailTree and GlobalTree classes
  - the DetailNode and GlobalNode classes only implement the minimal functions to calculate the hierarchy, but no longer host the more advanced hierarchy functions and data (this will save memory and speed up regular use of pyffi outside qskope)
  - EdgeFilter for generic edge type filtering; this is now a parameter for every method that needs to list child nodes

Release 1.1.0 (Nov 18, 2008)

- nif format:
  - a large number of functions have moved from the optimizer spell to to the main interface, so they can be easily used in other scripts without having to import this spell module (getInterchangeableTriShape, getInterchangeableTriStrips, isInterchangeable)
  - new convenience functions in NiObjectNET, NiAVObject, and NiNode (setExtraDatas, setProperties, setEffects, setChildren, etc.)
  - updates for Fallout 3
- niftoaster
  - new fix_addtangentspace spell to add missing tangent space blocks
  - new fix_deltangentspace spell to remove tangent space blocks
  - new fix_texturepath spell to change / into and to fix corrupted newline characters (which sometimes resulted from older versions of nifskope) in NiSourceTexture file paths
  - new fix_clampmaterialalpha spell
  - new fix_detachhavoktristripsdata spell
  - the ffvt3r skin partition spell is now fix_ffvt3rskinpartition
  - new opt_cleanreflists spell
  - new opt_mergeduplicates spell
  - new opt_geometry spell
  - the optimize spell is now simply implemented as a combination of other spells
- new internal implementation of bsdiff algorithm
- removed cry dae filter (an improved version of this filter is now bundled with ColladaCGF)
- reorganization of file format description code
  - all generic format description specific code has been moved to the PyFFI.ObjectModels.FileFormat module
  - all xml/xsd description specific code has been moved to the PyFFI.ObjectModels.XML/XSD.FileFormat modules
  - new NifFormat.Data class which now implements all the NIF file read and write functions
- completely revamped spell system, which makes it much easier to customize spells, and also enables more efficient implementations (thanks to tazpn for some useful suggestions, see issue #2122196)
  - toaster can call multiple spells at once

  - toaster takes spell classes instead of modules
  - for backwards compatibility, there is a class factory which turns any old spell module into a new spell class (the Toaster class will automatically convert any modules that it finds in its list of spells, so you do not need to be worried about call the factory explicitly)
  - early inspection of the header is possible, to avoid having to read all of the file if no blocks of interest are present
  - possibility to prevent the spell to cast itself on particular branches (mostly useful to speed up the spell casting process)
  - spells have callbacks for global initialization and finalization of data within the toaster
  - possibility to write output to a log file instead of to sys.stdout
  - better messaging system (auto indentation, list nif tree as spell runs)
  - support for spell hierarchies and spell grouping, in parallel or in series or any combination of these
- replaced ad hoc class customization with partial classes (still wip converting all the classes)
- xml object model expression parser
  - implemented not operator
  - expressions can combine multiple operators (only use this if the result is independent of the order in which these operators are applied)
  - new < and > operators
  - support for vercond attribute for Fallout 3
- started on a new object model based on an ANTLR parser of a grammar aimed at file format descriptions; this parser will eventually yield a more streamlined, more optimized, and more customizable version of the current xml object model (this is not yet bundled with the release, initial code is on svn)

Release 1.0.5 (Sep 27, 2008)

- niftoaster optimize
  - fix for materials named skin, envmap2, etc. (issue #2121098)
  - fix for empty source textures in texdesc (issue #2118481)
- niftoaster
  - new spell to disable parallax (issue #2121283)
- toaster
  - new options –diff and –patch to create and apply patches; interal patcher uses bsdiff format, but you can also specify an arbitrary external diff/patch command via –diff-cmd and –patch-cmd options (the external command must take three arguments: oldfile, newfile, and patchfile); note that this is still in experimental stage, not ready for production use yet

Release 1.0.4 (Sep 18, 2008)

- niftoaster optimize
  - morph data optimization (issue #2116594, fixes "bow" weapons)

Release 1.0.3 (Sep 17, 2008)

- niftoaster optimize
  - detach NiTriStripsData from havok tree when block is shared with geometry data (fixes issue #2065018, MiddleWolfRug01.NIF)
  - fix in case merged properties had controllers (issue #2106668)
- fix writing of block order: bhkConstraint entities now always preceed the constraint block (this also fixes the "falling sign" issue with the niftoaster optimize spell, issue #2068090)

Release 1.0.2 (Sep 15, 2008)

- "negative mass" fix in inertia calculation

Release 1.0.1 (Sep 12, 2008)

- small fix in uninstaller (didn't remove crydaefilter script)
- crydaefilter converts %20 back into spaces (as rc doesn't recognize %20)
- bugfixes for niftoaster optimize spell (pyffi issue #2065018)

Release 1.0.0 (Jul 24, 2008)

- new NSIS installer (this solves various issues with Vista, and also allows the documentation to be bundled)
- new filter to prepare collada (.dae) files for CryEngine2 resource compiler
  - wraps scenes into CryExportNodes
  - corrects id/sid naming
  - fixes init_from image paths
  - adds phong and lamber shader sid's
  - enforces material instance symbol to coincide with target
  - sets material names in correct format for material library and physicalization
- started on support for collada format, by parsing the collada xsd schema description (this is still far from functional, but an initial parser is already included with the library, although it does not yet create any classes yet)
- fully optimal mopp generation for Oblivion (using the NifTools mopper.exe which is a command line utility that calls the mopp functions in the havok library, credit for writing the original wrapper goes to tazpn)
- minor updates to the nif.xml format description
- refactoring: library reorganized and some interfaces have been unified, also a lot of code duplication has been reduced; see README.TXT for more details on how to migrate from 0.x.x to 1.x.x
  - main format classes PyFFI.XXX have been moved to PyFFI.Formats.XXX

  - "XxxFormat.getVersion(cls, stream)" now always returns two integers, version and user_version
  - "XxxFormat.read(self, stream, version, user_version, ...)" for all formats
  - "XxxFormat.write(self, stream, version, user_version, *readresult, ...)" for all formats
  - in particular, CGF format game argument removed from read and write functions, but there are new CgfFormat.getGame and CgfFormat.getGameVersion functions to convert between (version, user_version) and game
  - also for the CGF format, take care that getVersion no longer returns the file type. It is returned with the CgfFormat.read function, however there is a new CgfFormat.getFileType function, if you need to know the file type but you don't want to parse the whole file
  - all XxxFormat classes derive from XmlFileFormat base class
  - common nameAttribute, walk, and walkFile functions
  - XxxTester modules have been moved to PyFFI.Spells.XXX, along with a much improved PyFFI.Spells module for toasters with loads of new options
  - some other internal code has been moved around
    * qskopelib -> PyFFI.QSkope
    * PyFFI.Bases -> PyFFI.ObjectModels.XML
  - a lot more internal code reorganization is in progress...
- much documentation has been added and improved

Release 0.11.0 (Jun 16, 2008)

- nif:
  - fixed updateTangentSpace for nifs with zero normals
- cfg:
  - a lot of new physics stuff: MeshPhysicsDataChunk mostly decoded (finally!!)
  - fixes for reading and writing caf files (they are missing controller headers)
  - activated BoneMeshChunk and BoneInitialPosChunk for Crysis
- tga:
  - improved tga file detection heuristic

Release 0.10.10 (Jun 8, 2008)

- nif:
  - minor updates in xml
  - NiPixelData saveAsDDS function now also writes DXT compressed formats, that is, pixel formats 4, 5, and 6 (contributed by taarna23)
  - fixed nifoptimize for nifs with particle systems (niftools issue #1965936)
  - fixed nifoptimize for nifs with invalid normals (niftools issue #1987506)

Release 0.10.9 (May 27, 2008)

- nif:
  - bspline interpolator fix if no keys
  - fixed bspline scale bug

Release 0.10.8 (Apr 13, 2008)

- cgf:
  - more decoded of the mesh physics data chunk
- nif:
  - scaling for constraints
  - ported the A -> B spell from nifskope (see the new getTransformAB and updateAB methods)

Release 0.10.7 (Apr 5, 2008)

- cgf:
  - indices are unsigned shorts now (fixes geometry corruption on import of large models)
  - MeshChunk.setGeometry gives useful error message if number of vertices is too large
- nif:
  - nif.xml has minor updates in naming
  - added NiBSplineData access functions (experimental, interface could still change)
  - started on support for compressed B-spline data
  - fixed block order writing of bhkConstraints

Release 0.10.6 (Mar 30, 2008)

- tga: added missing xml file
- nif:
  - removed some question marks so the fields can be accessed easily in python interface
  - ControllerLink and StringPalette functions and doctests
  - quaternion functions in Matrix33 and Matrix44
  - new bspline modules (still to implement later)
  - fixed NiTransformInterpolator scaling bug
- cgf:
  - use tempfile for write test
- quick install batch file for windows

Release 0.10.5 (Mar 27, 2008)

- qskope: make bitstructs editable
- cgf:
  - MeshChunk functions to get vertex colors (game independent).
  - Set vertex colors in setGeometry function.

Release 0.10.4 (Mar 26, 2008)

- cgf:
  - fixed tangent space doctest
  - setGeometry argument sanity checking
  - setGeometry fix for empty material list
  - setGeometry tangent space update fix if there are no uvs

Release 0.10.3 (Mar 24, 2008)

- added support for the TGA format
- tangentspace:
  - validate normals before calculating tangents
  - added new option to get orientation of tangent space relative to texture space (Crysis needs to know about this)
- installer detects Maya 2008 and copies relevant files to Maya Python directory for the Maya scripts to work
- cgf:
  - tangent space cgftoaster
  - new MeshChunk updateTangentSpace function

Release 0.10.2 (Mar 22, 2008)

- cgf:
  - fixed “normals” problem by setting last component of tangents to -1.0
  - meshchunk function to get all material indices, per triangle (game independent)
  - scaling fixes for datastreamchunk, meshchunk, and meshsubsetschunk
  - fixed version of BreakablePhysicsChunk
  - a few new findings in decoding the physics data (position and rotation)

Release 0.10.1 (Mar 21, 2008)

- cgf:
  - some minor xml updates
  - setGeometry function for MeshChunk to set geometry for both Far Cry and Crysis in a unified way
  - uv.v opengl flip fix for Crysis MeshChunk data
- MathUtils: new function to calculate bounding box, center, and radius
- qskope: fixed bug which prevented setting material physics type to NONE

Release 0.10.0 (Mar 8, 2008)

- cgf: ported A LOT of stuff from the Crysis Mod SDK 1.2; the most common CE2 chunks now read and write successfully

Release 0.9.3 (Mar 7, 2008)

- cgf:
  - decoded a lot of geometry data
    - vertices
    - normals
    - vertex colors
    - uvs
    - mesh material info
  - started decoding many other chunk types
  - added chr chunk types so files containing them can be processed (the data is ignored)
  - started adding functions to MeshChunk to have unified access to geometry data for both Far Cry and Crysis cgf files
- windows installer registers chr extension with qskope

Release 0.9.2 (Feb 26, 2008)

- full support for the xml enum tag type, with improved editor in qskope
- new common string types (shared between cgf and nif formats)
  - null terminated
  - fixed sized
  - variable sized starting with integer describing length
- qskope: no more duplicate ptr refs in global view
- qskope: refactored delegate editor system to be more transparent and much easier to extend
- cgf: crysis chunks have been partially decoded (still very much wip)
- cgf: added extra chunk size check on read to aid decoding

- dds: register dds extension with qskope on windows install
- nif: nifoptimize clamps material alpha to [0,1]

Release 0.9.1 (Feb 22, 2008)

- full support for the xml bitstruct tag (for types that contain bit flags)
- added PyFFI.Formats.DDS library for dds file format
- nif: new function for NiPixelData to save image as dds file
- niftoaster: script for exporting images from NiPixelData blocks
- nifoptimize:
  - merge identical shape data blocks
  - remove empty NiNode children
  - update skin partition only if block already exists

Release 0.9.0 (Feb 11, 2008)

- added PyFFI.Formats.KFM library for kfm file format
- cgf.xml and nif.xml updates
- new qBlockParent function to assign parents if the parent block does not contain a reference to the child, but the child contains a reference to the parent (as in MeshMorphTargetChunk and BoneInitialPosChunk)
- QSkope: root blocks sorted by reference number
- QSkope: added kfm format
- niftexdump: bug fixed when reading nifs that have textures without source

Release 0.8.2 (Jan 28, 2008)

- fixed installer bug (nifoptimize would not launch from context menu)
- qskope:
  - handle back-references and shared blocks
  - blocks are now numbered
  - improved display references

Release 0.8.1 (Jan 27, 2008)

- deep copy for structs and arrays
- nifoptimize:
  - detects cases where triangulated geometry performs better than stripified geometry (fixes a performance issue with non-smooth geometry reported by Lazarus)
  - can now also optimize NiTriShapes
  - throws away empty and/or duplicate children in NiNode lists

Release 0.8.0 (Jan 27, 2008)

- qskope: new general purpose tool for visualizing files loaded with PyFFI
- cgf: corrected the bool implementation (using True/False rather than an int)
- nif: many xml updates, support for Culpa Innata, updates for emerge demo
- support for forward declaration of types (required for UnionBV)
- PyFFI.__hexversion__ for numeric represenation of the version number

Release 0.7.5 (Jan 14, 2008)

- added a DTD for the 'fileformat' document type, to validate the xml
- bits tag for bitstructs, instead of add tag, to allow validation
- cgf: write the chunk header table at start, for crysis
- nifoptimize:
  - new command line option '-x' to exclude blocks per type
  - fixes corrupted texture paths (that is, files that got corrupted with nifskope 1.0 due to the \r \n bug)
  - on windows, the script can now be called from the .nif context menu
  - accept both lower and upper case 'y' for confirmation
  - new command line option '-p' to pause after run
- niftoaster: fix reporting of file size difference in readwrite test
- bug fixed when writing nifs of version <= 3.1
- support for multiple 'Top Level Object' (roots) for nifs of version <= 3.1
- various xml fixes
  - new version 20.3.0.2 from emerge demo
  - NiMeshPSysData bugfix and simplification
  - replaced NiTimeController Target with unknown int to cope with invalid pointers in nif versions <= 3.1
- fixed bug nifmakehsl.py script
- fixed bug in nifdump.py script
- new post installation script for installing/uninstalling registry keys

Release 0.7.4 (Dec 26, 2007)

- fix in nif xml for a long outstanding issue which caused some nifs with mopp shapes to fail
- fixed file size check bug in readwrite test for nif and cgf
- initial read and write support for crysis cgf files
- support for versions in structs
- updates for controller key types 6, 9, and 10, in cgf xml

Release 0.7.3 (Dec 13, 2007)

- nif: fixed error message when encountering empty block type
- nif: dump script with block selection feature
- cgf: fix transform errors, ported matrix and vector operations from nif library

Release 0.7.2 (Dec 3, 2007)

- NifTester: new raisereaderror argument which simplifies the older system and yields more instructive backtraces
- nif: better support for recent nif versions, if block sizes do not match with the number of bytes read then the bytes are skipped and a warning is printed, instead of raising an exception

Release 0.7.1 (Nov 27, 2007)

- nif: fixed applyScale in bhkRigidBody

Release 0.7 (Nov 19, 2007)

- fixed a problem locating the customized functions for Fedora 8 python which does not look in default locations besides sys.path
- new vector and matrix library under Utils (for internal use)
- new quick hull library for computing convex hulls
- new inertia library for computing mass, center of gravity, and inertia tensors of solid and hollow objects
- nif: fixed order of bhkCollisionObject when writing NIF files
- nif: new bhkRigidBody function for updating inertia, center of gravity, and mass, for all types of primitives

Release 0.6 (Nov 3, 2007)

- nifoptimize removes duplicate property blocks
- reduced memory footprint in skin data center and radius calculation for the nif format
- new option to ignore strings when calculating hash
- code has been cleaned up using pylint
- added a lot more documentation
- refactored all common functions to take **kwargs as argument
- read and write functions have the file stream as first non-keyword argument
- refactored and simplified attribute parsing, using a common _filteredAttributeList method used by all methods that need to parse attributes; the version and user_version checks are now also consistent over all functions (i.e. getRefs, getLinks, etc.)
- added more doctests

Release 0.5.2 (Oct 25, 2007)

- added hash functions (useful for identifying and comparing objects)

Release 0.5.1 (Oct 19, 2007)

- fixed a bug in the nif.xml file which prevented Oblivion skeleton.nif files to load

Release 0.5 (Oct 19, 2007)

- new functions to get block size
- various small bugs fixed
- nif: support for new versions (20.2.0.6, 20.2.0.7, 20.2.0.8, 20.3.0.3, 20.3.0.6, 20.3.0.9)
- nif: block sizes are now also written to the NIF files, improving support for writing 20.2.0.7+ nif versions
- nif: fixed flattenSkin bug (reported by Kikai)

Release 0.4.9 (Oct 13, 2007)

- nif: nifoptimize no longer raises an exception on test errors, unless you pass the -r option
- nif: nifoptimize will try to restore the original file if something goes wrong during write, so - in theory - it should no longer leave you with corrupt nifs; still it is recommended to keep your backups around just in case
- nif: niftesters recoded to accept arbitrary argument dictionaries; this could cause incompatibilities for people writing their own scripts, but the upgrade to the new system is fairly simple: check the niftemplate.py script
- nif: fixed bug in updateTangentSpace which caused an exception when uvs or normals were not present
- nif: doctest for unsupported blocks in nifs

Release 0.4.8 (Oct 7, 2007)

- cgf: MeshMorphTargetChunk is now supported too
- nif: new script (niftexdump.py) to dump texture and material info
- nif: added template script for quickly writing new nif scripts

Release 0.4.7 (Oct 4, 2007)

- nif: new optimizer script

Release 0.4.6 (Sep 29, 2007)

- nif and cgf documentation improved
- added a number of new doctests
- nif: new scripts
  - niftoaster.py for testing and modifying NIF files (contributed by wz)
  - nifvisualizer.py for visualizing nif blocks (contributed by wz)
  - nifmakehsl.py for making hex workshop structure libraries for all nif versions
- nif: bundling NifVis and NifTester modules so you can make your own nif toasters and visualizers
- nif: fixed rare issue with skin partition calculation
- cgf: new script
  - cgftoaster.py for testing and modifying cgf files (similar to niftoaster.py)
- cgf: bundling CgfTester module so you can make your own cgf toasters
- cgf: various xml bugs fixed
- cgf: write support improved (but not entirely functional yet)
- cgf: material chunk custom function for extraction material shader and script
- Expression.py: support for empty string check in condition

Release 0.4.5 (Sep 16, 2007)

- issue warning message instead of raising exception for improper rotation matrix in setScaleRotationTranslation
- fixed skin partition bug during merge
- skin partition bone index padding and sorting for Freedom Force vs. the 3rd Reich

Release 0.4.4 (Sep 2, 2007)

- added mopp parser and simple mopp generator

Release 0.4.3 (Aug 17, 2007)

- fixed bug that occurred if userver = 0 in the xml (fixes geometry morph data in NIF versions 20.0.0.4 and up)
- NIF:
  - tree() function has been extended
  - some minor cleanups and more documentation

Release 0.4.2 (Aug 15, 2007)

- kwargs for getRefs
- NIF:
  - fixed bug in skin partition calculation
  - when writing NIF files the refs are written in sequence (instead of the links, so missing links will yield an exception, which is a good thing)
  - new functions to get list of extra data blocks and to add effect

Release 0.4.1 (Aug 14, 2007)

- NIF:
  - new function to add collision geometries to packed tristripsshape
  - fixed bug in bhkListShape.addShape

Release 0.4 (Aug 12, 2007)

- NIF:
  - new function updateBindPosition in NiGeometry to fix a geometry rest position from current bone positions
  - removed deprecated functions
  - (!) changed interface of addBone, no longer takes "transform" argument; use the new function updateBindPosition instead

Release 0.3.4 (Aug 11, 2007)

- improved documentation
- fixed the 'in' operator in Bases/Array.py
- NIF:
  - doctest for NiNode
  - flatten skin fix for skins that consist of multiple shapes
  - support for the most common oblivion havok blocks

Release 0.3.3 (Aug 8, 2007)

- NIF:
  - fixed a bug in the skin center and radius calculation
  - added copy function to Vector3
  - fixed NiGeometry doctest

Release 0.3.2 (Aug 7, 2007)

- simplified interface (still wip) by using keyword arguments for common functions such as read and write
- NIF:
  - fix for skin partition blocks in older nif versions such as Freedom Force vs. 3rd Reich
  - support for triangle skin partitions
  - added stitchstrips option for skin partitions
  - added a NiGeometry function to send bones to bind pose

Release 0.3.1 (Aug 6, 2007)

- NIF:
  - new function for getting geometry skin deformation in rest pose
  - old rest pose functions are deprecated and will be removed from a future release

Release 0.3 (Aug 2, 2007)

- NIF:
  - fixed an issue with writing skeleton.nif files
- CGF:
  - reading support for the most common blocks in static cgf files; experimental

Release 0.2.1 (Jul 29, 2007)

- NIF:
  - fixed bug in getTransform
  - new option in findChain to fix block type

Release 0.2 (Jul 29, 2007)

- fixed argument passing when writing arrays
- NIF: added getControllers function to NiObjectNET

Release 0.1 (Jul 22, 2007)

- bug fixed when writing array of strings
- NIF
  - new function to add bones
  - XML update, supports newer versions from Emerge Demo

Release 0.0 (Jul 7, 2007)

- first public release

1.7.8 Todo list

Todo: Write documentation.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pyffi/checkouts/latest/pyffi/spells/cgf/__init__.py:doc
of pyffi.spells.cgf, line 4.)

Todo: Limit the size of shapes (see operation optimization mod for Oblivion!)

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pyffi/checkouts/latest/pyffi/spells/nif/optimize.py:doc
of pyffi.spells.nif.optimize.SpellOptimizeGeometry.branchentry, line 8.)

Todo:

- Write dedicated utilities to optimize particular games (start with Oblivion, maybe eventually also do Fallout 3, Morrowind, etc.).

(The original entry is located in ../TODO.rst, line 3.)

Todo: Aion caf format (MtlNameChunk header?).

(The original entry is located in ../TODO.rst, line 9.)

Todo: refactoring plans

- what's up with get_global_child_names?
- common base classes for pyffi.object_models.xml.basic.BasicBase/StructBase and pyffi.object_models.xsd.SimpleType/ComplexType (e.g. pyffi.ObjectModel.SimpleType/ComplexType)
- derive object_models.array_type and object_models.StructType from common subclass pyffi.object_models.ComplexType, use these then as base classes for object_models.xml.array and object_models.xml.struct_.StructBase
- use pyffi.utils.graph for all object_models.XXX implementations
- upgrade QSkope and XML model to use GlobalNode instead of the current ad hoc system with Refs
- improve the abstract object_models.Delegate classes (i.e. naming, true abstract base classes, defining a common interface); also perhaps think about deriving these delegate classes from TreeLeaf (only leafs have editors!)?
- ditch version and user_version from the object_models interface, and instead use object_models.Data as a global root element that contains all file information with a minimal format independent interface; implementation plan (this is already partially implemented, namely in the nif format):
  - use abstract Data and Tree base classes fo the XSD parser, in subsequent 2.x.x releases
  - update the XML parser to follow the same scheme, when switching from 2.x.x to 3.0.0, and document the 2.x.x -> 3.0.0 migration strategy

  - deprecate the old methods (XxxFormat.read, XxxFormat.write, XxxFormat.getVersion, and so on) in 3.x.x
  - remove old method in 4.x.x
- one of the aims is that qskope no longer relies on any object_models.xml/object_models.xsd specific implementations; if it only relies on the abstract base classes in object_models.Graph and object_models.Data then future expansions are a lot easier to cope with; in particular, qskope should never have to import from object_models.XXX, or Formats.XXX

(The original entry is located in ../TODO.rst, line 13.)

Todo: Doctests for all spells.

(The original entry is located in ../TODO.rst, line 62.)

Todo: Improve overall documentation, for instance:

- add docstrings for all spells
- add docstrings for all spell methods

(The original entry is located in ../TODO.rst, line 66.)

Todo:

- move all regression tests to the tests directory (but keep useful examples in the docstrings!)
- add spell support for qskope directly using the pyffi.spells module
- allow qskope to create new spells, from a user supplied spells module
  - custom spell module creation wizard (creates dir structure for user and stores it into the configuration)
  - custom spell creation wizard (adds new spell to user's spell module)
  - automatic templates for typical spells
- pep8 conventions
  - resolve all complaints from cheesecake's pep8 checker

(The original entry is located in ../TODO.rst, line 73.)

Todo:

- Write dedicated utilities to optimize particular games (start with Oblivion, maybe eventually also do Fallout 3, Morrowind, etc.).

Todo: Aion caf format (MtlNameChunk header?).

Todo: refactoring plans

- what's up with get_global_child_names?

- common base classes for pyffi.object_models.xml.basic.BasicBase/StructBase and pyffi.object_models.xsd.SimpleType/ComplexType (e.g. pyffi.ObjectModel.SimpleType/ComplexType)
- derive object_models.array_type and object_models.StructType from common subclass pyffi.object_models.ComplexType, use these then as base classes for object_models.xml.array and object_models.xml.struct_.StructBase
- use pyffi.utils.graph for all object_models.XXX implementations
- upgrade QSkope and XML model to use GlobalNode instead of the current ad hoc system with Refs
- improve the abstract object_models.Delegate classes (i.e. naming, true abstract base classes, defining a common interface); also perhaps think about deriving these delegate classes from TreeLeaf (only leafs have editors!)?
- ditch version and user_version from the object_models interface, and instead use object_models.Data as a global root element that contains all file information with a minimal format independent interface; implementation plan (this is already partially implemented, namely in the nif format):
  - use abstract Data and Tree base classes fo the XSD parser, in subsequent 2.x.x releases
  - update the XML parser to follow the same scheme, when switching from 2.x.x to 3.0.0, and document the 2.x.x -> 3.0.0 migration strategy
  - deprecate the old methods (XxxFormat.read, XxxFormat.write, XxxFormat.getVersion, and so on) in 3.x.x
  - remove old method in 4.x.x
- one of the aims is that qskope no longer relies on any object_models.xml/object_models.xsd specific implementations; if it only relies on the abstract base classes in object_models.Graph and object_models.Data then future expansions are a lot easier to cope with; in particular, qskope should never have to import from object_models.XXX, or Formats.XXX

Todo: Doctests for all spells.

Todo: Improve overall documentation, for instance:

- add docstrings for all spells
- add docstrings for all spell methods

Todo:

- move all regression tests to the tests directory (but keep useful examples in the docstrings!)
- add spell support for qskope directly using the pyffi.spells module
- allow qskope to create new spells, from a user supplied spells module
  - custom spell module creation wizard (creates dir structure for user and stores it into the configuration)
  - custom spell creation wizard (adds new spell to user's spell module)
  - automatic templates for typical spells
- pep8 conventions
  - resolve all complaints from cheesecake's pep8 checker

1.7.9 Thanks

Special thanks go in particular to:

- Guido van Rossum, and with him many others, for Python, and in particular for having metaclasses in Python: metaclasses make PyFFI's implementation very easy.
- m4444x for nifskope, which has been an inspiration for PyFFI's xml based design, and of course also an inspiration for QSkope.
- wz for his support, and for testing of the library, when the first version was being written.
- seith for design of the windows installer artwork.
- Crytek for releasing the Far Cry SDK and Crysis SDK, which contains much information about the cgf file format. This has saved many months of hard work.
- Crytek and Bethesda for the great games they make.
- Havok, for releasing their SDK without which custom mopp generation would not have been possible.
- Karl Norby and Michael Summers for pyxsd, which forms the basis of the xsd object model, used for instance to support Collada.

1.7.10 Glossary

PyFFI Python File Format Interface. Also see *file*, *interface*, and `pyffi`.

file A byte stream.

file format See *interface*.

file format interface See *interface*.

interface An interface provides a semantic translation of a byte stream to an organized collection of named Python objects, which are typically bundled into a classes.

spell A transformation that can be applied to a file.

toaster Applies one or more spells to all files of all subdirectories of a given directory.

1.8 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

F

G

H

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z