PyFFI

Release 2.2.3

Amorilia

May 21, 2019

CONTENTS

Release 2.2.3

Date May 21, 2019

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-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
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.") # doctest: +REPORT_NDIFF
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.

blending

Vertex weight.

bone

The bone chunk.

offset

The bone offset?

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.

id
The chunk identifier.

offset
Position of the chunk in the CGF file.

type
Type of chunk referred to.

version
Version of the chunk referred to.

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.

material
Material index.

sm_group
Smoothing group.

v_0
First vertex index.

v_1
Second vertex index.

v_2
Third vertex index.

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.

offset

Position of the chunk table in the CGF file.

signature

The CGF file signature.

type

The CGF file type (geometry or animation).

version

The version of the chunk table.

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 MeshChunk(*template=None*, *argument=None*, *parent=None*)
Bases: `pyffi.formats.cgf._MeshChunk`, `object`

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

get_colors()
Generator for all vertex colors.

get_material_indices()
Generator for all materials (per triangle).

get_normals()
Generator for all normals.

get_num_triangles()
Get number of triangles.

get_triangles()
Generator for all triangles.

get_uv_triangles()
Generator for all uv triangles.

get_uvs()
Generator for all uv coordinates.

get_vertices()
Generator for all vertices.

set_geometry(*verticeslist=None*, *normalslist=None*, *triangleslist=None*, *matlist=None*, *uvslist=None*, *colorslist=None*)
Set geometry data.

```
>>> from pyffi.formats.cgf import CgfFormat
>>> chunk = CgfFormat.MeshChunk()
>>> vertices1 = [(0,0,0),(0,1,0),(1,0,0),(1,1,0)]
>>> vertices2 = [(0,0,1),(0,1,1),(1,0,1),(1,1,1)]
>>> normals1 = [(0,0,-1),(0,0,-1),(0,0,-1),(0,0,-1)]
>>> normals2 = [(0,0,1),(0,0,1),(0,0,1),(0,0,1)]
>>> triangles1 = [(0,1,2),(2,1,3)]
>>> triangles2 = [(0,1,2),(2,1,3)]
>>> uvs1 = [(0,0),(0,1),(1,0),(1,1)]
>>> uvs2 = [(0,0),(0,1),(1,0),(1,1)]
>>> colors1 = [(0,1,2,3),(4,5,6,7),(8,9,10,11),(12,13,14,15)]
>>> colors_2 = [(50,51,52,53),(54,55,56,57),(58,59,60,61),(62,63,64,65)]
>>> chunk.set_geometry(verticeslist = [vertices1, vertices2],
...                    normalslist = [normals1, normals2],
...                    triangleslist = [triangles1, triangles2],
...                    uvslist = [uvs1, uvs2],
...                    matlist = [2,5],
...                    colorslist = [colors1, colors_2])
>>> print(chunk) # doctest: +ELLIPSIS +REPORT_UDIFF
<class 'pyffi.formats.cgf.MeshChunk'> instance at ...
* has_vertex_weights : False
* has_vertex_colors : True
* in_world_space : False
* reserved_1 : 0
```

(continues on next page)

(continued from previous page)

```
* reserved_2 : 0
* flags_1 : 0
* flags_2 : 0
* num_vertices : 8
* num_indices : 12
* num_uvs : 8
* num_faces : 4
* material : None
* num_mesh_subsets : 2
* mesh_subsets : <class 'pyffi.formats.cgf.MeshSubsetsChunk'> instance at␣
↪...
* vert_anim : None
* vertices :
    <class 'pyffi.object_models.xml.array.Array'> instance at ...
    0: <class 'pyffi.formats.cgf.Vertex'> instance at ...
    * p : [  0.000  0.000  0.000 ]
    * n : [  0.000  0.000 -1.000 ]
    1: <class 'pyffi.formats.cgf.Vertex'> instance at ...
    * p : [  0.000  1.000  0.000 ]
    * n : [  0.000  0.000 -1.000 ]
    2: <class 'pyffi.formats.cgf.Vertex'> instance at ...
    * p : [  1.000  0.000  0.000 ]
    * n : [  0.000  0.000 -1.000 ]
    3: <class 'pyffi.formats.cgf.Vertex'> instance at ...
    * p : [  1.000  1.000  0.000 ]
    * n : [  0.000  0.000 -1.000 ]
    4: <class 'pyffi.formats.cgf.Vertex'> instance at ...
    * p : [  0.000  0.000  1.000 ]
    * n : [  0.000  0.000  1.000 ]
    5: <class 'pyffi.formats.cgf.Vertex'> instance at ...
    * p : [  0.000  1.000  1.000 ]
    * n : [  0.000  0.000  1.000 ]
    6: <class 'pyffi.formats.cgf.Vertex'> instance at ...
    * p : [  1.000  0.000  1.000 ]
    * n : [  0.000  0.000  1.000 ]
    7: <class 'pyffi.formats.cgf.Vertex'> instance at ...
    * p : [  1.000  1.000  1.000 ]
    * n : [  0.000  0.000  1.000 ]
* faces :
    <class 'pyffi.object_models.xml.array.Array'> instance at ...
    0: <class 'pyffi.formats.cgf.Face'> instance at ...
    * v_0 : 0
    * v_1 : 1
    * v_2 : 2
    * material : 2
    * sm_group : 1
    1: <class 'pyffi.formats.cgf.Face'> instance at ...
    * v_0 : 2
    * v_1 : 1
    * v_2 : 3
    * material : 2
    * sm_group : 1
    2: <class 'pyffi.formats.cgf.Face'> instance at ...
    * v_0 : 4
    * v_1 : 5
    * v_2 : 6
    * material : 5
```

(continues on next page)

(continued from previous page)

```
    * sm_group : 1
    3: <class 'pyffi.formats.cgf.Face'> instance at ...
    * v_0 : 6
    * v_1 : 5
    * v_2 : 7
    * material : 5
    * sm_group : 1
* uvs :
    <class 'pyffi.object_models.xml.array.Array'> instance at ...
    0: <class 'pyffi.formats.cgf.UV'> instance at ...
    * u : 0.0
    * v : 0.0
    1: <class 'pyffi.formats.cgf.UV'> instance at ...
    * u : 0.0
    * v : 1.0
    2: <class 'pyffi.formats.cgf.UV'> instance at ...
    * u : 1.0
    * v : 0.0
    3: <class 'pyffi.formats.cgf.UV'> instance at ...
    * u : 1.0
    * v : 1.0
    4: <class 'pyffi.formats.cgf.UV'> instance at ...
    * u : 0.0
    * v : 0.0
    5: <class 'pyffi.formats.cgf.UV'> instance at ...
    * u : 0.0
    * v : 1.0
    6: <class 'pyffi.formats.cgf.UV'> instance at ...
    * u : 1.0
    * v : 0.0
    7: <class 'pyffi.formats.cgf.UV'> instance at ...
    * u : 1.0
    * v : 1.0
* uv_faces :
    <class 'pyffi.object_models.xml.array.Array'> instance at ...
    0: <class 'pyffi.formats.cgf.UVFace'> instance at ...
    * t_0 : 0
    * t_1 : 1
    * t_2 : 2
    1: <class 'pyffi.formats.cgf.UVFace'> instance at ...
    * t_0 : 2
    * t_1 : 1
    * t_2 : 3
    2: <class 'pyffi.formats.cgf.UVFace'> instance at ...
    * t_0 : 4
    * t_1 : 5
    * t_2 : 6
    3: <class 'pyffi.formats.cgf.UVFace'> instance at ...
    * t_0 : 6
    * t_1 : 5
    * t_2 : 7
* vertex_colors :
    <class 'pyffi.object_models.xml.array.Array'> instance at ...
    0: <class 'pyffi.formats.cgf.IRGB'> instance at ...
    * r : 0
    * g : 1
    * b : 2
```

(continues on next page)

(continued from previous page)

```
    1: <class 'pyffi.formats.cgf.IRGB'> instance at ...
    * r : 4
    * g : 5
    * b : 6
    2: <class 'pyffi.formats.cgf.IRGB'> instance at ...
    * r : 8
    * g : 9
    * b : 10
    3: <class 'pyffi.formats.cgf.IRGB'> instance at ...
    * r : 12
    * g : 13
    * b : 14
    4: <class 'pyffi.formats.cgf.IRGB'> instance at ...
    * r : 50
    * g : 51
    * b : 52
    5: <class 'pyffi.formats.cgf.IRGB'> instance at ...
    * r : 54
    * g : 55
    * b : 56
    6: <class 'pyffi.formats.cgf.IRGB'> instance at ...
    * r : 58
    * g : 59
    * b : 60
    7: <class 'pyffi.formats.cgf.IRGB'> instance at ...
    * r : 62
    * g : 63
    * b : 64
* vertices_data : <class 'pyffi.formats.cgf.DataStreamChunk'> instance at␣
↪...
* normals_data : <class 'pyffi.formats.cgf.DataStreamChunk'> instance at .
↪..
* uvs_data : <class 'pyffi.formats.cgf.DataStreamChunk'> instance at ...
* colors_data : <class 'pyffi.formats.cgf.DataStreamChunk'> instance at ..
↪.
* colors_2_data : None
* indices_data : <class 'pyffi.formats.cgf.DataStreamChunk'> instance at .
↪..
* tangents_data : <class 'pyffi.formats.cgf.DataStreamChunk'> instance at␣
↪...
* sh_coeffs_data : None
* shape_deformation_data : None
* bone_map_data : None
* face_map_data : None
* vert_mats_data : None
* reserved_data :
    <class 'pyffi.object_models.xml.array.Array'> instance at ...
    0: None
    1: None
    2: None
    3: None
* physics_data :
    <class 'pyffi.object_models.xml.array.Array'> instance at ...
    0: None
    1: None
    2: None
    3: None
```

(continues on next page)

(continued from previous page)

```
* min_bound : [  0.000  0.000  0.000 ]
* max_bound : [  1.000  1.000  1.000 ]
* reserved_3 :
    <class 'pyffi.object_models.xml.array.Array'> instance at ...
    0: 0
    1: 0
    2: 0
    3: 0
    4: 0
    5: 0
    6: 0
    7: 0
    8: 0
    9: 0
    10: 0
    11: 0
    12: 0
    13: 0
    14: 0
    15: 0
    16: 0
    etc...
<BLANKLINE>
>>> print(chunk.mesh_subsets) # doctest: +ELLIPSIS
<class 'pyffi.formats.cgf.MeshSubsetsChunk'> instance at ...
* flags :
    <class 'pyffi.formats.cgf.MeshSubsetsFlags'> instance at ...
    * sh_has_decompr_mat : 0
    * bone_indices : 0
* num_mesh_subsets : 2
* reserved_1 : 0
* reserved_2 : 0
* reserved_3 : 0
* mesh_subsets :
    <class 'pyffi.object_models.xml.array.Array'> instance at ...
    0: <class 'pyffi.formats.cgf.MeshSubset'> instance at ...
    * first_index : 0
    * num_indices : 6
    * first_vertex : 0
    * num_vertices : 4
    * mat_id : 2
    * radius : 0.7071067...
    * center : [  0.500  0.500  0.000 ]
    1: <class 'pyffi.formats.cgf.MeshSubset'> instance at ...
    * first_index : 6
    * num_indices : 6
    * first_vertex : 4
    * num_vertices : 4
    * mat_id : 5
    * radius : 0.7071067...
    * center : [  0.500  0.500  1.000 ]
<BLANKLINE>
>>> print(chunk.vertices_data) # doctest: +ELLIPSIS
<class 'pyffi.formats.cgf.DataStreamChunk'> instance at ...
* flags : 0
* data_stream_type : VERTICES
* num_elements : 8
```

(continues on next page)

(continued from previous page)

```
* bytes_per_element : 12
* reserved_1 : 0
* reserved_2 : 0
* vertices :
    <class 'pyffi.object_models.xml.array.Array'> instance at ...
    0: [  0.000  0.000  0.000 ]
    1: [  0.000  1.000  0.000 ]
    2: [  1.000  0.000  0.000 ]
    3: [  1.000  1.000  0.000 ]
    4: [  0.000  0.000  1.000 ]
    5: [  0.000  1.000  1.000 ]
    6: [  1.000  0.000  1.000 ]
    7: [  1.000  1.000  1.000 ]
<BLANKLINE>
>>> print(chunk.normals_data) # doctest: +ELLIPSIS
<class 'pyffi.formats.cgf.DataStreamChunk'> instance at ...
* flags : 0
* data_stream_type : NORMALS
* num_elements : 8
* bytes_per_element : 12
* reserved_1 : 0
* reserved_2 : 0
* normals :
    <class 'pyffi.object_models.xml.array.Array'> instance at ...
    0: [  0.000  0.000 -1.000 ]
    1: [  0.000  0.000 -1.000 ]
    2: [  0.000  0.000 -1.000 ]
    3: [  0.000  0.000 -1.000 ]
    4: [  0.000  0.000  1.000 ]
    5: [  0.000  0.000  1.000 ]
    6: [  0.000  0.000  1.000 ]
    7: [  0.000  0.000  1.000 ]
<BLANKLINE>
>>> print(chunk.indices_data) # doctest: +ELLIPSIS
<class 'pyffi.formats.cgf.DataStreamChunk'> instance at ...
* flags : 0
* data_stream_type : INDICES
* num_elements : 12
* bytes_per_element : 2
* reserved_1 : 0
* reserved_2 : 0
* indices :
    <class 'pyffi.object_models.xml.array.Array'> instance at ...
    0: 0
    1: 1
    2: 2
    3: 2
    4: 1
    5: 3
    6: 4
    7: 5
    8: 6
    9: 6
    10: 5
    11: 7
<BLANKLINE>
>>> print(chunk.uvs_data) # doctest: +ELLIPSIS
```

(continues on next page)

(continued from previous page)

```
<class 'pyffi.formats.cgf.DataStreamChunk'> instance at ...
* flags : 0
* data_stream_type : UVS
* num_elements : 8
* bytes_per_element : 8
* reserved_1 : 0
* reserved_2 : 0
* uvs :
    <class 'pyffi.object_models.xml.array.Array'> instance at ...
    0: <class 'pyffi.formats.cgf.UV'> instance at ...
    * u : 0.0
    * v : 1.0
    1: <class 'pyffi.formats.cgf.UV'> instance at ...
    * u : 0.0
    * v : 0.0
    2: <class 'pyffi.formats.cgf.UV'> instance at ...
    * u : 1.0
    * v : 1.0
    3: <class 'pyffi.formats.cgf.UV'> instance at ...
    * u : 1.0
    * v : 0.0
    4: <class 'pyffi.formats.cgf.UV'> instance at ...
    * u : 0.0
    * v : 1.0
    5: <class 'pyffi.formats.cgf.UV'> instance at ...
    * u : 0.0
    * v : 0.0
    6: <class 'pyffi.formats.cgf.UV'> instance at ...
    * u : 1.0
    * v : 1.0
    7: <class 'pyffi.formats.cgf.UV'> instance at ...
    * u : 1.0
    * v : 0.0
<BLANKLINE>
>>> print(chunk.tangents_data) # doctest: +ELLIPSIS
<class 'pyffi.formats.cgf.DataStreamChunk'> instance at ...
* flags : 0
* data_stream_type : TANGENTS
* num_elements : 8
* bytes_per_element : 16
* reserved_1 : 0
* reserved_2 : 0
* tangents :
    <class 'pyffi.object_models.xml.array.Array'> instance at ...
    0, 0: <class 'pyffi.formats.cgf.Tangent'> instance at ...
    * x : 32767
    * y : 0
    * z : 0
    * w : 32767
    0, 1: <class 'pyffi.formats.cgf.Tangent'> instance at ...
    * x : 0
    * y : -32767
    * z : 0
    * w : 32767
    1, 0: <class 'pyffi.formats.cgf.Tangent'> instance at ...
    * x : 32767
    * y : 0
```

(continues on next page)

(continued from previous page)

```
    * z : 0
    * w : 32767
    1, 1: <class 'pyffi.formats.cgf.Tangent'> instance at ...
    * x : 0
    * y : -32767
    * z : 0
    * w : 32767
    2, 0: <class 'pyffi.formats.cgf.Tangent'> instance at ...
    * x : 32767
    * y : 0
    * z : 0
    * w : 32767
    2, 1: <class 'pyffi.formats.cgf.Tangent'> instance at ...
    * x : 0
    * y : -32767
    * z : 0
    * w : 32767
    3, 0: <class 'pyffi.formats.cgf.Tangent'> instance at ...
    * x : 32767
    * y : 0
    * z : 0
    * w : 32767
    3, 1: <class 'pyffi.formats.cgf.Tangent'> instance at ...
    * x : 0
    * y : -32767
    * z : 0
    * w : 32767
    4, 0: <class 'pyffi.formats.cgf.Tangent'> instance at ...
    * x : 32767
    * y : 0
    * z : 0
    * w : 32767
    4, 1: <class 'pyffi.formats.cgf.Tangent'> instance at ...
    * x : 0
    * y : -32767
    * z : 0
    * w : 32767
    5, 0: <class 'pyffi.formats.cgf.Tangent'> instance at ...
    * x : 32767
    * y : 0
    * z : 0
    * w : 32767
    5, 1: <class 'pyffi.formats.cgf.Tangent'> instance at ...
    * x : 0
    * y : -32767
    * z : 0
    * w : 32767
    6, 0: <class 'pyffi.formats.cgf.Tangent'> instance at ...
    * x : 32767
    * y : 0
    * z : 0
    * w : 32767
    6, 1: <class 'pyffi.formats.cgf.Tangent'> instance at ...
    * x : 0
    * y : -32767
    * z : 0
    * w : 32767
```

(continues on next page)

(continued from previous page)

```
    7, 0: <class 'pyffi.formats.cgf.Tangent'> instance at ...
    * x : 32767
    * y : 0
    * z : 0
    * w : 32767
    7, 1: <class 'pyffi.formats.cgf.Tangent'> instance at ...
    * x : 0
    * y : -32767
    * z : 0
    * w : 32767
<BLANKLINE>
>>> print(chunk.colors_data) # doctest: +ELLIPSIS
<class 'pyffi.formats.cgf.DataStreamChunk'> instance at ...
* flags : 0
* data_stream_type : COLORS
* num_elements : 8
* bytes_per_element : 4
* reserved_1 : 0
* reserved_2 : 0
* rgba_colors :
    <class 'pyffi.object_models.xml.array.Array'> instance at ...
    0: <class 'pyffi.formats.cgf.IRGBA'> instance at ...
    * r : 0
    * g : 1
    * b : 2
    * a : 3
    1: <class 'pyffi.formats.cgf.IRGBA'> instance at ...
    * r : 4
    * g : 5
    * b : 6
    * a : 7
    2: <class 'pyffi.formats.cgf.IRGBA'> instance at ...
    * r : 8
    * g : 9
    * b : 10
    * a : 11
    3: <class 'pyffi.formats.cgf.IRGBA'> instance at ...
    * r : 12
    * g : 13
    * b : 14
    * a : 15
    4: <class 'pyffi.formats.cgf.IRGBA'> instance at ...
    * r : 50
    * g : 51
    * b : 52
    * a : 53
    5: <class 'pyffi.formats.cgf.IRGBA'> instance at ...
    * r : 54
    * g : 55
    * b : 56
    * a : 57
    6: <class 'pyffi.formats.cgf.IRGBA'> instance at ...
    * r : 58
    * g : 59
    * b : 60
    * a : 61
    7: <class 'pyffi.formats.cgf.IRGBA'> instance at ...
```

(continues on next page)

(continued from previous page)

```
    * r : 62
    * g : 63
    * b : 64
    * a : 65
<BLANKLINE>
```

Parameters

- **verticeslist** – A list of lists of vertices (one list per material).
- **normalslist** – A list of lists of normals (one list per material).
- **triangleslist** – A list of lists of triangles (one list per material).
- **matlist** – A list of material indices. Optional.
- **uvslist** – A list of lists of uvs (one list per material). Optional.
- **colorslist** – A list of lists of RGBA colors (one list per material). Optional. Each color is a tuple (r, g, b, a) with each component an integer between 0 and 255.

set_vertices_normals(*vertices*, *normals*)

B{Deprecated. Use L{set_geometry} instead.} Set vertices and normals. This used to be the first function to call when setting mesh geometry data.

Returns list of chunks that have been added.

update_tangent_space()

Recalculate tangent space data.

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).

w

Fourth coordinate.

x

First coordinate.

y

Second coordinate.

z

Third coordinate.

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 ScenePropsChunk(*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
```

(continues on next page)

(continued from previous page)

```
>>> 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.

w
Handness? Either 32767 (+1.0) or -32767 (-1.0).

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).

t_0
First vertex index.

t_1
Second vertex index.

t_2
Third vertex index.

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
>>> 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) # doctest: +ELLIPSIS
<class '...SourceInfoChunk'> instance at ...
* source_file : <None>
* date : Fri Sep 28 22:40:44 2007
* author : blender@BLENDER
<BLANKLINE>
<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
<BLANKLINE>
```

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, "/")
```

(continues on next page)

(continued from previous page)

```
...         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) # doctest: +ELLIPSIS +REPORT_NDIFF
<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 :
```

(continues on next page)

(continued from previous page)

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

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) # doctest: +ELLIPSIS
<...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
```

(continues on next page)

(continued from previous page)

```
>>> 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) # doctest: +ELLIPSIS
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) # doctest: +ELLIPSIS
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
```

(continues on next page)

(continued from previous page)

```
>>> 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.") # doctest: +REPORT_NDIFF
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 `str`s

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 `DetailNode`s

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 # doctest: +ELLIPSIS
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)])
```

(continues on next page)

(continued from previous page)

```
>>> 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()
```

(continues on next page)

(continued from previous page)

```
>>> 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
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.") # doctest: +REPORT_NDIFF
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) # doctest: +ELLIPSIS
>>> # 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.") # doctest: +REPORT_NDIFF
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.

data_size
Length of the data.

type
The type of the record.

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.") # doctest: +REPORT_NDIFF
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')
```

(continues on next page)

(continued from previous page)

```
>>> 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.") # doctest: +REPORT_NDIFF
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"
>>> 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)
```

(continues on next page)

(continued from previous page)

```
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.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 `str`s

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 `DetailNode`s

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.") # doctest: +REPORT_NDIFF
```

(continues on next page)

(continued from previous page)

```
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. Notethat Header.modifier_vertices counts up from the first modifieronwards. For example, if you were to take the 4th vertex to bemodified in the 2nd modifier and the size of the 1st modifierwas 10 vertices, then you would needHeader.modifier_vertices[14]. Therefore,Header.num_modifier_vertices =sum(modifier.num_vertices_to_modify for modifier inHeader.modifiers).

modifier_vertices
The actual modifier vertices (copied from Header.modifier_vertices).

name
Name of the Modifier.

vertices_to_modify
List of Vertices To Modify.

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 # doctest: +ELLIPSIS
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
```

(continues on next page)

(continued from previous page)

```
18
>>> data.read(stream) # doctest: +ELLIPSIS
>>> 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.") # doctest: +REPORT_NDIFF
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>
```

(continues on next page)

(continued from previous page)

```
        <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()

        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.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`).

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`

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 `str`s

`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")
```

(continues on next page)

(continued from previous page)

```
['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-tagnum/checkouts/hfloat/pyffi/spells/cgf/__init__.py:docstring of pyffi.spells.cgf, line 4.)

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

E

F

G

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z