io_anim_mvnx

Release 0.1.0

Andres FR

Jul 25, 2019

CONTENTS:

CHAPTER

ONE

IO_ANIM_MVNX PACKAGE

1.1 Submodules

1.2 io_anim_mvnx.mvnx module

This module contains functionality concerning the adaption of the XSENS MVN-XML format into our Python setup.

The official explanation can be found in section 14.4 of this document:: https://usermanual.wiki/Document/MVNUserManual.1147412416.pdf

A copy is stored in this repository.

The following section introduces more informally the contents of the imported MVN file and the way they can be accessed from Python:

```
# MVNX schemata can be found in this package or in
# https://www.xsens.com/mvn/mvnx/schema.xsd
mvn_path = "XXX"
mmvn = Mvnx(mvn_path)

# These elements contain some small metadata:
mmvn.mvnx.attrib
mmvn.mvnx.comment.attrib
mmvn.mvnx.securityCode.attrib["code"]
mmvn.mvnx.subject.attrib

# subject.segments contain 3D pos_b labels:
for ch in mmvn.mvnx.subject.segments.iterchildren():
    ch.attrib, [p.attrib for p in ch.points.iterchildren()]

# Segments can look as follows: ``['Pelvis', 'L5', 'L3', 'T12', 'T8', 'Neck',
'Head', 'RightShoulder', 'RightUpperArm', 'RightForeArm', 'RightHand',
'LeftShoulder', 'LeftUpperArm', 'LeftForeArm', 'LeftHand', 'RightUpperLeg',
'RightLowerLeg', 'RightFoot', 'RightToe', 'LeftUpperLeg', 'LeftLowerLeg',
'LeftFoot', 'LeftToe']``

# sensors is basically a list of names
for s in mmvn.mvnx.subject.sensors.iterchildren():
    s.attrib

#  Joints is a list that connects segment points:
for j in mmvn.mvnx.subject.joints.iterchildren():
    j.attrib["label"], j.getchildren()
```

(continues on next page)

(continued from previous page)

```
# miscellaneous:
for j in mmvn.mvnx.subject.ergonomicJointAngles.iterchildren():
    j.attrib, j.getchildren()

for f in mmvn.mvnx.subject.footContactDefinition.iterchildren():
    f.attrib, f.getchildren()

# The bulk of the data is in the frames.
frames_metadata, config_frames, normal_frames = mmvn.extract_frame_info()

# Metadata looks like this:
{'segmentCount': '23', 'sensorCount': '17', 'jointCount': '22'}

# config frames have the following fields:
['orientation', 'position', 'time', 'tc', 'ms', 'type']

# normal frames have the following fields:
['orientation', 'position', 'velocity', 'acceleration',
 'angularVelocity', 'angularAcceleration', 'footContacts',
 'sensorFreeAcceleration', 'sensorMagneticField', 'sensorOrientation',
 'jointAngle', 'jointAngleXZY', 'jointAngleErgo', 'centerOfMass', 'time',
 'index', 'tc', 'ms', 'type']
```

The following fields contain metadata about the frame:

time ms since start (integer). It is close to `int(1000.0 * index / samplerate)`, being equal most of the times and at most 1 milisecond away. It is neither truncated nor rounded, maybe it is given by the hardware.

index starts with 0, +1 each normal frame

tc string like '02:23:28:164'

ms unix timestamp like 1515983008686 (used to compute time)

type one of "identity", "tpose", "tpose-isb", "normal"

The following fields are float vectors of the following dimensionality:

orientation `segmentCount*4 = 92` Quaternion vector

position, velocity, acceleration, angularVelocity, angularAcceleration `segmentCount*3 = 69` 3D vectors in `(x,y,z)` format

footContacts `4` 4D boolean vector

sensorFreeAcceleration, sensorMagneticField `sensorCount*3 = 51`

sensorOrientation `sensorCount*4 = 68`

jointAngle, jointAngleXZY `jointCount*3 = 66`

jointAngleErgo `12`

centerOfMass `3`

The units are SI for position, velocity and acceleration. Angular magnitudes are in radians except the `jointAngle...` ones that are in degrees. All 3D vectors are in `(x,y,z)` format, but the `jointAngle...` ones differ in the Euler-rotation order by which they are computed (ZXY, standard or XZY, for shoulders usually).

class io_anim_mvnx.mvnx.**Mvnx**(*mvnx_path*, *mvnx_schema_path=None*)

Bases: `object`

This class imports and adapts an XML file (expected to be in MVNX format) to a Python-friendly representation. See this module's docstring for usage examples and more information.

export (*filepath*, *pretty_print=True*, *extra_comment=''*)
Saves the current `mvnx` attribute to the given file path as XML and adds the `self.mvnx.attrib["pythonComment"]` attribute with a timestamp.

extract_frame_info()

Returns The tuple `(frames_metadata, config_frames, normal_frames)`

static extract_frames(*mvnx*)
The bulk of the MVNX file is the `mvnx->subject->frames` section. This function parses it and returns its information in a python-friendly format.

Parameters **mvnx** – An XML tree, expected to be in MVNX format

Returns a tuple `(frames_metadata, config_frames, normal_frames)` where the metadata is a dict in the form `{'segmentCount': '23', 'sensorCount': '17', 'jointCount': '22'}`, the config frames are the first 3 frame entries (expected to contain special config info) and the normal_frames are all frames starting from the 4th. Both frame outputs are relational collections of dictionaries that can be formatted into tabular form.

extract_segments()

Returns A list of the segment names in `self.mvnx.subject.segments` ordered by id (starting at 1 and incrementing +1).

1.3 io_anim_mvnx.utils module

Utilities for interaction with Blender

class io_anim_mvnx.utils.**ArgumentParserForBlender**(*prog=None*, *usage=None*, *description=None*, *epilog=None*, *parents=[]*, *formatter_class=<class 'argparse.HelpFormatter'>*, *prefix_chars='-'*, *fromfile_prefix_chars=None*, *argument_default=None*, *conflict_handler='error'*, *add_help=True*, *allow_abbrev=True*)

Bases: `argparse.ArgumentParser`

This class is identical to its superclass, except for the parse_args method (see docstring). It resolves the ambiguity generated when calling Blender from the CLI with a python script, and both Blender and the script have arguments. E.g., the following call will make Blender crash because it will try to process the script's -a and -b flags:

```
blender --python my_script.py -a 1 -b 2
```

To bypass this issue this class uses the fact that Blender will ignore all arguments given after a double-dash ('–'). The approach is that all arguments before '–' go to Blender, arguments after go to the script. The following CLI calls work fine:

```
blender --python my_script.py -- -a 1 -b 2
blender --python my_script.py --
```

get_argv_after_doubledash(*argv*)

Parameters **argv** (*list<str>*) – Expected to be sys.argv (or alike).

Returns The argv sublist after the first `'--'` element (if present, otherwise returns an empty list).

Return type list of str

Note: Works with any *ordered* collection of strings (e.g. list, tuple).

parse_args()

This method is expected to behave identically as in the superclass, except that the sys.argv list will be pre-processed using get_argv_after_doubledash before. See the docstring of the class for usage examples and details.

Note: By default, *argparse.ArgumentParser* will call *sys.exit()* when encountering an error. Blender will react to that shutting down, making it look like a crash. Make sure the arguments are correct!

class io_anim_mvnx.utils.**KeymapManager**

Bases: `list`

This class implements functionality for registering/deregistering keymaps into Blender. It also behaves like a regular list, holding the keymaps currently registered. To inspect the registered keymaps simply iterate the instance.

KEYMAP_NAME = 'Object Mode'

KEYMAP_REGION_TYPE = 'WINDOW'

KEYMAP_SPACE_TYPE = 'EMPTY'

register(*context*, *key*, *stroke_mode*, *op_name*, *ctrl=True*, *shift=True*, *alt=False*)

Adds a new keymap to this collection, and to the config in `context.window_manager.keyconfigs.addon`. See the API for details:

https://docs.blender.org/api/blender2.8/bpy.types.KeyMap.html
https://docs.blender.org/api/blender2.8/bpy.types.KeyMapItem.html
https://docs.blender.org/manual/de/dev/advanced/keymap_editing.html

Usage example:

```
kmm = KeymapManager()
kmm.register(bpy.context, "D", "PRESS", MyOperator.bl_idname)
```

Parameters

- **context** (*bpy.types.Context*) – The Blender context to work in.
- **key** (*str*) – See bpy.types.KeyMapItem.key_modifier
- **stroke_mode** (*str*) – See bpy.types.KeyMapItem.value
- **op_name** (*str*) – Name of a valid operation in `bpy.ops` (usually the `bl_idname`)
- **ctrl, shift, alt** (*booleans*) – Modifiers of the `key`

Returns None

unregister()
Removes every mapped item from every keymap in this collection, and then empties the collection.

class io_anim_mvnx.utils.**OperatorToMenuManager**
Bases: list

This class implements functionality for adding/removing operators into Blender UI menus. It also behaves like a regular list, holding the currently registered items. Usage example:

```
omm = OperatorToMenuManager()
# In register():
omm.register(MyOperator, bpy.types.VIEW3D_MT_object)
# ... in unregister():
omm.unregister
```

register(*op_class*, *menu_class*)

Parameters

- **op_class** (*bpy.types.Operator*) – (Sub)class handle with desired functionality.
- **menu_class** (*bpy.types.{Header, Panel, ..}*) – Class handle for the Blender GUI where the functionality can be triggered.

Note: op_class must define the bl_idname and bl_label fields.

unregister()
Removes every mapped operator from every menu class in this collection, then empties the collection.

io_anim_mvnx.utils.**make_timestamp**(*timezone='Europe/Berlin'*)
Output example: day, month, year, hour, min, sec, milisecs: 10_Feb_2018_20:10:16.151

io_anim_mvnx.utils.**resolve_path**(**path_elements*)
A convenience path wrapper to find elements in this package. Retrieves the absolute path, given the OS-agnostic path relative to the package root path (by bysically joining the path elements via os.path.join). E.g., the following call retrieves the absolute path for <PACKAGE_ROOT>/a/b/test.txt:

```
resolve_path("a", "b", "test.txt")
```

Params strings path_elements From left to right, the path nodes, the last one being the filename.

Return type str

io_anim_mvnx.utils.**rot_euler_degrees**(*rot_x*, *rot_y*, *rot_z*, *order='XYZ'*)

Parameters **rot** (*float*) – Rotation angle in degrees.

Returns An Euler rotation object with the given rotations (converted to gradians) and rotation order.

1.4 Module contents

Init file for the add-on.

To install it, make sure Blender's Python is able to find it under addon_utils.paths(), and that the Blender version matches to make it installable.

Alternatively, run this init file as a script from Blender.

io_anim_mvnx.**register**()
Main register function, called on startup by Blender

io_anim_mvnx.**unregister**()
Main unregister function, called on shutdown by Blender

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX