SQLAlchemy: Recipes

The Problem

Date values from strings

- Deserializing JSON to ORM objects
- JSON is a lame type-less encoding scheme.
- Dates are strings:
  - “2011-12-06T00:16:00”
  - These strings need to become dates magically.

Craft Some Magic

Date values from strings

```
class JSONDateTime(TypeDecorator):
    """Allow storing a string into a datetime value, this allows for
       automatically conversion of the JSON date strings into date values"""
    impl = DateTime

    def __init__(self, *arg, **kw):
        TypeDecorator.__init__(self, *arg, **kw)

    def process_bind_param(self, value, dialect):
        if value:
            if isinstance(value, basestring):
                if (len(value) == 19):
                    return datetime.strptime(value, '%Y-%m-%dT%H:%M:%S')
                elif (len(value) == 16):
                    return datetime.strptime(value, '%Y-%m-%dT%H:%M')
                elif (len(value) == 0):
                    return None
            elif isinstance(value, datetime):
                return value
            raise Exception('Cannot store value "{0}" as DateTime'.format(value))
        return None
```

Use The Magic

Date values from strings

```
class LogEntry(Base):
    __tablename__ = 'logentry'
    LUID                = Column(String(32), primary_key=True)
    PARENT_LUID         = Column(String(32))
    ACTOR_LUID          = Column(String(32))
    objectid            = Column(Integer, index=True)
    actiondate          = Column(JSONDateTime)
    actorobjectid       = Column(Integer)
    message             = Column(String(255))
    action              = Column(String(20))
```

- This will create a DateTime column in the database that can be assigned to using a string.
- The ORM knows that this is a date-time value.

The Problem

EXPLAIN an SQLalchemy Query

- What is the query path of a query generated by SQLalchemy.
  - Manual queries can be easily testing using “EXPLAIN” / “EXPLAIN ANALYZE”.
  - SQLalchemy queries only appear in the logs, and criteria and statement must then be combined for testing.
    - Awkward!

Craft Some Magic

EXPLAIN an SQLalchemy Query

```
import pprint
from sqlalchemy import *
from sqlalchemy.ext.compiler import compiles
from sqlalchemy.sql.expression import Executable, ClauseElement, _literal_as_text
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, aliased

class Explain(Executable, ClauseElement):
    def __init__(self, stmt, analyze=False):
        self.statement = _literal_as_text(stmt)
        self.analyze = analyze

@compiles(Explain, 'postgresql')
def pg_explain(element, compiler, **kw):
    text = "EXPLAIN "
    if element.analyze:
        text += "ANALYZE "
    text += compiler.process(element.statement)
    return text
```

Use The Magic
EXPLAIN an SQLalchemy Query

```
if __name__ == '__main__':

    Base = declarative_base()

    class Person(Base):
        __tablename__ = 'person'
        objectid   = Column('company_id', Integer, primary_key=True)
        first_name = Column('firstname', String)
        last_name  = Column('name', String)

    engine = create_engine('postgresql://OGo@127.0.0.1:5432/OGo', echo=False)

    sess = sessionmaker(engine)()

    query = sess.query(Person).filter(and_(Person.objectid > 10000,
                                           Person.last_name.ilike('W%')))

    print 'STATEMENT:\n {0}'.format(query.statement)
    x = sess.execute(Explain(query, analyze=True)).fetchall()
    pprint.pprint(x)
```

See The Magic

EXPLAIN an SQLalchemy Query

```
awilliam@linux-yu4c:~> python explain.py
STATEMENT:
 SELECT person.company_id, person.firstname, person.name
FROM person
WHERE person.company_id > :company_id_1 AND lower(person.name) LIKE
lower(:name_1)
[(u'Seq Scan on person  (cost=0.00..798.07 rows=1276 width=16)
                        (actual time=11.532..63.180 rows=1331 loops=1)',),
 (u"  Filter: ((company_id > 10000) AND ((name)::text ~~* 'W%'::text))",),
 (u'Total runtime: 63.307 ms',)]
```

The Problem

Parent / Child Relationships

- An ORM object exists in a hierarchy with parents and children.
  - This hierarchy should be simple to navigate.
  - Code should be able to enumerate from the relationship.

Craft Some Magic

Parent / Child Relationships

```
from sqlalchemy import *
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, relation, backref

Base = declarative_base()

class Task(Base):
    __tablename__ = 'job'
    objectid   = Column('job_id', Integer,  primary_key=True)
    name       = Column('name', String)
    parentid   = Column('parent_job_id', Integer, ForeignKey('job.job_id'))

    children = relation('Task',
                        backref=backref("parent",
                                        remote_side='job.c.job_id'))

    def __repr__(self):
        return '<Task objectid={0} parentid={1}/>'.\
               format(self.objectid, self.parentid
```

Use The Magic

Parent / Child Relationships

```
if __name__ == '__main__':

    engine = create_engine('postgresql://OGo@127.0.0.1:5432/OGo', echo=False)

    session = sessionmaker(engine)()

    query = session.query(Task).filter(Task.parentid != None)

    x = query.all()[0]
    print 'Task', x,
    print 'Parent', x.parent
    print 'Chilren',  x.children
    print 'Parent\'s Children:', x.parent.children
```

See The Magic

Parent / Child Relationships

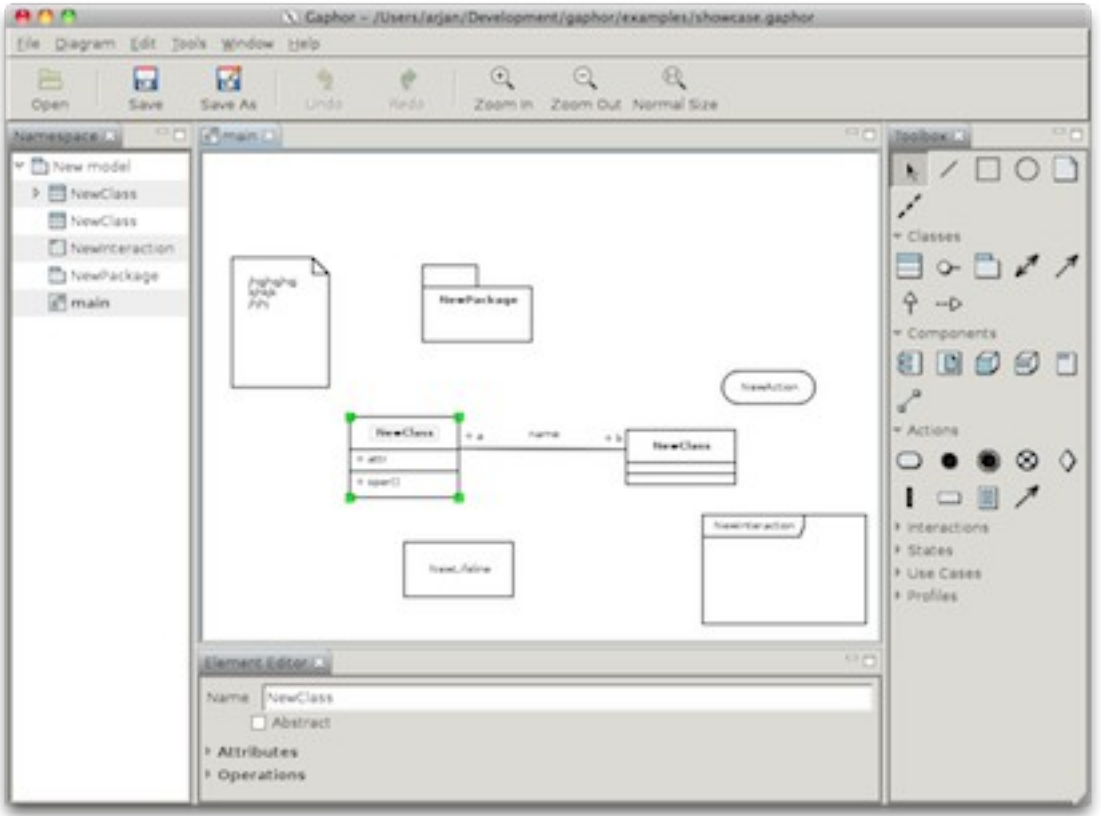
```
awilliam@linux-yu4c:~> python parent.py
Task <Task objectid=14262051 parentid=12975725/> Parent <Task objectid=12975725
parentid=None/>
Chilren []
Parent's Children: [<Task objectid=14262051 parentid=12975725/>]
```

The Problem

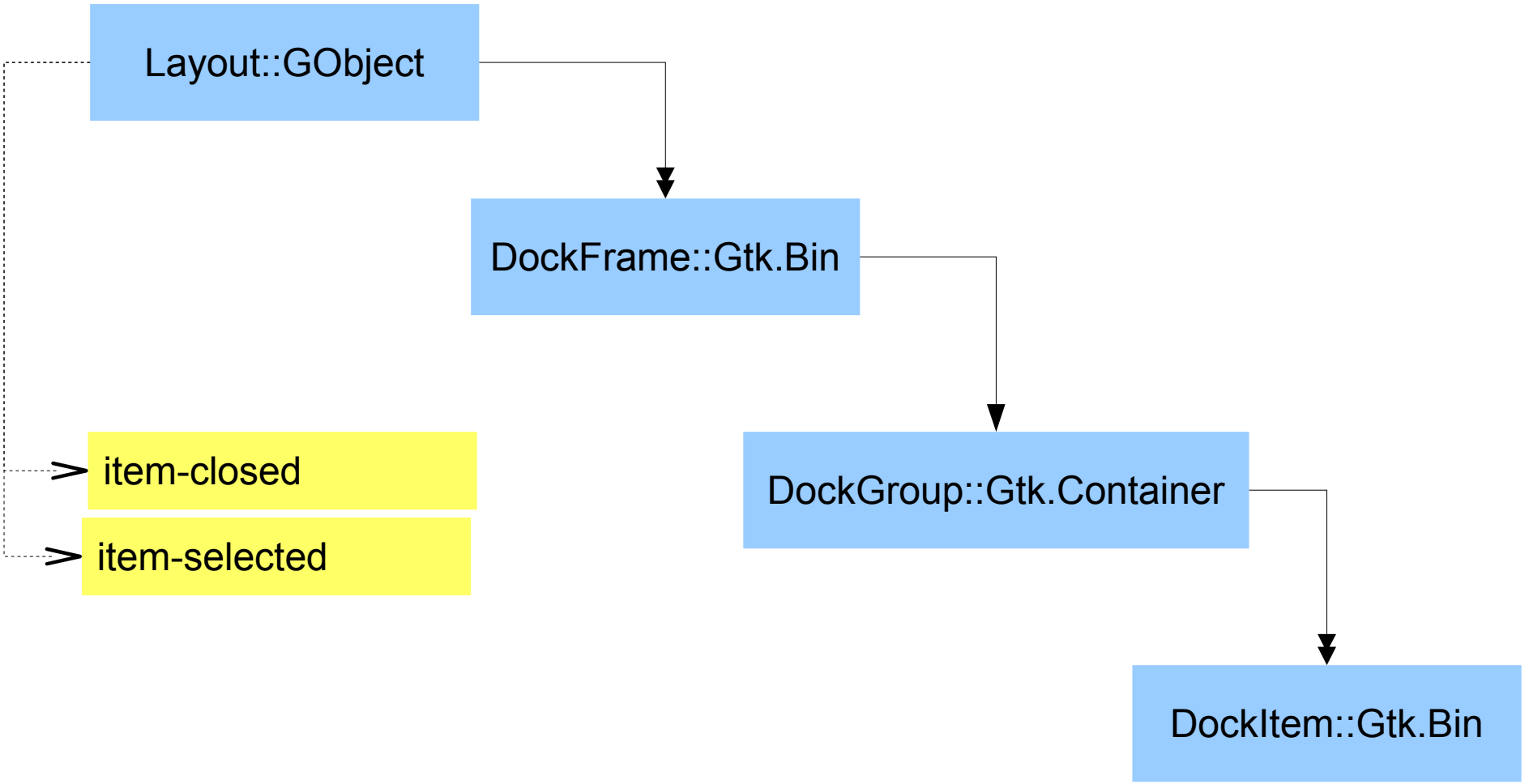
- Create a more flexible / professional UI the provides MDF / docking functionality
  - like in Eclipse, MonoDevelop, etc...

etk.docking

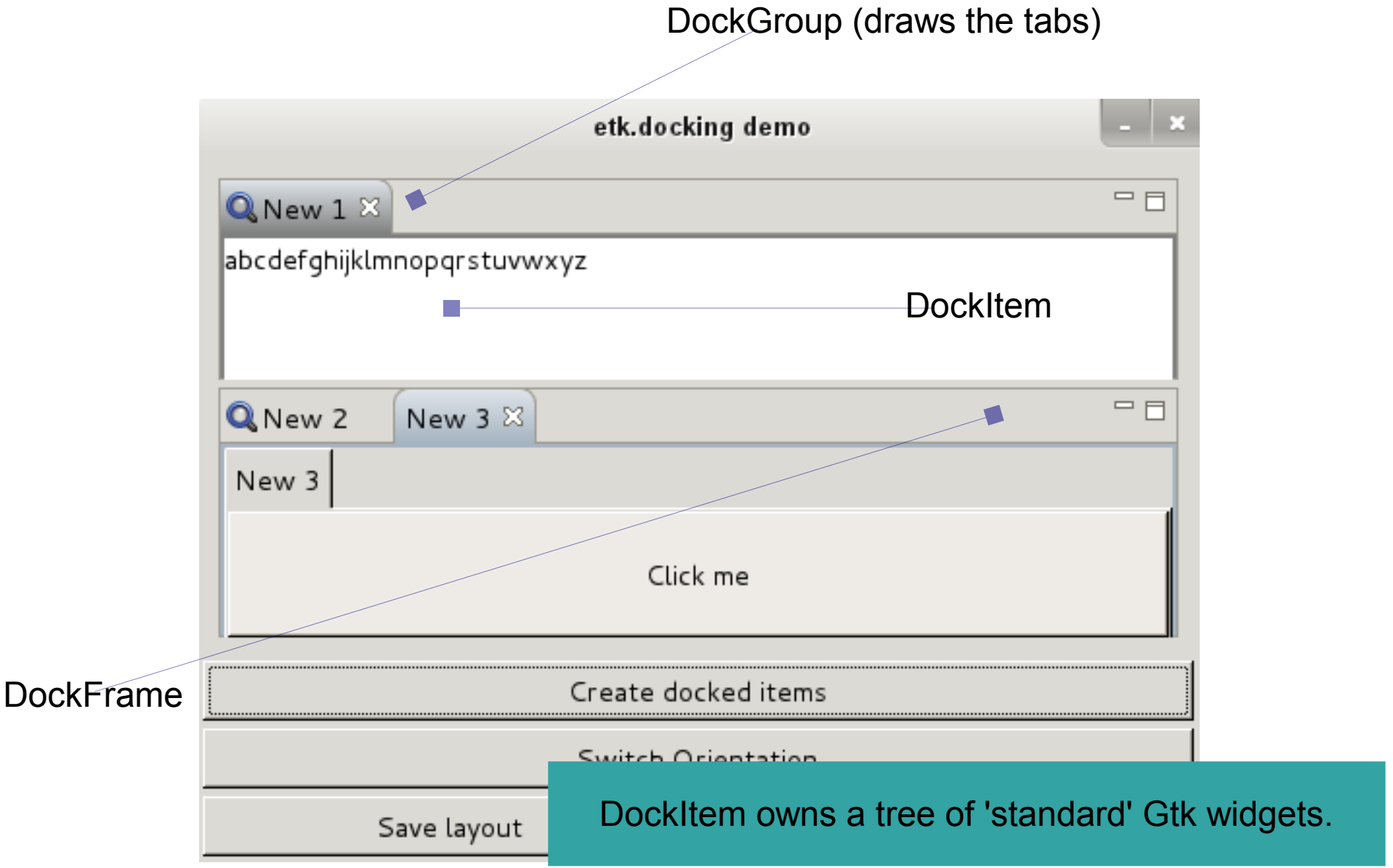
- https://github.com/dieterv/etk.docking/
- Became Beta on 2011-05-08
- Primary Developer: Dieter Verfaillie
- License: LGPL
- Used in Gaphor



Widgets


Layout::GObject
DockFrame::Gtk.Bin
DockGroup::Gtk.Container
DockItem::Gtk.Bin
item-closed
item-selected

Example


DockGroup (draws the tabs)
etk.docking demo
New 1
abcdefghijklmnopqrstuvwxyz
DockItem
New 2
New 3
New 3
Click me
DockFrame
Create docked items
Switch Orientation
Save layout
DockItem owns a tree of 'standard' Gtk widgets.