Tryton Crash Course

N. Évrard

Installing tryton

Prerequisites

Initializing tryton

A basic module

A minimal module

Defining some object / tables

Defining a tree and a form view

Default values

Some more advanced features

Workflows

Buttons

Function fields

Wizards

Extending pre-existing tryton objects

Extending models

Extending views

A tryton module creation crash course

N. Évrard

B_2CK

Tryton Unconference Leipzig 2014

Outline

Installing tryton
Prerequisites
Initializing tryton
A basic module
A minimal module
Defining some object / tables
Defining a tree and a form view
Default values
Some more advanced features
Workflows
Buttons
Function fields
Wizards
Extending pre-existing tryton objects
Extending models
Extending views

1. Installing tryton
   - Prerequisites
   - Initializing tryton
2. A basic module
   - A minimal module
   - Defining some object / tables
   - Defining a tree and a form view
   - Default values
3. Some more advanced features
   - Workflows
   - Buttons
   - Function fields
   - Wizards
4. Extending pre-existing tryton objects
   - Extending models
   - Extending views

Python good practices (pip & virtualenv)

Installing tryton
Prerequisites
Initializing tryton
A basic module
A minimal module
Defining some object / tables
Defining a tree and a form view
Default values
Some more advanced features
Workflows
Buttons
Function fields
Wizards
Extending pre-existing tryton objects
Extending models
Extending views

pip is THE tool for installing and managing Python packages.
virtualenv is THE tool used to create isolated Python environment.

- create isolated Python environments
  - Do not mix different version of your libraries / applications
  - Installation of packages in the user `$HOME`
- `pip install virtualenv` (or your distro package manager)
- `virtualenv --system-site-packages .venv/trytond`
- `source .venv/trytond/bin/activate`

hgnested is a mercurial extension. The tryton project use it to easily apply the same command on nested repositories.

Installing trytond

Tryton Crash Course

N. Évrard

Installing tryton
Prerequisites
Initializing tryton
A basic module
A minimal module
Defining some object / tables
Defining a tree and a form view
Default values
Some more advanced features
Workflows
Buttons
Function fields
Wizards
Extending pre-existing tryton objects
Extending models
Extending views

Example

```
$ hg nclone http://hg.tryton.org/trytond -b 3.4
$ cd trytond
$ pip install -e .
$ pip install vobject
```

Setting up a trytond config file

Tryton Crash Course
N. Évrard
Installing tryton
Prerequisites
Initializing tryton
A basic module
A minimal module
Defining some object / tables
Defining a tree and a form view
Default values
Some more advanced features
Workflows
Buttons
Function fields
Wizards
Extending pre-existing tryton objects
Extending models
Extending views

Here is a minimal example of a configuration file. You should save it in `$HOME/.trytond.conf`

Example

```
[database]
path = /home/training/databases
```

We will set the `TRYTOND_CONFIG` environment variable

Example

```
$ export TRYTOND_CONFIG=$HOME/.trytond.conf
```

Initializing a minimal trytond database

Tryton Crash Course

N. Évrard

Installing tryton
- Prerequisites
- Initializing tryton

A basic module
- A minimal module
- Defining some object / tables
- Defining a tree and a form view
- Default values

Some more advanced features
- Workflows
- Buttons
- Function fields
- Wizards

Extending pre-existing tryton objects
- Extending models
- Extending views

Example

```
$ touch ~/databases/test.sqlite
$ ./bin/trytond -d test -u ir res
```

trytond will ask you for the `admin` password at the end of the installation process.

Adding a new modules

Tryton Crash Course

N. Évrard

Installing tryton
Prerequisites
Initializing tryton
A basic module
A minimal module
Defining some object / tables
Defining a tree and a form view
Default values
Some more advanced features
Workflows
Buttons
Function fields
Wizards
Extending pre-existing tryton objects
Extending models
Extending views

In this tutorial we will use a MQ repository in order to progress step by step.

Example

```
$ cd trytond/modules
$ hg init training
$ cd training/.hg
$ hg clone http://hg.tryton.org/training -b 3.4 patches
```

A minimal trytond modules

Tryton Crash Course

N. Évrard

Installing tryton
Prerequisites
Initializing tryton
A basic module
A minimal module
Defining some object / tables
Defining a tree and a form view
Default values
Some more advanced features
Workflows
Buttons
Function fields
Wizards
Extending pre-existing tryton objects
Extending models
Extending views

A minimal trytond modules needs two files:

- `__init__.py` the usual file needed by all python modules
- `tryton.cfg` the file that helps tryton glue together model and view definitions

The content of `tryton.cfg`

Installing tryton
Prerequisites
Initializing tryton
A basic module
A minimal module
Defining some object / tables
Defining a tree and a form view
Default values
Some more advanced features
Workflows
Buttons
Function fields
Wizards
Extending pre-existing tryton objects
Extending models
Extending views

Example

```
[tryton]
version=0.0.1
depends:
    ir
    res
```

Creating a model

Tryton Crash Course

N. Évrard

Installing tryton

Prerequisites

Initializing tryton

A basic module

A minimal module

Defining some object / tables

Defining a tree and a form view

Default values

Some more advanced features

Workflows

Buttons

Function fields

Wizards

Extending pre-existing tryton objects

Extending models

Extending views

Example

```
from trytond.model import ModelSQL

__all__ = ['Opportunity']

class Opportunity(ModelSQL):
    'Opportunity'
    __name__ = 'training.opportunity'
```

Register the model

Tryton Crash Course

N. Évrard

Installing tryton

Prerequisites

Initializing tryton

A basic module

A minimal module

Defining some object / tables

Defining a tree and a form view

Default values

Some more advanced features

Workflows

Buttons

Function fields

Wizards

Extending pre-existing tryton objects

Extending models

Extending views

Example

```
from trytond.pool import Pool
from .opportunity import *

def register():
    Pool.register(
        Opportunity,
        module='training', type_='model')
```

Adding fields to a model

Tryton Crash Course

N. Évrard

Installing tryton

Prerequisites

Initializing tryton

A basic module

A minimal module

Defining some object / tables

Defining a tree and a form view

Default values

Some more advanced features

Workflows

Buttons

Function fields

Wizards

Extending pre-existing tryton objects

Extending models

Extending views

Example

```
class Opportunity(ModelSQL):
    'Opportunity'
    __name__ = 'training.opportunity'
    _rec_name = 'description'
    description = fields.Char('Description',
        required=True)
    start_date = fields.Date('Start Date',
        required=True)
    end_date = fields.Date('End Date')
    party = fields.Many2One('party.party',
        'Party', required=True)
    comment = fields.Text('Comment')
```

Displaying data

Tryton Crash Course

N. Évrard

Installing tryton
- Prerequisites
- Initializing tryton

A basic module
- A minimal module
- Defining some object / tables
- Defining a tree and a form view
- Default values

Some more advanced features
- Workflows
- Buttons
- Function fields
- Wizards

Extending pre-existing tryton objects
- Extending models
- Extending views

To use the presentation layer your model must inherit from `ModelView`

Example

```
class Opportunity(ModelSQL, ModelView):
```

You must also add the xml presentation file in the `tryton.cfg` configuration file

Example

```
xml:
    opportunity.xml
```

Defining a view

Tryton Crash Course

N. Évrard

Installing tryton
- Prerequisites
- Initializing tryton

A basic module
- A minimal module
- Defining some object / tables
- Defining a tree and a form view
- Default values

Some more advanced features
- Workflows
- Buttons
- Function fields
- Wizards

Extending pre-existing tryton objects
- Extending models
- Extending views

- View objects are normal tryton objects (`trytond/ir/ui/view.py`)
- Two kind of view:

  Tree view a list of record

  Form view a view for editing/creating one record

A tree view

Tryton Crash Course

N. Évrard

Installing tryton

Prerequisites

Initializing tryton

A basic module

A minimal module

Defining some object / tables

Defining a tree and a form view

Default values

Some more advanced features

Workflows

Buttons

Function fields

Wizards

Extending pre-existing tryton objects

Extending models

Extending views

Example

```
<record model="ir.ui.view" id="opportunity_view_list">
    <field name="model">training.opportunity</field>
    <field name="type">tree</field>
    <field name="name">opportunity_list</field>
</record>
```

Example

```
<tree string="Opportunities">
    <field name="party"/>
    <field name="description"/>
    <field name="start_date"/>
    <field name="end_date"/>
</tree>
```

A form view

Tryton Crash Course

N. Évrard

Installing tryton

Prerequisites

Initializing tryton

A basic module

A minimal module

Defining some object / tables

Defining a tree and a form view

Default values

Some more advanced features

Workflows

Buttons

Function fields

Wizards

Extending pre-existing tryton objects

Extending models

Extending views

Example

```
<record model="ir.ui.view" id="opportunity_view_form">
    <field name="model">training.opportunity</field>
    <field name="type">form</field>
    <field name="name">opportunity_form</field>
</record>
```

Example

```
<form string="Opportunity">
    <label name="party"/>
    <field name="party"/>
    <label name="description"/>
    <field name="description"/>
    <label name="start_date"/>
    <field name="start_date"/>
    <label name="end_date"/>
    <field name="end_date"/>
    <separator name="comment" colspan="4"/>
    <field name="comment" colspan="4"/>
</form>
```

Adding default values

Tryton Crash Course

N. Évrard

Installing tryton

Prerequisites

Initializing tryton

A basic module

A minimal module

Defining some object / tables

Defining a tree and a form view

Default values

Some more advanced features

Workflows

Buttons

Function fields

Wizards

Extending pre-existing tryton objects

Extending models

Extending views

Create a method in the object with the name `default_<field_name>`

Example

```
@staticmethod
def default_start_date():
    pool = Pool()
    Date = pool.get('ir.date')
    return Date.today()
```

Adding workflow to object

Tryton Crash Course

N. Évrard

Installing tryton
- Prerequisites
- Initializing tryton

A basic module
- A minimal module
- Defining some object / tables
- Defining a tree and a form view
- Default values

Some more advanced features
- Workflows
- Buttons
- Function fields
- Wizards

Extending pre-existing tryton objects
- Extending models
- Extending views

Your object should inherit from `Workflow`

Example

```
class Opportunity(Workflow, ModelSQL, ModelView):
```

It must have a state field:

Example

```
    state = fields.Selection([
            ('opportunity', 'Opportunity'),
            ('converted', 'Converted'),
            ('lost', 'Lost'),
            ], 'State', required=True,
        readonly=True, sort=False)
```

Defining a workflow

Tryton Crash Course

N. Évrard

Installing tryton

Prerequisites

Initializing tryton

A basic module

A minimal module

Defining some object / tables

Defining a tree and a form view

Default values

Some more advanced features

Workflows

Buttons

Function fields

Wizards

Extending pre-existing tryton objects

Extending models

Extending views

A workflow is composed of transitions:

Example

```
@classmethod
def __setup__(cls):
    super(Opportunity, cls).__setup__()
    cls._transitions |= set((
        ('opportunity', 'converted'),
        ('opportunity', 'lost'),
        ))
```

Defining transition method

Tryton Crash Course

N. Évrard

Installing tryton

Prerequisites

Initializing tryton

A basic module

A minimal module

Defining some object / tables

Defining a tree and a form view

Default values

Some more advanced features

Workflows

Buttons

Function fields

Wizards

Extending pre-existing tryton objects

Extending models

Extending views

Each transition must have a class method:

Example

```
@classmethod
@Workflow.transition('converted')
def convert(cls, opportunities):
    pool = Pool()
    Date = pool.get('ir.date')
    cls.write(opportunities, {
            'end_date': Date.today(),
            })
```

Adding buttons in view

Tryton Crash Course

N. Évrard

Installing tryton
- Prerequisites
- Initializing tryton

A basic module
- A minimal module
- Defining some object / tables
- Defining a tree and a form view
- Default values

Some more advanced features
- Workflows
- Buttons
- Function fields
- Wizards

Extending pre-existing tryton objects
- Extending models
- Extending views

Now we need to add buttons in the view

Example

```
<button name="convert" string="Convert" icon="tryton-go-next"/>
```

The method must be "button decorated" to be callable and defined.

Example

```
@classmethod
def __setup__(cls):
    ...
    cls._buttons.update({
            'convert': {},
            'lost': {},
            })

@classmethod
@ModelView.button
@Workflow.transition('converted')
def convert(cls, opportunities):
    ...
```

Defining a simple function field

Tryton Crash Course

N. Évrard

Installing tryton
- Prerequisites
- Initializing tryton

A basic module
- A minimal module
- Defining some object / tables
- Defining a tree and a form view
- Default values

Some more advanced features
- Workflows
- Buttons
- Function fields
- Wizards

Extending pre-existing tryton objects
- Extending models
- Extending views

A function field is a field that is computed into python its data is not persistently store into the database.

Example

```
duration = fields.Function(fields.Integer('Duration'), 'get_duration')

def get_duration(self, name=None):
    if not self.start_date or not self.end_date:
        return None
    return (self.end_date - self.start_date).days
```

Any tryton type can be used. Note also that since this signature is the same as the one of an `on_change_with` then the same function can be used for both.
Of course a `setter` and a `searcher` can also be defined in order to modify or search corresponding data.

Defining a function field operating by batch

Tryton Crash Course

N. Évrard

Installing tryton
- Prerequisites
- Initializing tryton

A basic module
- A minimal module
- Defining some object / tables
- Defining a tree and a form view
- Default values

Some more advanced features
- Workflows
- Buttons
- Function fields
- Wizards

Extending pre-existing tryton objects
- Extending models
- Extending views

The previous example makes one call per record.
Tryton provides a way to compute the values by batch. In order to do so the getter must be a classmethod and it must return a dictionary mapping the `id` to the function value.

Example

```
description_length = fields.Function(fields.Integer('Description Length'),
    'get_description_length')

@classmethod
def get_description_length(cls, opportunities, name):
    cursor = Transaction.cursor()

    opportunity = cls.__table__()
    query = opportunity.select(
        opportunity.id, CharLength(opportunity.description))
    cursor.execute(*query)

    return dict(cursor.fetchall())
```

Adding actions to model

Tryton Crash Course

N. Évrard

Installing tryton
Prerequisites
Initializing tryton
A basic module
A minimal module
Defining some object / tables
Defining a tree and a form view
Default values
Some more advanced features
Workflows
Buttons
Function fields
Wizards
Extending pre-existing tryton objects
Extending models
Extending views

Sometime you want to add functionalities to a model that do not suite the use of a button. For this kind of use case the wizard is your solution.
A wizard is composed of two things:

- A set of views that represent the form to gather the user input
- A "state machine" that define what should be done

Wizard views

Tryton Crash Course

N. Évrard

Installing tryton
- Prerequisites
- Initializing tryton

A basic module
- A minimal module
- Defining some object / tables
- Defining a tree and a form view
- Default values

Some more advanced features
- Workflows
- Buttons
- Function fields
- Wizards

Extending pre-existing tryton objects
- Extending models
- Extending views

Those are standard `ModelView`, you can define `on_change`, `on_change_with` and default values on them.

Example

```
class ConvertOpportunitiesStart(ModelView):
    'Convert Opportunities'
    __name__ = 'training.opportunity.convert.start'
```

Wizard "state machine"

Tryton Crash Course

N. Évrard

Installing tryton
- Prerequisites
- Initializing tryton

A basic module
- A minimal module
- Defining some object / tables
- Defining a tree and a form view
- Default values

Some more advanced features
- Workflows
- Buttons
- Function fields
- Wizards

Extending pre-existing tryton objects
- Extending models
- Extending views

This is a class that inherits from `Wizard`. You'll be able to define different states on it.

Example

```
from trytond.wizard import Wizard, StateView, StateTransition, Button

class ConvertOpportunities(Wizard):
    'Convert Opportunities'
    __name__ = 'training.opportunity.convert'

    start = StateView('training.opportunity.convert.start',
        'training.opportunity_convert_start_view_form', [
            Button('Cancel', 'end', 'tryton-cancel'),
            Button('Convert', 'convert', 'tryton-ok', default=True),
            ])
    convert = StateTransition()

    def transition_convert(self):
        pool = Pool()
        Opportunity = pool.get('training.opportunity')
        opportunities = Opportunity.browse(
            Transaction().context['active_ids'])
        Opportunity.convert(opportunities)
        return 'end'
```

Activating the wizard

Tryton Crash Course

N. Évrard

Installing tryton

Prerequisites

Initializing tryton

A basic module

A minimal module

Defining some object / tables

Defining a tree and a form view

Default values

Some more advanced features

Workflows

Buttons

Function fields

Wizards

Extending pre-existing tryton objects

Extending models

Extending views

Example

```
<record model="ir.action.wizard" id="act_convert_opportunities">
    <field name="name">Convert Opportunities</field>
    <field name="wiz_name">training.opportunity.convert</field>
    <field name="model">training.opportunity</field>
</record>
<record model="ir.action.keyword" id="act_convert_opportunities_keyword">
    <field name="keyword">form_action</field>
    <field name="model">training.opportunity,-1</field>
    <field name="action" ref="act_convert_opportunities"/>
</record>
```

Extending existing objects

Tryton Crash Course

N. Évrard

Installing tryton
- Prerequisites
- Initializing tryton

A basic module
- A minimal module
- Defining some object / tables
- Defining a tree and a form view
- Default values

Some more advanced features
- Workflows
- Buttons
- Function fields
- Wizards

Extending pre-existing tryton objects
- Extending models
- Extending views

Sometimes you want to extend existing objects to add miscellaneous information. Doing so is just a matter of (tryton) inheritance:

Example

```
from trytond.model import fields
from trytond.pool import PoolMeta

__all__ = ['Party']
__metaclass__ = PoolMeta


class Party:
    __name__ = 'party.party'
    opportunities = fields.One2Many(
            'training.opportunity', 'party',
            'Opportunities')
```

Extending existing views

Tryton Crash Course

N. Évrard

Installing tryton
- Prerequisites
- Initializing tryton

A basic module
- A minimal module
- Defining some object / tables
- Defining a tree and a form view
- Default values

Some more advanced features
- Workflows
- Buttons
- Function fields
- Wizards

Extending pre-existing tryton objects
- Extending models
- Extending views

Modifing existing view is done by adding record in the XML files that specify an XPATH and what to do with the resulting node.

Example

```
<record model="ir.ui.view" id="party_view_form">
    <field name="model">party.party</field>
    <field name="inherit" ref="party.party_view_form"/>
    <field name="name">party_form</field>
</record>
```

Example

```
<data>
    <xpath expr="/form/notebook/page[@id='accounting']"
        position="after">
        <page name="opportunities" col="1">
            <separator name="opportunities"/>
            <field name="opportunities"/>
        </page>
    </xpath>
</data>
```