

Introduction à Archetypes

Introduction à Archetypes

- Buts de Archetypes
- Contenu
- Le Schema
- Les Fields
- Les Widgets
- Les « validators »
- Views & Actions
- Un produit très simple
- Conclusions

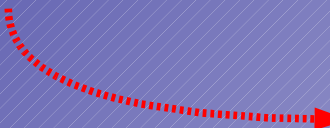
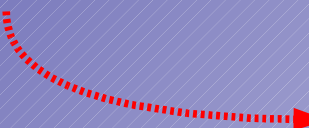
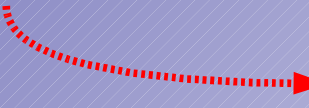
Buts

- Création de types de contenu générique
 - Dans Zope:
 - Zclass (Limité)
 - Produit (Complexe)
 - Dans Plone:
 - Archetypes (- Complexe et - Limité)
- Produit « Skinnable »
- Ajout/édition du produit fini simple pour end user
- Ajout dans Catalogue
- Ajout dans Workflow

Contenu

- Que contient un type de contenu ?
 - Des champs (titre, document, date, ...)
 - Des formulaires pour remplir les champs + validation
 - Des manières de voir le type de contenu

Contenu - Archetypes

- Que contient un type de contenu ?
 - Des champs (titre, document, date, ...)
 Schema + Fields
 - Des formulaires pour remplir les champs + validation
 Widgets + validators
 - Des manières de voir le type de contenu
 Views & Actions

Hiérarchie

Produit

Classe

Schema

Fields

Widget



Le Schema

- Défini une liste de fields (champs)
- Chaque field contient un seul widget
- Le schema va définir le type de contenu
- Peut faire des sommes de schema

- Exemple:

```
schema = BaseSchema + Schema(  
    StringField(  
        name='id',  
        required=0,  
        searchable=1,  
        mode='r',  
        widget=StringWidget(  
            description="The number",  
            label="number"  
        )  
    ),  
    BooleanField(  
        name='recommended',  
        accessor="isRecommended",  
        searchable=1,  
        default=0,  
        write_permission=CMFCorePermissions.ReviewPortalContent,  
        widget=BooleanWidget(  
            description="",  
            label="Recommended Project",  
        )  
    ),  
)
```

utiliser un éditeur intelligent! 

Les Fields

- Un champ est une partie d'information d'un type de contenu
- Un field contient des attributs et le widget lui correspondant
- Créer ses propres fields (Simple!)
- Liste des types de fields de base:

| Nom | Type | Widget Associé | Description |
|-----------------|----------------------|-----------------|--|
| BooleanField | Booleen | ComputedWidget | Champs Vrai ou Faux |
| DateTimeField | Date et Heure | CalendarWidget | Stocker des dates ou des heures |
| FileField | Fichiers | FileWidget | Stocker des fichiers textes, word, Openoffice, ... |
| FixedPointField | Nombres à point fixe | DecimalWidget | Stocker des données numériques |
| FloatField | Décimaux | DecimalWidget | Stocker des décimaux |
| ImageField | Image | ImageWidget | Stocker des images + redimensionnement |
| IntegerField | Entier | StringWidget | Stocker des entiers |
| LinesField | Listes | LinesWidget | Stocker des listes |
| PhotoField | Image | PhotoWidget | Stocker des images |
| ReferenceField | Reference | ReferenceWidget | Faire des références entre objets |
| StringField | String | StringWidget | Stocker des strings (<100 mots) |
| TextField | String | StringWidget | Stocker des strings (>100 mots) + transformations |

Les Fields - Attributs

| Nom | Description | Exemple |
|-------------------|---|---|
| accessor | Nom de la méthode utilisée pour obtenir la valeur du champ avec transformation possible | getFieldValues |
| default | valeur par défaut du champ | foobar |
| default_method | méthode pour accéder à l'info dans le champ | getSpecialDescription |
| edit_accessor | nom de la méthode pour obtenir la valeur sans transformation | getRawValue |
| enforceVocabulary | Utilise uniquement le vocabulaire défini | True/False |
| name | Nom unique du champ | |
| mode | Protection en lecture - écriture | lecture seule: r / écriture seule: w / lecture écriture: rw |
| multiValued | Plusieurs valeurs possibles dans le champ (liste) | True/False |
| mutator | Nom de la méthode qui modifier la valeur du champ | setFieldValue |
| primary | Si activé corps du document envoyé par FTP et WebDAV | True/False |
| required | champ obligatoire ? | True/False |
| schemata | Place le champ dans un groupe de champs | |
| isMetadata | champ considéré comme métadate | True/False |
| searchable | Contenu du champ disponible lors d'une recherche? | True/False |
| validators | Nom du validateur (cfr plus loins) | isInt |
| vocabulary | Liste des valeurs utilisées | ['Chimay','Orval','Westmalle'] |
| storage | Endroit où sera stockée l'information | SQLStorage, AttributeStorage (default) |
| widget | Le widget utilisé (cf plus loins) | BooleanWidget |

+ attributs spécifiques aux différents fields
Cf [Products/Archetypes/Field.py](#)

Les Widgets

- Comment représenter les données du champ?
- Chacun a une représentation en html
- Doit être approprié pour type field!
- Ecrivez vos Widgets (Simple!)

| Nom | Description | Attributs Supplémentaires |
|----------------------|--|---------------------------------------|
| Boolean | 2 CheckBox | |
| CalendarWidget | Calendrier Plone | |
| ComputedWidget | Valeur en HTML | |
| DecimalWidget | Input text | size |
| EpozWidget | Editeur RichText Epoz | format, rows, mode, cols |
| FileWidget | Input file | |
| IdWidget | Input text pour generation des Id | |
| ImageWidget | Affichage de l'image et modification possible de l'image | display_threshold (si < then affiche) |
| IntegerWidget | Input text | size |
| KeywordWidget | Liste de mot clés | |
| LabelWidget | Label d'un formulaire | |
| LinesWidget | Textarea | rows, columns |
| MultiSelectionWidget | Selection | format: select, checkbox |
| PasswordWidget | Input password | |
| RichWidget | Insertion de texte, word, ... + modification | |
| ReferenceWidget | Element sélectionné avec liste de références possibles | |
| SelectionWidget | Selection | format: flex, select, radio |
| StringWidget | Input text | size, maxlength |
| TextAreaWidget | Insertion de texte, word, ... + modification | allowed_content_types |

Les Widgets - Attributs

| Nom | Description | Valeurs possibles |
|-------------------|---|------------------------------------|
| label | Le label qui apparait dans l'interface | Titre du document |
| modes | Le mode dans lequel sera affiché le widget | view ou edit |
| populate | si actif on met la valeur dans le champ (utile pour pwd) | True/False |
| postback | si actif et si erreur champ sera rempli avec l'ancienne valeur | True/False |
| visible | visible dans l'interface de l'utilisateur. Dictionnaire. visible/hidden/invisible | {'view':'visible','edit':'hidden'} |
| description | La description du champ | |
| label_msgid | Message Id pour traduction du label | |
| description_msgid | Message Id pour traduction de la description | |
| i18n_domain | Domaine de traduction | plone |

Plus d'infos sur [Products/Archetypes/Widget.py](#)

Les « Validators »

- Attribut validator d'un champ permet de valider les input d'un champ (~Formulator)
- Basé sur Range ou Regex
- Ecrivez vos propres validator (Simple!)
- Validators de base:

| Nom | Description |
|----------------------------|---|
| inNumericRange | Test si input dans un interval |
| isDecimal | Test si input est un nombre décimal |
| isInt | Test si input est un nombre entier |
| isPrintable | Test si input contient que des chiffres ou des lettres ou des espaces |
| isSSN | Test si input est code de sécurité social américain (9 chiffres) |
| isUSPhoneNumber | Test si input est un numero de téléphone americain (10 chiffres) |
| isInternationalPhoneNumber | Test si input contient au moins 1 numero |
| isZopeCode | Test si input contient 5 ou 9 chiffres |
| isURL | Test si input est une URL valide |
| isEmail | Test si input est un email valide |
| isUnixLikeName | Test si input commence par une lettre et continue avec des lettres ou chiffres ou / ou underscore |
| isEmpty | Test si input est vide |
| isEmptyNoError | Test si input est vide sans message d erreur si pas ok |
| isMaxSize | Test la taille d'un fichier |
| isValidDate | Test si input est une date |

Plus d'infos dans [Products/validation/validators/BaseValidators.py](#)

Views & Actions

- Lier au produit des pages templates
- Archetypes génère automatiquement des pages templates selon le schema
- Mais heureusement => modification possible des pages générées automatiquement
- 2 aspects:

1. Actions définies dans la classe

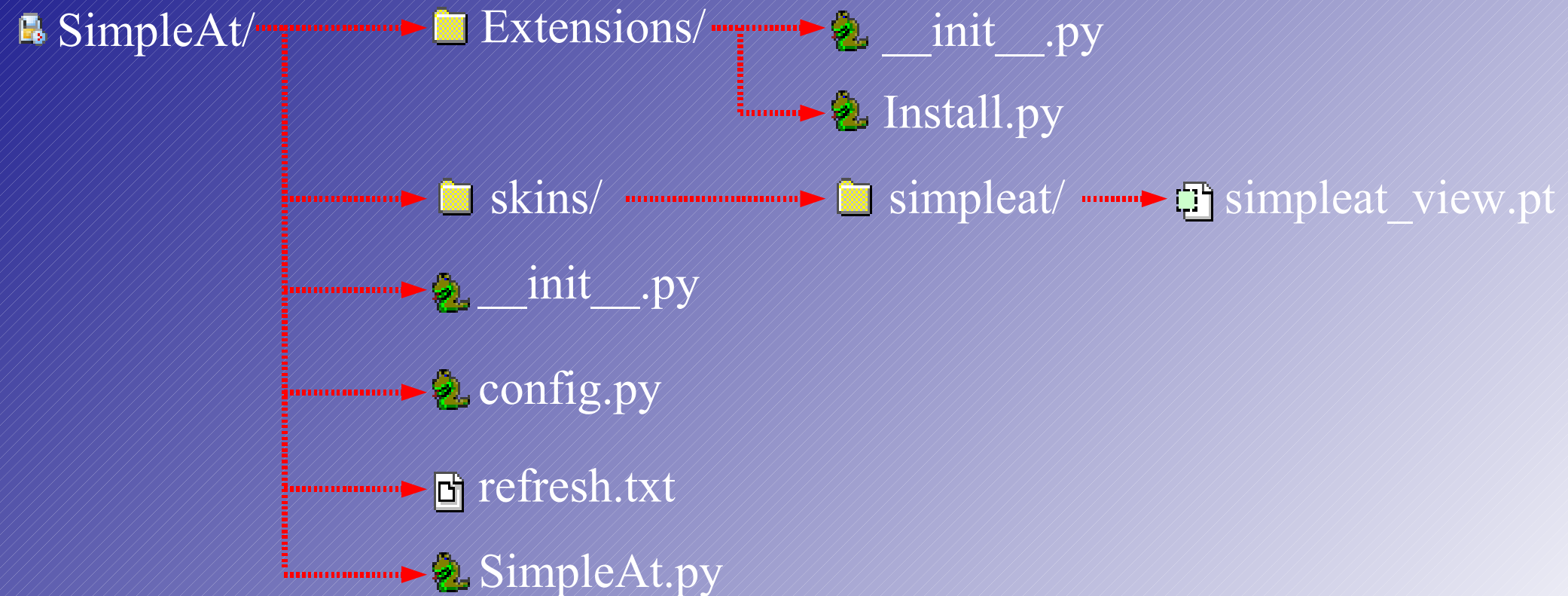
```
actions = ({ 'id': 'view',  
            'name': 'View',  
            'action': 'string:${object_url}/myproduct_view',  
            'permissions': (CMFCorePermissions.View,)  
            },)
```

2. Page Template définies dans skins

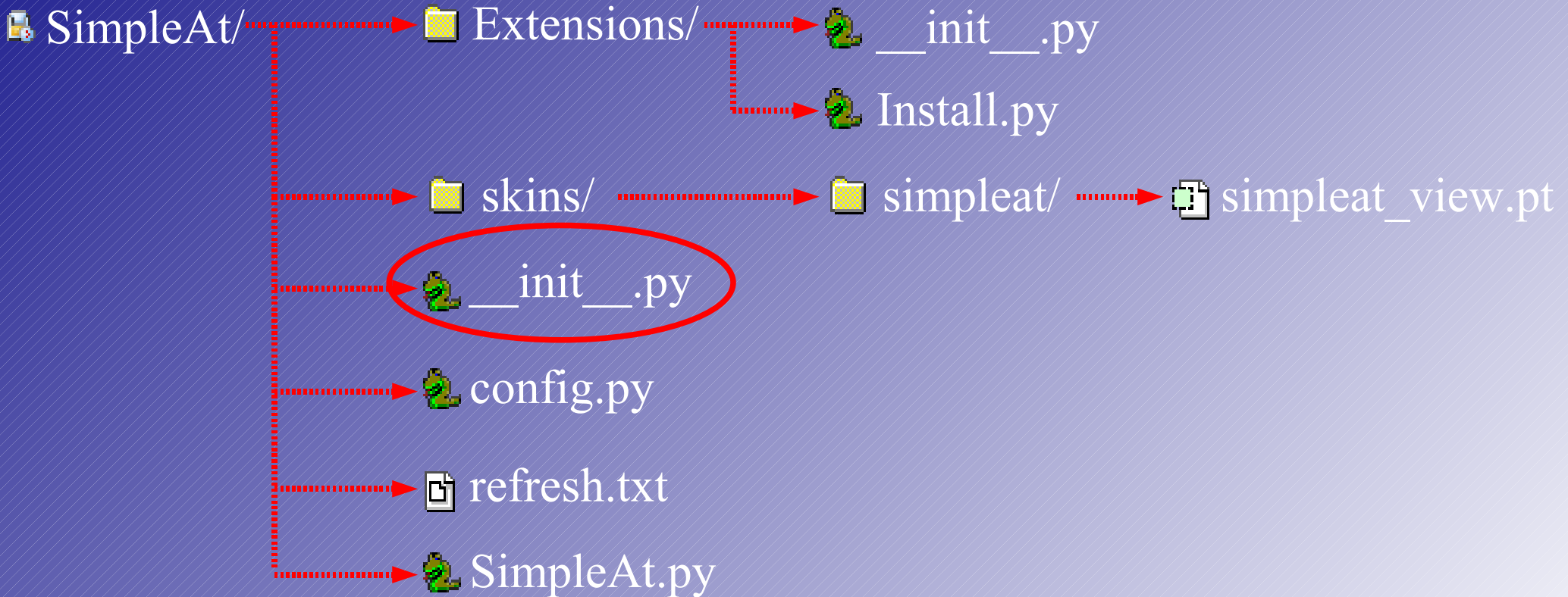
Products/MyProduct/skins/myproduct/myproduct_view.pt

SimpleAT – Un AT très simple

- Hiérarchie sur le FS d'un produit AT:



Hiérarchie





SimpleAt / `__init__.py` - Initialisation

```
from config import *
import Extensions.Install

from Products.CMFCore import utils
from Products.CMFCore.DirectoryView import registerDirectory
from Products.Archetypes.public import *
from Products.Archetypes import listTypes

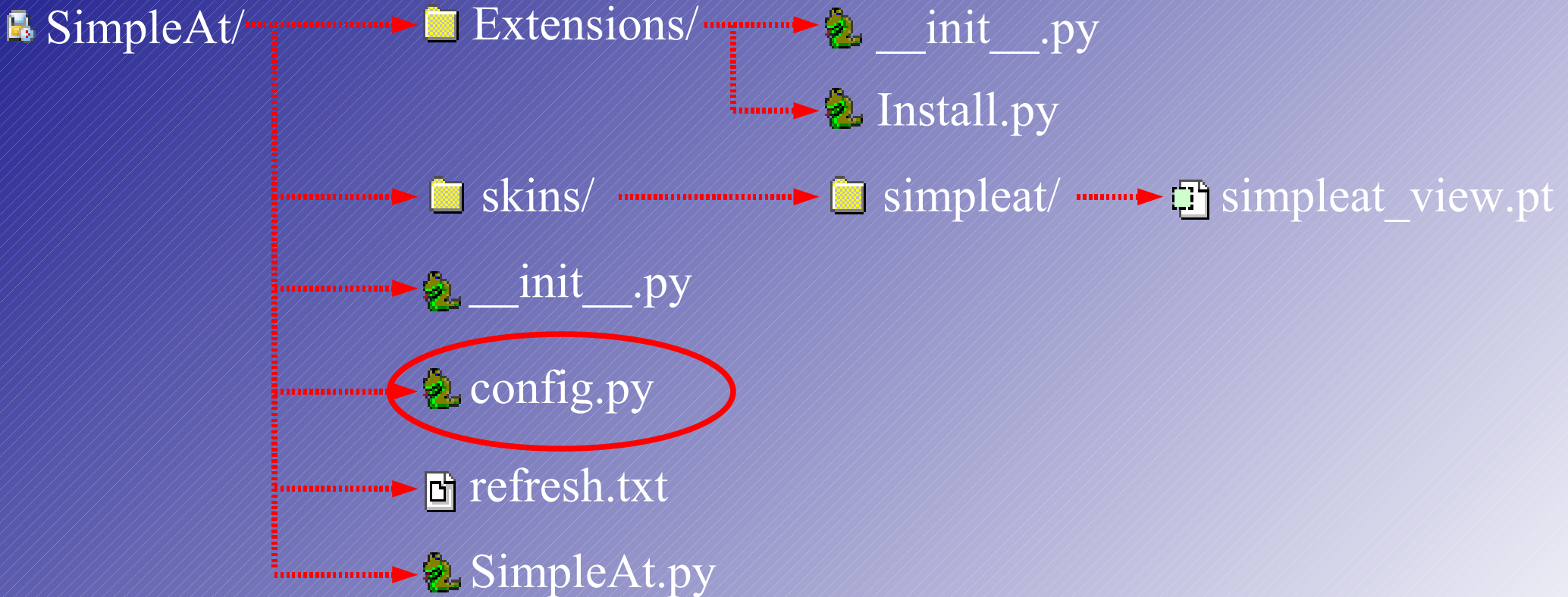
registerDirectory(SKINS_DIR, GLOBALS)           # enregistrement du répertoire skins
install_globals = globals()                   # variables globales python/zope

def initialize(self, context=None):
    # fonction d'initialisation du produit
    import SimpleAt
    content_types, constructors, ftis = process_types(
        listTypes(PROJECTNAME),                # prend toutes les types et classes défini dans produit
        PROJECTNAME)

    # on a toutes les infos nécessaires pour installation du/des types dans content_types, constructors et
    # ftis (Factory Type Informations [icônes, default_view, actions, ...])

    utils.ContentInit(                          # enregistrement du produit auprès du CMF
        PROJECTNAME + ' Content',
        content_types = content_types,          # tout les content types définis dans le produit
        permission = ADD_CONTENT_PERMISSION,    # Nom de la permission d'ajout
        extra_constructors = constructors,      # constructeur par défaut défini par archetypes
        fti = ftis).initialize(self)
```

Hiérarchie





SimpleAt / config.py – Variables globales

Déclaration des variables globales:

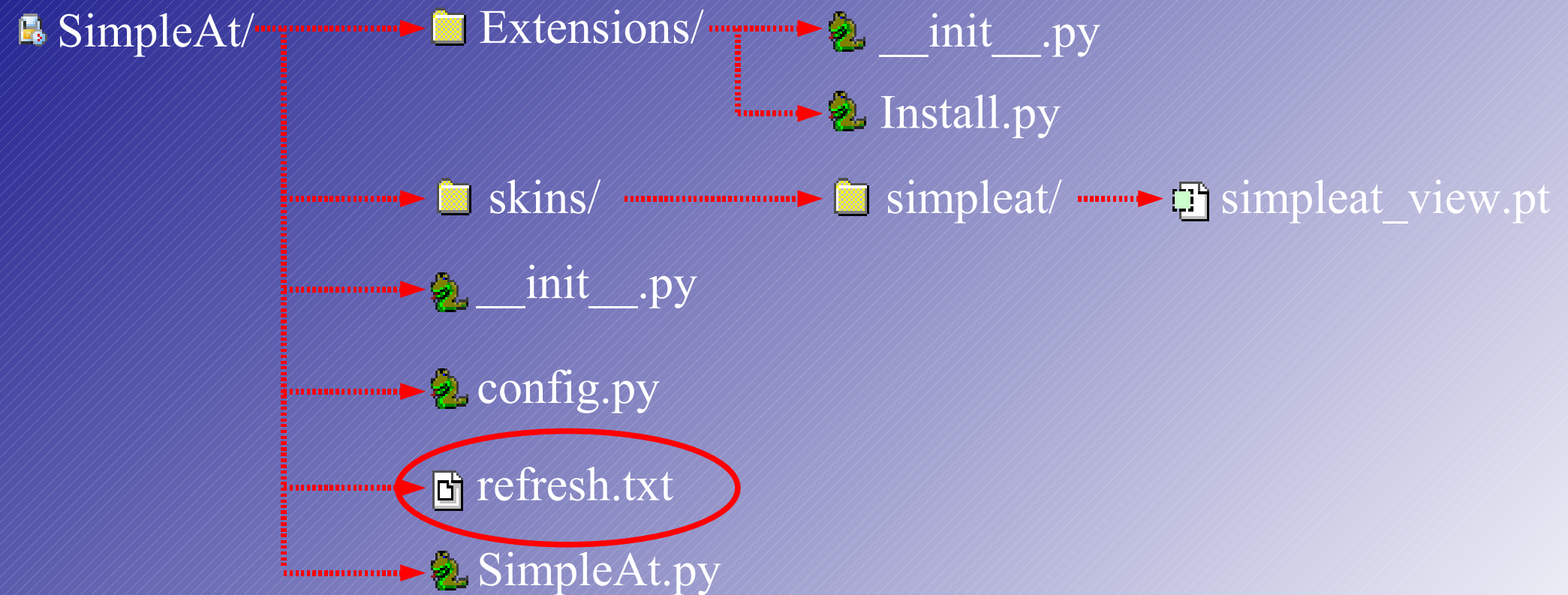
PROJECTNAME = 'SimpleAt'

ADD_CONTENT_PERMISSION = 'SimpleAt: Add SimpleAT Content'

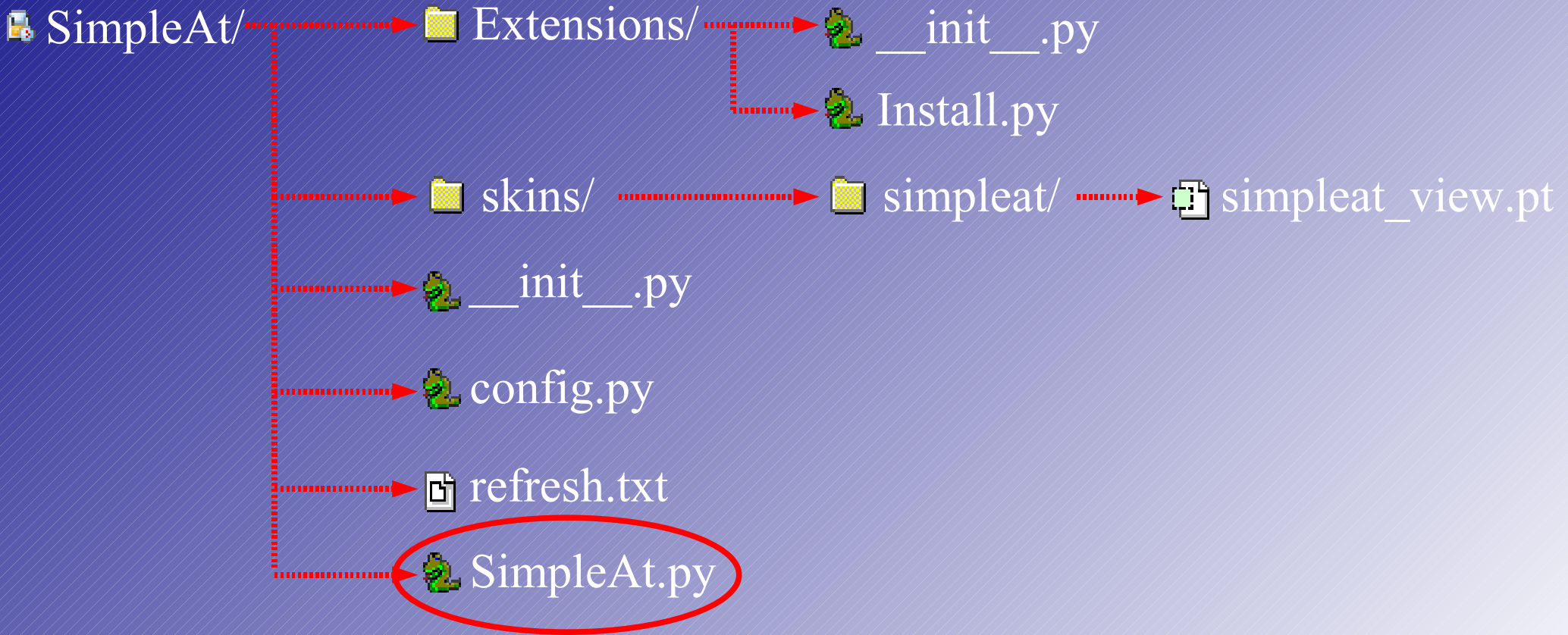
SKINS_DIR = 'skins'

GLOBALS = **globals()**

Hiérarchie



Hiérarchie





SimpleAt / SimpleAt.py – Définition du Content type

```
# Import d'Archetypes
from Products.Archetypes.public import BaseContent
from Products.Archetypes.public import registerType
from Products.Archetypes.public import BaseSchema

# Import de Zope
from AccessControl import ClassSecurityInfo

# Import des valeurs globales
from config import PROJECTNAME

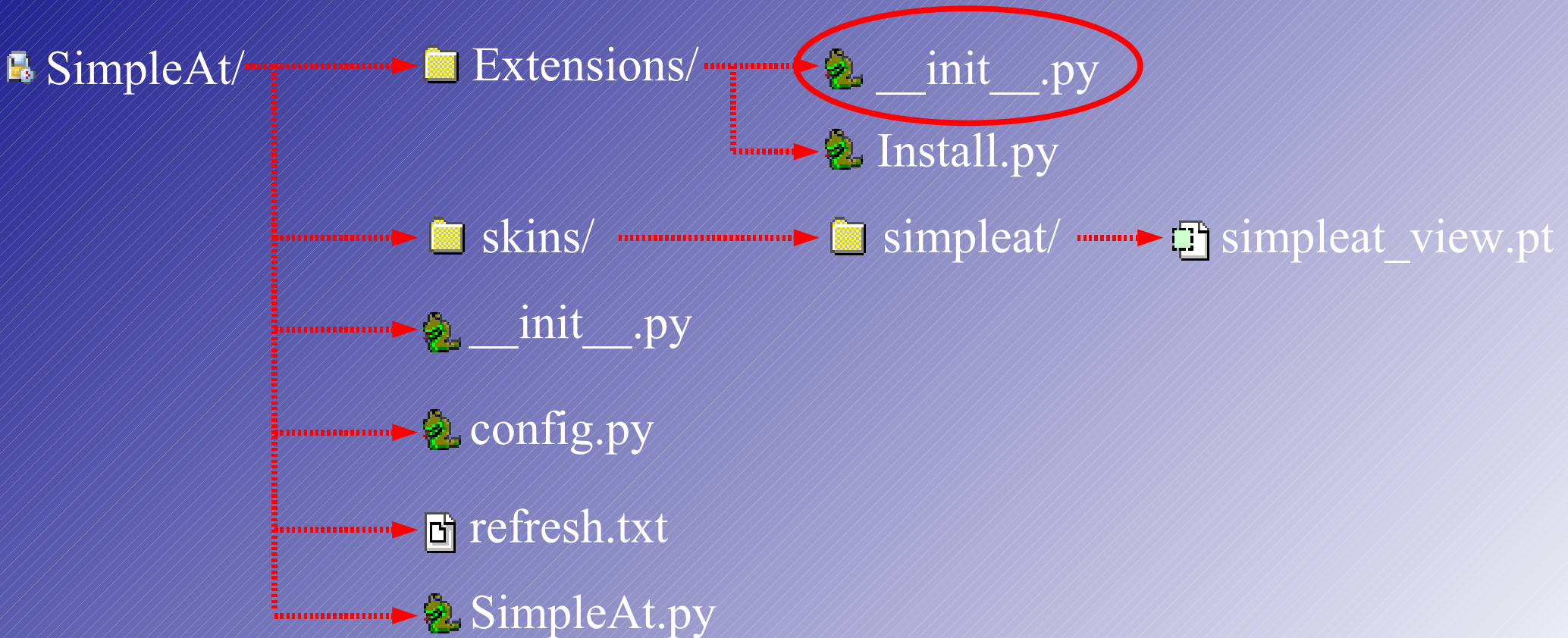
class SimpleAt(BaseContent):
    "Un simple document"
    # déclaration de la gestion des permissions/de la sécurité des objets dans zope
    security = ClassSecurityInfo()

    # implémente les classes dont on hérite
    __implements__ = (BaseContent.__implements__)

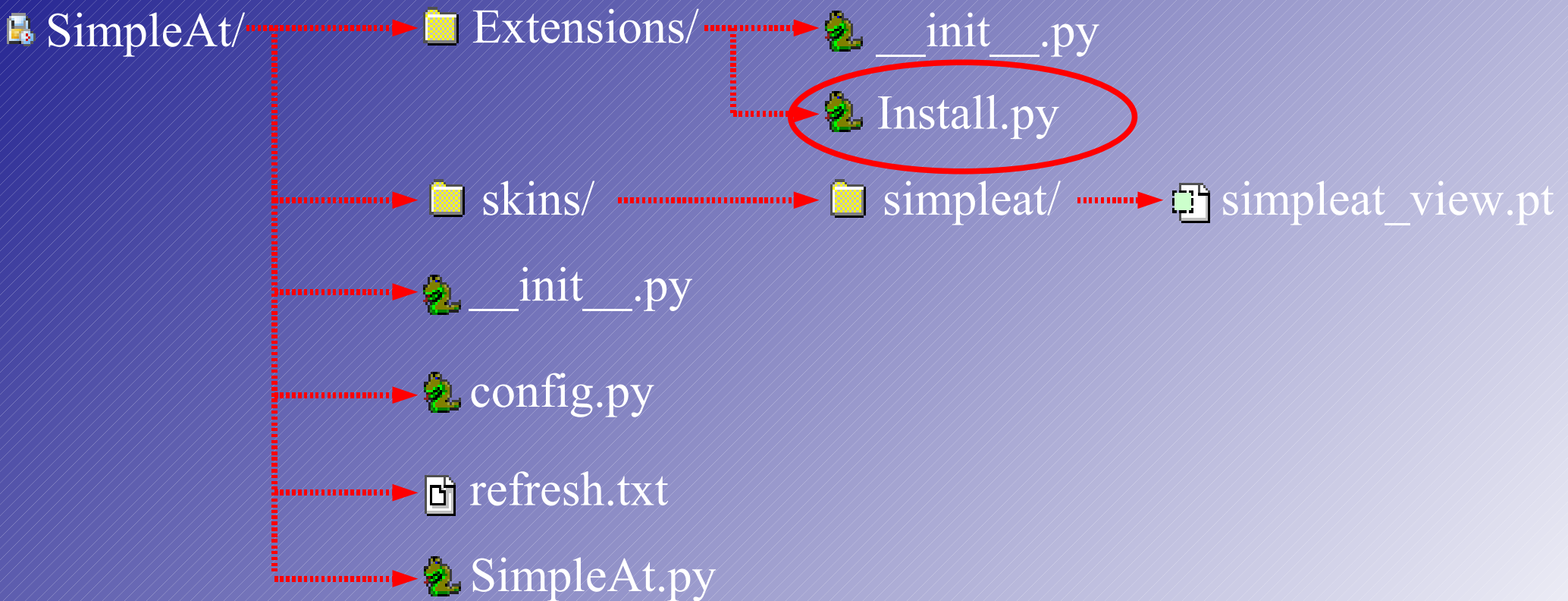
    # définit le schema
    schema = BaseSchema

# enregistrement de la classe auprès de Archetypes – cf __init__
registerType(SimpleAt,PROJECTNAME)
```

Hiérarchie



Hiérarchie





SimpleAt / Extensions / Install.py QuickInstaller Registration

```
from Products.SimpleAt.config import PROJECTNAME, GLOBALS
from Products.Archetypes.public import listTypes
from Products.Archetypes.Extensions.utils import installTypes, install_subskin

from StringIO import StringIO





def install(self):
    out = StringIO()

    installTypes(self, out,
                 listTypes(PROJECTNAME),
                 PROJECTNAME)

    install_subskin(self, out, GLOBALS)

    print >> out, "Successfully installed %s." % PROJECTNAME
    return out.getvalue()
```

SimpleAt Complet

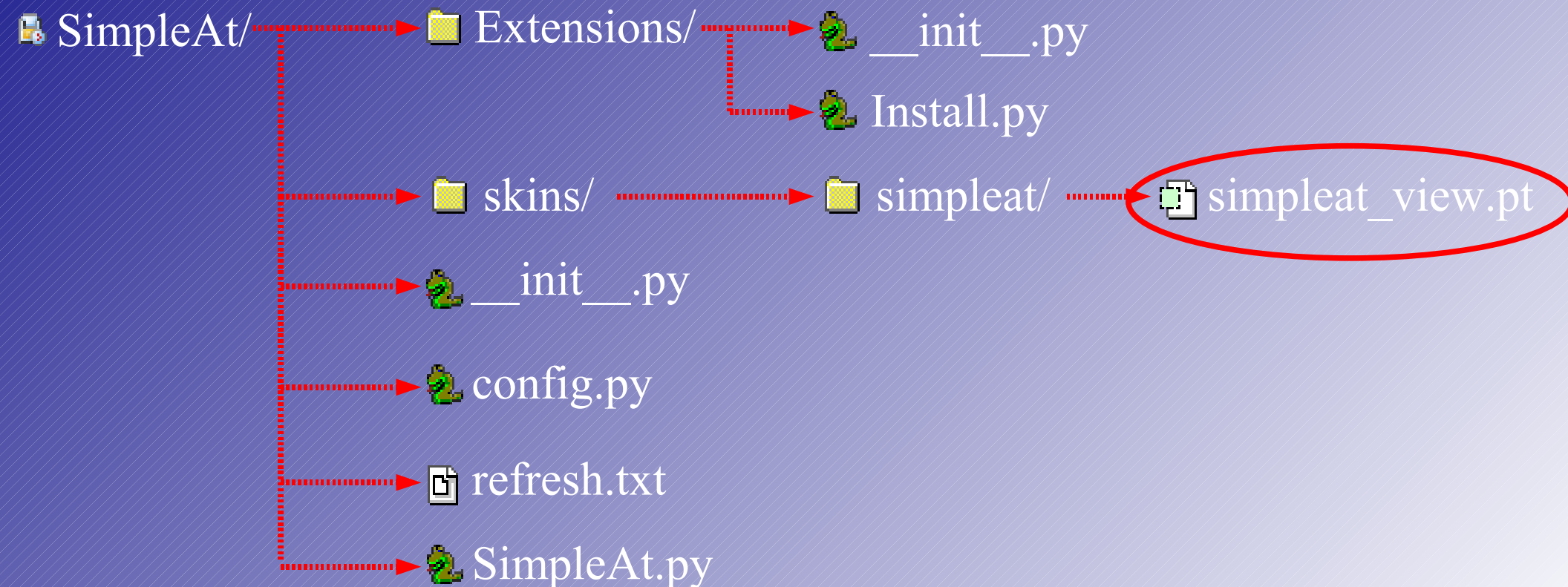
- On a tout ce qu'il faut pour placer le type de contenu dans Plone.
- Installons le...
 - Placer SimpleAt dans \$ZopeInstance/Products
 - Vérifier permissions sur répertoires et fichiers
 - Démarrer Zope
 - Dans Plone  Plone Setup  Add/Remove Products  Install SimpleAt  Lire Install Log
 - Ajouter un SimpleAt dans Plone

A vous de jouer...

- Télécharger le SimpleAt sur <http://web.jfroche.be/cgi-bin/viewcvs.cgi/root.tar.gz?root=SimpleAt&view=tar> et installez le...

Custom View

- Le View du SimpleAt un peu trop simple...
Mettons le à jour...



SimpleAt / skins / simpleat / simpleat_view.pt

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-US"
  lang="en-US"
  metal:use-macro="here/main_template/macros/master">
<body>
<div metal:fill-slot="main">
  <div metal:define-macro="main" tal:omit-tag="">
    <h1><span tal:content="here/title_or_id" /></h1>

    <div tal:repeat="field python: here.Schema().filterFields(isMetadata=0)">
      <fieldset>

        <legend>
          <span tal:content="python: field.getName()"/>
        </legend>
        <span metal:use-macro="python: here.widget(field.getName(), mode='view')"/>

      </fieldset>
    </div>

  </div>
</div>
</body>
</html>
```

Modification du SimpleAt.py

- La PT existe bien dans le produit mais il faut dire au produit que la vue View est maintenant la template que l'on vient de définir:

```
from Products.CMFCore import CMFCorePermissions
```

```
class SimpleAt(BaseContent):
```

```
    "Un simple document"
```

```
    # déclaration de la gestion des permissions/de la sécurité des objets dans zope
```

```
    security = ClassSecurityInfo()
```

```
    # implémente les classes dont on hérite
```

```
    __implements__ = (BaseContent.__implements__)
```

```
    # définit le schema
```

```
    schema = BaseSchema
```

```
    actions = ( { 'id' : 'view',  
                  'name' : 'View',  
                  'action' : 'string:${object_url}/simpleat_view',  
                  'permissions' : (CMFCorePermissions.View,)  
                },  
              )
```

Redémarrage

- Redémarrage de Zope et réinstallation du produit dans QuickInstaller, tous les objets héritent directement de la nouvelle vue... (pas héritage si modification du schema! cf plus loin...)

A vous de jouer...

- Customisez votre propre View en affichant dans la page template le titre du SimpleAt ...
- Utiliser:
 - title_or_id()
 - Title
 - définir votre propre méthode (ex: getTitle()) dans SimpleAt.py
- Faire la même chose en utilisant le portal_skins


Modification du Schema

- Modifions le schema pour introduire un fichier Word dans notre SimpleAt.
- Traduction Word -> HTML par Archetypes & PortalTransforms & (wvWare | win32)

```
#schema = BaseSchema
from Products.Archetypes.public import Schema
from Products.Archetypes.public import FileField
from Products.Archetypes.public import FileWidget

schema = BaseSchema + Schema((
    FileField('Document',
        searchable=1,
        required=1,
        primary=1,
        allowable_content_types=('application/msword',),
        default_output_type='text/html',
        widget=FileWidget( label='Fichier Word',
                           description='Le document à stocker'),
    ),
),)
```

Mise à jour SimpleAt dans Plone

- Redémarrage de Zope (ou refresh Product)
- Il faut mettre à jour le schema pour les objets déjà existant dans Plone:
 - archetype_tool  Update Schema
- Recherche sur le contenu des documents word dans Plone grâce au Catalog & Archetypes & wvWare

Toujours plus loins...

- Content Type Container
- UnitTest – DocTest – Tests Fonctionnels...
- Utilisation de ArchGenXML & ArgoUML
- Ecrire ses propres Widgets...
- Ecrire des validators...
- Changer de système de stockage (SQLStorage, ExternalStorage...)
- Ecrire son Workflow dans le produit (DCWorkflow)

Conclusions

- Outil pour programmeur !
- Rapide à déployer
- Trop de modifications entre versions... :(
- Stockage externe encore un peu complexe...
- Facile pour content type de base !

Apprendre à pêcher

- Site Plone.org:
 - <http://plone.org/documentation/archetypes>
- Tutorial de Raphael Ritz
 - <http://www.neuroinf.de/PloneDevTutorial>
- Le Collective
 - <http://sourceforge.net/projects/collective>
- Lire les sources !