



DSI módulo de vacaciones para OpenERP 4.2.0

Título: DSI módulo vacaciones OpenERP 4.2.0		
Autor: Solid Rock IT - Aitzol Egia Amezua		
Código:	Versión: 1.0	Fecha: 12-03-2009

Índice de contenido

1 – Objetivo del documento	3
2 – Estructura general de módulos.....	4
2.1 – Estructura básica.....	5
2.1.1 – Ficheros Python.....	6
2.1.2 - Vistas.....	7
2.1.3 – Flujos de trabajo.....	8
2.1.4 – Asistentes o wizards.....	9
2.1.5 - Informes.....	10

Solid Rock IT
tu departamento de informática

1 – Objetivo del documento

El objetivo de este manual es proporcionar a los desarrolladores una guía de como se ha implementado el módulo de vacaciones.

Este manual se complementa con el ASI (Análisis sistemas de información) y el manual de usuario.

Solid Rock IT
tu departamento de informática

2 – Estructura general de módulos

Los módulos creados para OpenERP deben estar compuestos por archivos XML, Python (.py) y opcionalmente ficheros RML, SXW (Documento de texto OpenOffice.org 1.0) y XSL.

Cada tipo de archivo juega un papel determinado en el servidor y mientras que los archivos XML y Python son necesarios incluso en los módulos más básicos, los ficheros opcionales de tipo RML, SXW y XSL sólo se utilizan en módulos más avanzados para la creación de informes.

Por un lado, los ficheros XML son empleados para la definición de vistas pudiendo generar desde simples hasta complejos diseños de formularios para obtener una aplicación de gran usabilidad. Los elementos que componen los menús, definición de flujos de trabajo y wizards también se especifican en los archivos xml.

Por otro lado, los archivos .py contienen toda la lógica contenida en el módulo codificada en Python, como la manipulación de objetos, cálculos y generación de informes avanzados.

Los archivos opcionales de tipo SXW y RML se emplean como ya se ha citado anteriormente para generar informes en PDF. El primero permite la generación de plantillas que luego se convierten en RML, los cuales serán empleados por el servidor para incluir información de la base de datos en la plantilla y generar el informe en PDF.

Por último los archivos XSL se utilizan para generar informes más complejos que en el caso anterior, ya que mediante estos ficheros y documentos XML generados desde la base de datos con Python, es posible determinar el estilo del informe en PDF y el lugar que tomará cada valor almacenado en determinadas etiquetas XML.

Una vez presentados los diferentes tipos de ficheros que podemos encontrar en un módulo de OpenERP se detallará a continuación la estructura general de los módulos, prestando especial atención al módulo de gestión de vacaciones.

2.1 – Estructura básica

Todos los módulos incluidos en OpenERP cuentan con una misma estructura básica que contiene los archivos XML y Python:

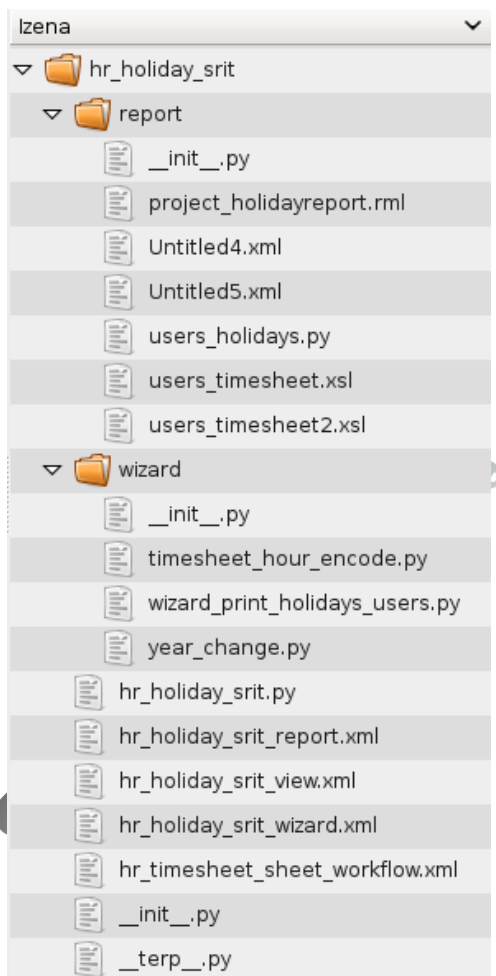


Figura 7.3: Estructura del módulo

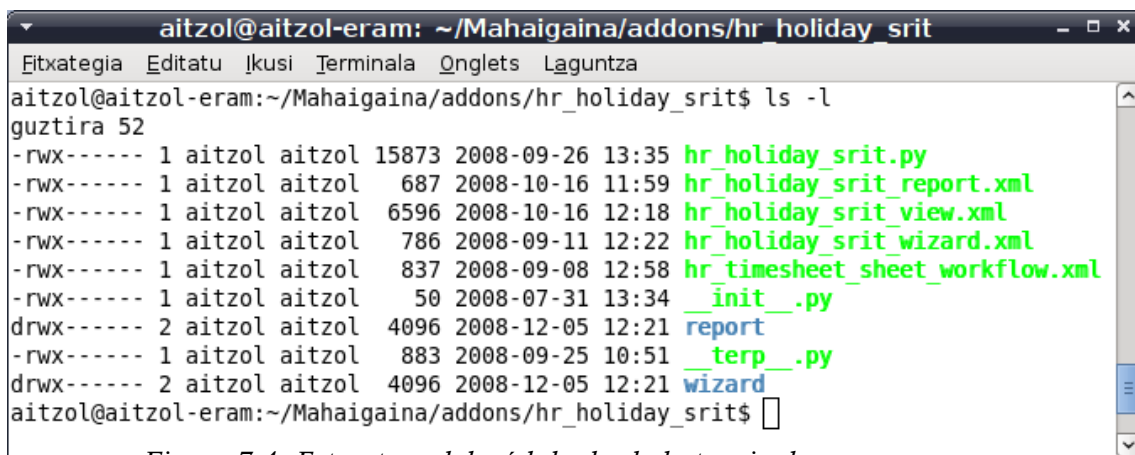


Figura 7.4: Estructura del módulo desde la terminal

En el momento de crear la estructura básica del módulo, hay que seguir ciertas normas para nombrar algunos ficheros XML:

- “nombre_modulo”_view.xml: Archivo XML donde se definen las vistas necesarias en el módulo.
- “nombre_modulo”_workflow.xml: XML donde se definen los flujos de trabajo en el módulo.
- “nombre_modulo”_wizard.xml: XML donde se encuentran definidos los wizards programados dentro de la carpeta “wizard”.
- “nombre_modulo”_report.xml: XML donde se definen la lista de informes existentes en la carpeta “report”.

2.1.1 – Ficheros Python

En primer lugar, los ficheros de Python proporcionan como ya se ha mencionado anteriormente toda la lógica que reside en el módulo. Debido a que Python es un lenguaje orientado objetos, estos archivos contienen las clases y atributos y métodos asociados a ellos. A continuación se muestra un ejemplo:

```
from osv import fields #import necesario
from osv import osv #import necesario
import datetime
import calendar
import time
class hr_holidays(osv.osv): #definición de la clase
    # nombre de la nueva clase o de la heredada
    _inherit = 'hr.holidays'
    # descripción de la clase
    _description = "Holidays"

    #lista de atributos de la clase
    _columns = {
'extra_day':fields.integer("Extra day",size=2,readonly=True),
'planning_id':fields.integer("Planning",size=3,readonly=True),
    }

    #definición de métodos
    def create_days(self, cr, uid, ids, *args):
        # programación en Python
        selfobj=self.browse(cr, uid, ids, None)
        for s in selfobj:
            d=s.date_from1
            dd=s.date_to1
            ...
```

Dentro de la carpeta raíz del módulo existen dos archivos .py especiales denominados “__init__.py” y “__terp__.py” que inicializan los módulos de python para la ejecución del módulo y proporcionan una descripción del módulo junto con la lista de ficheros a modificar durante una posible actualización del módulo.

2.1.2 - Vistas

En lo que a ficheros XML se refiere, éstos definen vistas, elementos del menú, flujos de trabajo, informes y wizards utilizando para ello determinadas etiquetas y atributos.

Para la definición de vistas es necesario en primer lugar definir dentro del archivo "nombre_modulo"_view.xml primero la vista (formulario o árbol), segundo la acción asociada y por último el elemento del menu.

```
<!-- Inicio definición nueva vista -->
<record model="ir.ui.view" id="view_holiday_list">
  <!-- Nombre de la lista -->
  <field name="name">hr.holidays.holidaylist</field>
  <field name="model">hr.employee</field>
  <field name="priority" eval="2"/>
  <!-- Tipo de vista: árbol o formulario-->
  <field name="type">tree</field>
  <field name="arch" type="xml">
  <tree string="Holiday list">
    <!-- Lista de atributos a mostrar en la vista-->
    <field name="name"/>
    <field name="holiday_max"/>
    <field name="total_taken_hol"/>
    <field name="extra_days"/>
    <field name="total_taken_extra"/>
    <field name="total_validated"/>
  </tree>
  </field>
</record>
<!-- Fin nueva vista-->

<!-- Inicio definición de acción-->
<record model="ir.actions.act_window" id="action_holiday_list">
  <field name="name">Holiday list</field>
  <field name="res_model">hr.employee</field>
  <field name="view_type">form</field>
  <field name="view_mode">tree,form</field>
  <!-- el valor view_id debe coincidir con el id de la vista que deseamos asociar a la
acción-->
  <field name="view_id" ref="view_holiday_list"/>
</record>
<!-- Fin definición de acción-->

<!-- Inicio definición elemento de menu-->
<menuitem name="Human Resources/Holidays Request/Resumen vacaciones de
empleados" id="menu_holiday_list" action="action_holiday_list"/>
<!-- El valor del atributo action debe coincidir con el valor del atributo id de la acción que
queremos asociar a este elemento de menu-->
<!-- Fin definición elemento de menu-->
```

Una vez definidos la vista, acción y elemento de menú correctamente será posible que una vez que el usuario haga clic en la opción del menú “Resumen vacaciones de empleados” en la ruta “Human Resources/Holidays Request” se muestre la vista definida anteriormente. Esta operación se deberá repetir hasta crear todas las vistas necesarias en el módulo.

2.1.3 – Flujos de trabajo

Para la creación de flujos de trabajo es necesario crear el fichero “nombre_modulo”_workflow.xml y definir en él flujos de trabajo, roles (rol de usuario con privilegios para ejecutar el flujo de trabajo), actividades y transiciones.

```
<!-- Inicio definición de rol-->
<record model="res.roles" id="HR">
  <!-- Nombre del rol de usuario creado-->
  <field name="name">Timesheets validation</field>
</record>
<!--Fin definición de rol-->

<!-- Inicio definición de flujo de trabajo-->
<record model="workflow" id="wkf_timesheet">
  <field name="name">hr_timesheet_sheet.sheet</field>
  <field name="osv">hr_timesheet_sheet.sheet</field>
  <field name="on_create">True</field>
</record>
<!-- Fin definición de flujo de trabajo-->

<!-- Inicio definición de la actividad-->
<record model="workflow.activity" id="act_new">
<!-- Id del flujo de trabajo asociado-->
  <field name="wkf_id" ref="wkf_timesheet" />
  <field name="name">new</field>
  <field name="kind">function</field>
<!-- Método a ejecutar al realizarse la actividad. Puede tomar como valor un método de la clase-->
  <field name="action">write({'state':'new'})</field>
<!-- Primer estado del flujo de trabajo-->
  <field name="flow_start">True</field>
</record>
<!--Fin definición de la actividad del flujo de trabajo-->

<!-- Inicio definición de la transición-->
<record model="workflow.transition" id="t2">
<!--ID de la actividad precedente-->
  <field name="act_from" ref="act_confirm" />
<!-- ID de la siguiente actividad-->
  <field name="act_to" ref="act_done" />
  <field name="signal">done</field>
<!-- Rol de usuario necesario para poder realizar la transición-->
  <field name="role_id" ref="HR"/>
</record>
```

```
<!-- Fin de la definición de transición-->
```

2.1.4 – Asistentes o wizards

En caso de que en el modulo sean necesarios wizards, éstos deberán ser definidos en el archivo “nombre_modulo”_wizard.xml

```
<!-- Inicio definición de wizard-->
<wizard
  string="Year change"
  <!-- Identificador del wizard de la forma:
  wizard_”nombre_modulo”.”nombre_wizard”-->
  id="wizard_hr_holidays_srit.changeyear"
  model="hr.holidays"
  <!-- Nombre del wizard de la forma:
  “nombre_modulo”.”nombre_wizard”-->
  name="hr_holidays_srit.changeyear"/>
</data>
<!-- Fin definición de wizard-->
```

Mientras que la definición del wizard se encuentra en el archivo que se encuentra en la raíz del módulo, archivo de python (o archivos en caso de tener más de un wizard) se encontrarán dentro de la carpeta “wizard” del módulo.

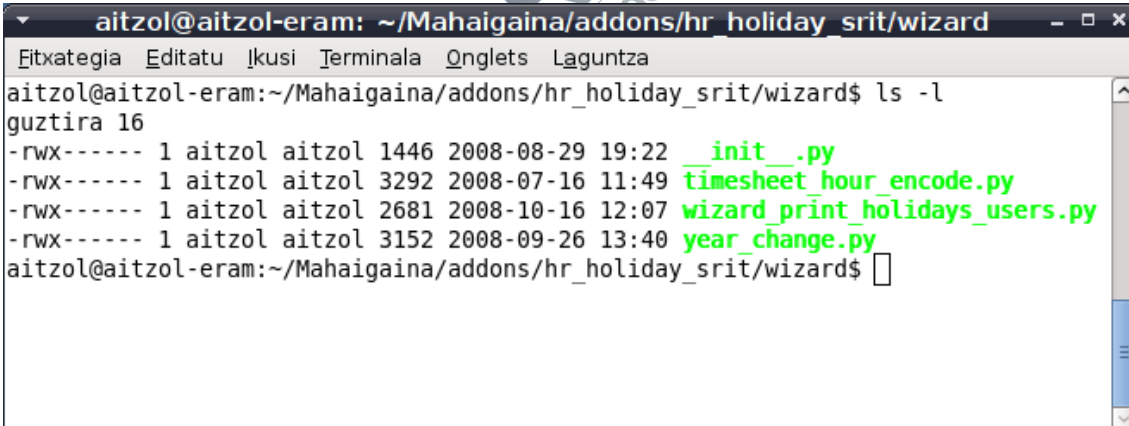


Figura 7.5: Estructura del wizard desde la terminal

En el caso del wizard anteriormente definido, el código correspondiente se encuentra en year_change.py, del que a continuación se muestran los aspectos más importantes a tener en cuenta al crear un nuevo wizard:

```
<!-- Import-->
import wizard          #import necesario
import netsvc         #import necesario
import pooler         #import necesario

import datetime      #import opcional
```

```
<!-- Definición de la clase del wizard y sus métodos-->
class wizard_year_change(wizard.interface):
    <!-- definición de métodos-->
    def _change_year(self, cr, uid, data, context):
        <!-- Programación en Python-->
        cur_month =
        datetime.datetime.today().strftime("%m")
        cur_day = datetime.datetime.today().strftime("%d")
        ...
        <!-- Fin método-->

<!-- Lista de estados-->
states = {
    'init':{
        'actions':[],
        'result':{'type':'form', 'arch':form, 'fields':fields, 'state':[('end','Cancel','gtk-cancel')],
<!-- Próximo estado-->
        ('yearch','Year change',"True)}}
    },
    'yearch': {
        <!-- Acción asociada al estado: método
        _change_year-->
        'actions': [_change_year],
        'result' : {'type' : 'form', 'arch':finalform, 'fields':finalfields, 'state':[ ('end','Close'), ]}
    },
}
```

<!-- nueva instancia de la clase wizard_year_change
El argumento debe coincidir con el valor del atributo name de la definición del wizard
en el XML -->
wizard_year_change('hr_holidays_srit.changeyear')

2.1.5 - Informes

En lo que a informes se refiere se debe tener en cuenta que éstos se pueden realizar de dos formas:

1. Mediante plantillas realizadas en OpenOffice.org y exportadas desde SXW a RML: Este primer método es indicado para informes sencillos de rápida creación.
2. A través de ficheros XML creados en Python y tratados con XSL: Este segundo método esta indicado para informes más complejos que en el primer caso.

Para realizar el informe utilizando el primer método, en primer lugar se debe definir el informe en "nombre_modulo"_report.xml de forma similar a la realizada con flujos de trabajo y vistas.

```

<!-- Inicio definición de informe-->
<report id="project_category_print"
  string="Informe vacaciones"
  model="project.project"
  name="project.holiday"
  <!-- Archivo RML con la plantilla del informe desde la addons/-->
  rml="hr_holiday_srit/report/project_holidayreport.rml"
  menu="True"
  auto="True"/>
<!-- Fin definición de informe-->

```

Una vez que se haya definido el informe, es necesario crear una plantilla del informe con OpenOffice.org y guardarlo en formato SXW. En este documento se crearán las tablas que se mostrarán en el informe y añadir código que luego el servidor de OpenERP sustituirá por datos de la base de datos.

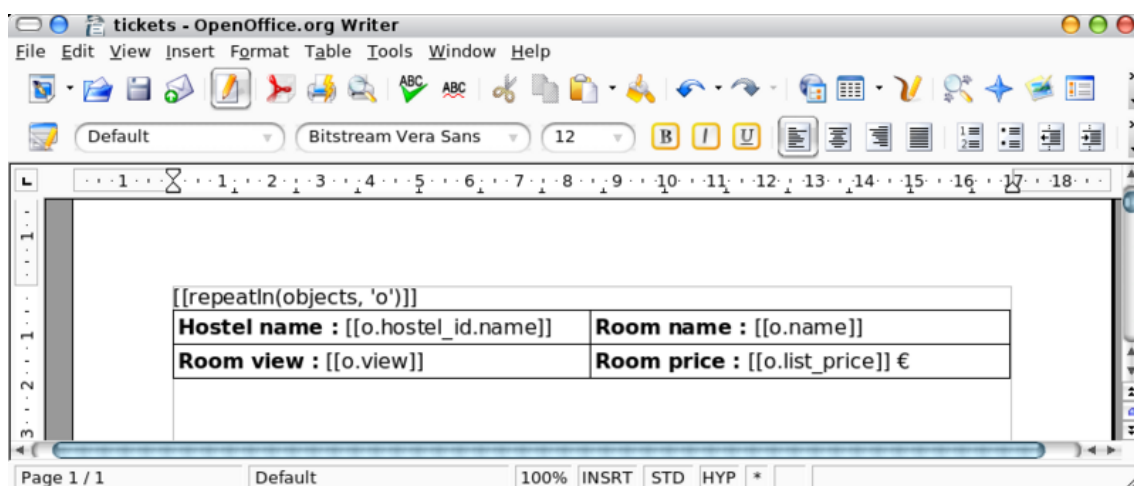


Figura 7.6: Diseño del informe desde OpenOffice

Una vez creado y guardado la plantilla en SXW éste deberá ser exportado a formato RML gracias al programa tiny_sxw2rml (puede ser descargado en el sitio web <http://tinyforge.org/projects/tinyreport/>) ejecutando el siguiente comando:

```

./tiny_sxw2rml.py addons/hr_holiday_srit/report/project_holidayreport.sxw >
addons/hr_holiday_report/report/project_holidayreport.rml

```

Desde el momento en el que el archivo RML se encuentre dentro de la carpeta report del módulo, el informe en PDF ya será visible desde el cliente de OpenERP.

Para que el informe sea realizado utilizando el segundo método, en primer lugar y al igual que en el método anterior se debe definir el informe aunque con un enfoque diferente.

En primer lugar y debido a que el contenido del informe varía en función de los datos introducidos en un wizard, primero se deberá definir el wizard que una vez ejecutado, imprimirá el informe en PDF.

```
<!-- Inicio definición del wizard-->
<wizard
  string="Holiday report"
  model="hr.holidays"
  <!-- Nombre del wizard-->
  name="hr_holidays_srit.holiday"
  menu="True"
  id="action_wizard_user_holiday" />
<!-- Fin definición wizard-->

<!-- Inicio definición de elemento del menu-->
<menuitem
  name="Human Resources/Holidays Request/Calendario
empleados"
  id="menu_action_wizard_user_holiday"
  <!-- El valor de action debe coincidir con el Id del wizard-->
  action="action_wizard_user_holiday"
  type="wizard"/>
<!-- Fin definición de elemento del menu-->
```

Una vez definido el wizard es necesario crear el código asociado a él que se encargará de recoger los datos introducidos por el usuario y llama a la clase encargada de construir el informe.

```
#Inicio definición de clase wizard
class wizard_report(wizard.interface):
    states={
        'init':{
            'actions':[_ get_value],
            'result':{ 'type':'form', 'arch':form, 'fields':fields, 'state':[( 'end','Cancel','gtk-
cancel'),('report','Imprimir', True)]}
        },
        'report':{
            'actions':[],
            'result':{ 'type':'print',
# Nombre del informe 'report':'hr_holidays_srit.holiday_calendar', 'state':'end'}
        }
    }
#Fin clase wizard

# Instancia de la clase, el argumento debe coincidir con # el valor de nombre del wizard en
el XML
wizard_report('hr_holidays_srit.holiday')
```

Para construir el informe se creará una nueva clase dentro de la carpeta report del módulo que será el encargado de recoger los datos de la base de datos y crear un XML a partir del cual y junto con el archivo XSL se formateará el informe en PDF.

```
class report_custom(report_rml):

    def create_xml(self, cr, uid, ids, data, context):
```

```
        start_date = datetime.date(data['form']
[year_from'], data['form']['month_from'],
        if data['form']['month_to']:
            end_date = datetime.date(data['form']
data['form']['month_to'], 1)
        else:
            end_date = start_date
        month_xml = "
        # más código python
        ...

# Instancia de la clase report_custom
report_custom(
    'report.hr_holidays_srit.holiday_calendar', 'hr.holidays', ",
    'addons/hr_holiday_srit/report/users_timesheet.xml')
```

La clase report_custom generará un XML con los datos que se mostrarán en el informe. El XML creado será similar al siguiente ejemplo:

```
<?xml version="1.0" encoding="UTF-8" ?>
<report>
  <month num="8">
    <employee id="3" name="Administrator">
      <time-element date="2" month="8">
        <state>3</state>
      </time-element>
      <time-element date="3" month="8">
        <state>3</state>
      </time-element>
      <time-element date="9" month="8">
        <state>3</state>
      </time-element>
      <time-element date="10" month="8">
        <state>3</state>
      </time-element>
      <time-element date="16" month="8">
        <state>3</state>
      </time-element>
      <time-element date="17" month="8">
        <state>3</state>
      </time-element>
      <time-element date="23" month="8">
        <state>3</state>
      </time-element>
      <time-element date="24" month="8">
        <state>3</state>
      </time-element>
      <time-element date="26" month="8">
        <state>0</state>
      </time-element>
      <time-element date="30" month="8">
```

```
<state>3</state>
</time-element>
<time-element date="31" month="8">
  <state>3</state>
</time-element>
</employee>
</month>
</report>
```

Al crear el XML desde Python con los datos que deben aparecer en el informe y aplicando el estilo XSL correspondiente se generará el PDF con el informe correspondiente, en este caso un calendario con los festivos y vacaciones solicitadas por un conjunto de empleados.

Solid Rock IT
tu departamento de informática