Índice general

OpenERM es la primera especificación «abierta» para el almacenamiento de reportes electrónicos. Las siglas *OERM* hacen referencia a *Open electronic report management* una forma moderna de llamar lo que hace algunos años se conocía como **'COLD'**, *Computer output to laser disk*. Asimismo es la primer implementación oficial de dicha especificación.

CAPÍTULO 1

Un poco de historia

Desde los inicios y en todo tipo de apliciones informaticas, se han generado enormes cantidades de informes, estos reportes terminaban su ciclo de vida en el papel, informes de 80 o 132 columnas, de formato habitualmente tabular. Millones de hojas de papel fueron impresas y distribuídas de esta forma en todo tipo de empresa a lo largo y ancho del mundo. Sin embargo el acceso a la información mediante el «papel», en poco tiempo hizo notar sus limitaciones. Surge una teconolgía, que tuvo su esplendor en las decada del 80 y del 90, se trata de lo que en ese entonces se habia bautizado como *«COLD»*, es decir *«Computer output to laser disk»*. El concepto era simple, se «capturaba» la salida hacia la impresora de los sistemas centralizados, normalmente «Mainframes» o grandes computadores, y esta salida era guardada en archivos electrónicos, almacenada, indizada y distribuída mediante discos ópticos (laser disk), pudiendo luego ser visualizada mediante un software *«COLD»* en PCs u otras mini computadoras.

Este «paradigma», un gran computador central, aplicaciones que explotan la información en reportes de tipo texto, distribución final en papel, tuvo una larga vida. Sin embargo los cambios en la tecnología, el abaratamiento de los costos y otros factores han dejado de lado este «modelo» por otros. Los sistemas «COLD» han ido langideciendo de a poco, sin embargo sigue existiendo un «nicho» importante para este tipo de herramientas: aún hoy existen empresas que apoyan la gestión de su negocio en grande computadores o «Mainframes» y siguen generando enormes cantidades de listados.

CAPÍTULO 2

Estatus del proyecto a abril 2019

Esta es la situación actual del proyecto.

Definiciones:

- Estructura fisica dónde salvar los reportes

Funcionalidad:

- Varios algoritmos de compresión, ver: *openerm.Compressor*
- Cifrado (Spritz y Fernet), ver: *openerm.Cipher*
- Se implementó una clase para el guardado y recuperación de los reportes y sus páginas, ver: *openerm.Database*

Herramientas:

- Spl2oerm - Procesador básico de spooles:
  - Procesamiento de spooles ASCII/EBCDIC de registro de longitud fija, ver: *openerm.SpoolFixedRecordLength*
  - Procesamiento de spooles ASCII/EBCDIC de registro de longitud variable con info de canal, ver: *openerm.SpoolHostReprint*
  - Identificación de páginas, por texto de salto de página o info de canal
  - Identificación simple de reportes por texto encontrado en página
  - Configuración completa del proceso definido en archivo de configuración yaml
  - Salvado de los reportes en el Database final
- readoermdb - Lectura de un database OERM:
  - Lectura de un Database

- Recuperación de reportes
- Lectura de cualquier reporte
- Extracción de páginas

To do:

- Mejorar la identificación de reportes
- Mas opciones, multiples textos a ubicar
- Configurar ventanas de búsquedas (cabeceras/footers)
- Optimizar identificación mediante algún algoritmo mejorado
- Paralelizar el proceso del spool, separando la lectura de páginas de la identificación de archivos y el salvado final
- Captura de datos (fecha, sistema, aplicación, etc) desde el mismo reporte
- Definir mejor los datos adicionales de los reportes. Hoy el único dato real que identifica un reporte el el nombre del mismo

CAPÍTULO 3

Openerm - API

Estos son los módulos principales

3.1 OERM - Especificación v1

3.2 Estructura de un database OpenErm

Un Database OpenErm es un archivo que almacena reportes electrónicos de forma comprimida y/o encriptada. Se representa físicamente por tres archivos básicos:

- <**database**>.oerm: (o «DATA») Es el archivo físico principal, es simplemente un contenedor de bloques. Los bloques son conjuntos arbitrarios y variables de bytes. Los bloques pueden ser de dos tipos
  - Contenedor de metadatos
  - Contenedor de páginas
- <**database**>.cidx.oerm: índice de bloques. Básicamente es una lista con los offsets o posiciones físicas dónde comienza cada bloque del archivo principal.
- <**database**>.ridx.oerm: índice de reportes. Es la lista de offsets o posiciones físicas de los bloques contenedores de metadatos de cada uno de los reportes que se almacenan en el archivo princiapl.

El archivo principal <**database**>.oerm o «DATA» contiene toda la información fundamental, tanto el índice de bloques como el de reporte puede ser regenerado en cualquier momento a partir de archivo «DATA»

```
Estructura de un <database>.oerm

+========+
| "oerm" |              --> "Magic number" (4 bytes)
+====================+
|     Bloque 1       |   --> Bytes (longitud variable)
+====================+
|                    |
|     Bloque 2       |   --> Bytes (longitud variable)
|                    |
+====================+
..
+====================+
|     Bloque  N      |   --> Bytes (longitud variable)
+====================+


Estructura de cualquier Bloque

+========================+
| Long.Total del Bloque |   --> long (4 bytes)
+========================+
| Tipo de Bloque  |         --> int (1 bytes)
+=================+
| Alg. Compresión |         --> int (1 bytes)
+=================+
| Alg. Cifrado    |         --> int (1 bytes)
+========================+
| Long. de los Datos     |   --> long (4 bytes)
+========================+
|                        |
|        Datos           |   --> Longitud variable (datos comprimibles)
|                        |
+========================+
|                        |
|    Datos variables     |   --> (Opcional) Longitud variable (datos NO␣
↪comprimibles)
|                        |
+========================+
```

3.3 Openerm - API

El API básica de OpenErm ofrece por un lado clases públicas, que debieran ser las que utilicemos en nuestras aplicaciones, y otras que son de uso interno a las que debieramos escaparles.

3.3.1 Clases de uso público

OermClient

Un **OermClient** es el objeto que permite conectarse a uno o más repositorios de reportes. Es la clase «oficial» para acceder a los reportes, conceptualmente un **OermClient** ofrece uno o más catálogos, los catálogos son organizaciones lógicas de los reportes físicos

Ver también:

- *openerm.MetadataContainer*
- *openerm.PageContainer*

`class openerm.OermClient.OermClient(`*configfile=None*`)`

Bases: `object`

Clase Cliente para acceso a reportes Oerm. Los reportes OERM se clasifican en: Catalogos y Repositorios. Un catalogo en realidad representa un conjunto lógico de repositorios, los repositorios representan carpetas físicas.

Parámetros `configfile` (*string*) – Nombre del archivo de configuración (Formato YAML)

`add_repo(`*catalog_id*, *path*, *update=False*`)`

Procesa el path de un repositorio de datbases Oerm y genera el repo.db (sqlite). Basicamente cataloga cada database y genera una base sqlite (repo.db) en el directorio root del repositorio.

Parámetros

- `catalog_id` (*string*) – Id del catálogo al cual se le agregará este repositorio
- `path` (*string*) – Carpeta principal del repositorio
- `update` (*bool*) – (Opcional) Se actualiza o regenera completamente el catalogo

Ejemplo:

```
>>> from openerm.OermClient import OermClient
>>> c = OermClient("samples/openermcfg.yaml")
>>> c.add_repo("catalogo1", "/var/repo1")
```

`catalog_create(`*catalogdict*`)`

Crear un catálogo (lógico) de repositorios Oerm.

Parámetros `catalogdict` (*dict*) – Configuración del catálogo

Raise: ValueError si el catalogo <id> ya existe

Ejemplo:

```
>>> from openerm.OermClient import OermClient
>>> c = OermClient("samples/openermcfg.yaml")
>>> catalog_config = {"catalogo1": { "name": "Ejemplo catalogo␣
↪local", "type": "path", "enabled": True, "url": "c:/oerm/"}}
>>> c.catalog_create(catalog_config)
```

`catalogs`(*enabled=True*)

Lista los catalogos disponibles

Parámetros `enabled` (*bool*) – (Opcional) **True**: Solo los habilitados, **False**: Los deshabilitados, **None**: Todos

Devuelve Lista de catalogos

Tipo del valor devuelto List

Ejemplo

```
>>> from openerm.OermClient import OermClient
>>> c = OermClient("samples/openermcfg.yaml")
>>> c.catalogs(enabled=None)
>>> print("Catalogos disponibles: {0}".format(c.
↪catalogs(enabled=None)))
Catalogos disponibles: {'sql-test': {'enabled': False, 'name':
↪'Ejemplo catalogo SQL', 'type': 'sql'}, '
local-test': {'enabled': True, 'name': 'Ejemplo catalogo local', 'type
↪': 'path', 'urls':
['D:\pm\data\git.repo\openerm\samples\repo',
'D:\pm\data\git.repo\openerm\samples\otro']}}
```

`close_catalog`()

Cierra el catalgo activo

`current_catalog`()

Retorna el catalogo activo

Devuelve dict

`open_catalog`(*catalogid*)

Apertura de un catalogo

Parámetros `catalogid` (*string*) – Id del catalogo que se desea abrir

Tipos de catalogos:

- Lista de paths + repo.db
- Centralizado en Servidor SQL (TO DO)

`open_repo`(*reponum*)

Abre un repositorio

Parámetros `repo` (*int*) – Número del repositorio a abrir

Raise: ValueError: Si el repositorio no existe en el catalgo abierto

`query_reports`(*reporte=None*, *sistema=None*, *aplicacion=None*, *departamento=None*, *fecha=None*, *limit=None*, *returntype='list'*)

Consulta básica para buscar un reporte en los repositorios del catalogo. La busqueda se hace por cualquier de los atributos básicos de un reporte, y se puede hacer búsquedas parciales tipo LIKE en sql

Parámetros

- `report` (*string*) – Nombre del reporte
- `sistema` (*string*) – Nombre del sistema
- `aplicacion` (*string*) – Nombre de la aplicación
- `departamento` (*string*) – Nombre del departamento
- `fecha` (*string*) – Fecha de emsión del reporte
- `limit` (*int*) – cantidad máxima de resultados
- `returntype` (*string*) – Tipo de retorno

Devuelve list/string

Ejemplo:

```
>>> from openerm.OermClient import OermClient
>>> c = OermClient("samples/openermcfg.yaml")
>>> c.open_catalog("local-test")
>>> c.open_repo("Prueba1")
>>> resultados = c.query_reports(reporte="Carta", returntype=
↪"tablestr")
>>> print(resultados)
+-------------------------------------------------------+----------+-
↪---------------+-----------+--------------+-----------+---------
↪------------+
| Nombre                                                |    Fecha |␣
↪Departamento   | Sistema   | Aplicación   |   Páginas | Path    ␣
↪           |
|-------------------------------------------------------+----------+-
↪---------------+-----------+--------------+-----------+---------
↪------------|
| R8101614 - Cartas fianza activas por moneda y clas | 20160923 |␣
↪n/a            | n/a       | n/a          |         1 |␣
↪test1\database.oerm |
| R8101614 - Cartas fianza activas por moneda y clas | 20160923 |␣
↪n/a            | n/a       | n/a          |        20 |␣
↪test1\database.oerm |
| R8101611 - Cartas fianza requeridas por funcionari | 20160923 |␣
↪n/a            | n/a       | n/a          |         1 |␣
↪test1\database.oerm |
```

(continué en la próxima página)

(proviene de la página anterior)

```
+-------------------------------------------------+----------+-
↪---------------+-----------+--------------+-----------+---------
↪------------+
```

`reports()`

Retorna la lista completa de reportes del repositorio activo

Devuelve list

Ejemplo:

```
>>> from openerm.OermClient import OermClient
>>> c = OermClient("samples/openermcfg.yaml")
>>> c.open_catalog("local-test")
>>> c.open_repo("Prueba1")
>>> print(c.reports())
```

`repos()`

Lista los repositorios disponibles del catalgo activo

Devuelve Lista de repositorios

Tipo del valor devuelto List

Ejemplo

```
>>> from openerm.OermClient import OermClient
>>> c = OermClient("samples/openermcfg.yaml")
>>> c.open_catalog("local-test")
>>> print("Repositorios disponibles: {0}".format(c.repos()))
Repositorios disponibles: {'Prueba2': 'D:\pm\data\git.
↪repo\openerm\samples\otro\repo.db',
'Prueba1': 'D:\pm\data\git.repo\openerm\samples\repo\repo.db'}
```

`systems()`

Retorna la lista completa de sistemas del repositorio activo

Devuelve list

Ejemplo:

```
>>> from openerm.OermClient import OermClient
>>> c = OermClient("samples/openermcfg.yaml")
>>> c.open_catalog("local-test")
>>> c.open_repo("Prueba1")
>>> print(c.systems())
```

Database

Esta clase representa un objeto **Database** de reportes OERM. Basicamente es el objeto que permite salvar y recuperar reportes electrónicos desde sus archivos físicos.

Ver también:

- *openerm.MetadataContainer*
- *openerm.PageContainer*

class openerm.Database.Database(*file='prueba.oerm', mode='rb', default_compress_method=1, default_compress_level=1, default_encription_method=0, pages_in_container=10*)

Bases: `object`

Clase base para el manejo de un contenedor de reportes OERM

Esta clase representa un contenedor de reportes OERM se usa para la lectura y escritura de este tipo de datos.

Parámetros

- `file` (*string*) – Nombre del archivo físicos
- `mode` (*string*) – Modo "wb", "ab" o "rb" (Default: «rb»)
- `default_compress_method` (*int*) – Algoritmo default de compresion
- `default_compress_level` (*int*) – Nivel de compresión 0=mínimo, 1=normal, 2=máximo. Por defecto: 1.
- `default_encription_method` (*int*) – Algoritmo de encriptación
- `pages_in_container` (*int*) – Cantidad de páginas por contenedor

Ejemplo

```
>>> from openerm.Database import Database
>>> # Apertura en modo lectura
>>> dbin = Database(file = "out/.sin_compression_sin_encriptacion.oerm")
>>> # Apertura en modo escritura (NO append)
>>> dbout = Database(file = "out/.sin_compression_sin_encriptacion.oerm",␣
↪mode="wb")
```

add_page(*page*)

Agregar una página al reporte

> **Advertencia:** Documentación pendiente

`add_report`(*reporte='n/a'*, *sistema='n/a'*, *aplicacion='n/a'*, *departamento='n/a'*, *fecha='20190504'*)
Agregar un reporte al contenedor

> **Advertencia:** Documentación pendiente

`close`()
Cerrar el Database

> **Advertencia:** Documentación pendiente

`find_text`(*text*, *reports=None*)
Búsqueda de un texto dentro de uno o más reportes

Parámetros

- `text` (*string*) – Patrón de texto a buscar
- `reports` (*list*) – Lista de reportes dónde buscar o None en todos

Ejemplo

```
>>> from openerm.Database import Database
>>> db = Database(file = "out/.sin_compression_sin_encriptacion.oerm")
>>> db.find_text("IWY3")
[(2, 10, 991, 'AGH8B2NULTCTJOL-[IWY3]-4K6D8RRBYCRQCH')]
```

Devuelve Lista de reportes y páginas

`flush`()
«Comitear» los cambios

> **Advertencia:** Documentación pendiente

`get_report`(*reporte*)
Retorna el id de un Reporte

> **Advertencia:** Documentación pendiente

reports()
Retorna la colección de Reportes del **Database**

Devuelve

- *openerm.Reports*

Ejemplo

```
>>> from openerm.Database import Database
>>> db = Database(file = "out/.sin_compression_sin_encriptacion.oerm")
>>> for report in db.reports():
...     print(report)
Report: Reporte 1
Report: Reporte 2
>>>
```

set_report(*reporte*)
Establece el reporte actual

> **Advertencia:** Documentación pendiente

SpoolHostReprint

Un **SpoolHostReprint** es el objeto que permite la lectura de las salidas de impresión del tipo FCFC que son aquellos en los que la primer columna representa un «canal» de control para la impresora. Esta clase considera esta columna y particularmente el codigo «1» que representa el salto de pagina. Esta columna podra ser quitada o no segun se requiera.

Ver también:

- *openerm.SpoolFixedRecordLength*

class openerm.SpoolHostReprint.SpoolHostReprint(*inputfile*, *buffer_size=102400*, *encoding='Latin1'*)
Bases: object

Clase base para lectura de archivos tipo «host reprint».

Parámetros

- inputfile (*string*) – Path y nombre del archivo a leer
- buffer_size (*int*) – Opcional, tamaño del buffer de lectura. Por defecto 102400 bytes.
- encoding (*string*) – Opcional, Codificación de lectura. Por defecto *Latin1*

Devuelve None

Example

```
>>> from openerm.SpoolHostReprint import SpoolHostReprint
>>> with SpoolHostReprint(test_file, 102400) as s:
>>>             for page in s:
>>>                     print(page)
```

SpoolFixedRecordLength

Un **SpoolFixedRecordLength** es el objeto que permite la lectura de las salidas de impresión del tipo Registro de longitud fija. Este formato es típico en plataformas IBM, archivos en EBCDIC con una longitud de registro de 256 bytes son habituales de ver. Cada registro representa una línea, puede eventualmente ser del tipo FCFC, y contar con un canal de control.

Ver también:

- *openerm.SpoolHostReprint*

class openerm.SpoolFixedRecordLength.SpoolFixedRecordLength(*inputfile*, *buffer_size=102400*, *encoding='Latin1'*, *record_len=256*, *newpage_code='1'*)

Bases: `object`

Clase base para lectura de archivos de tamaño de registro fijo.

Parámetros

- `inputfile` (*string*) – Path y nombre del archivo a leer
- `buffer_size` (*int*) – Opcional, tamaño del buffer de lectura. Por defecto 102400 bytes.
- `encoding` (*string*) – Opcional, Codificación de lectura. Por defecto *Latin1*
- `record_len` (*int*) – Opcional, Longitud de registro (por defecto 256)
- `newpage_code` (*string*) – Opcional, Cadena o carácter que determina el salto de página

Devuelve None

Ejemplo

```
>>> from openerm.SpoolFixedRecordLength import␣
↪SpoolFixedRecordLength
>>> with SpoolFixedRecordLength(test_file, 102400) as s:
>>>             for page in s:
>>>                     print(page)
```

ReportMatcher

ReportMatcher es el objeto que identifica los reportes de una determinada cola de impresión. Lo hace mediante una serie de reglas que se definene en una archivo de configuración en formato YAML

`class openerm.ReportMatcher.ReportMatcher(`*configfile='openerm.cfg'*, *configbuffer=None*`)`

Bases: `object`

Matcher de reportes

El matcher de reportes se configura mediante un archivo yaml. Ver ejemplos en la carpeta *tools*, aunque tambien puede configurarase mediante un «buffer» o string yaml ya definido.

Parámetros

- `configfile` (*string*) – Path absoluto al archivo de configuración de la clase
- `configbuffer` (*string*) – Buffer con la configuración YAML de la clase.
- `parámatro tiene prioridad en caso de ingresar tambien configfile` (*Este*) –

Ejemplo

```
>>> from openerm.ReportMatcher import ReportMatcher
>>> r = ReportMatcher("../var/reports.yaml")
>>> print(r.match("R8101611"))
```

`match(`*page*`)`

Trata de determinar el reporte en función de los datos de una página

Parámetros `page` (*string*) – Texto de la página completa a identificar

Devuelve Datos del reporte

Tipo del valor devuelto tuple

Datos de retorno:

Tipo	Detalle
string	Id del reporte
string	Sistema
string	Departamento
string	Fecha

Ejemplo

```
>>> from openerm.ReportMatcher import ReportMatcher
>>> r = ReportMatcher("../var/reports.yaml")
>>> print(r.match("R8101611"))
```

Utils

Este módulo contiene todo tipo de funciones de uso general para el proyecto **OpenErm**

`class openerm.Utils.AutoNum`

Bases: `object`

Clase autonumeradora de valores

Ejemplo:

```
>>> from openerm.Utils import *
>>> my_id = AutoNum()
>>> my_id.get("Prueba")
1
>>> my_id.get("Otra cosa")
2
>>> my_id.get("Prueba")
1
```

`get(`*value*`)`

Retorna el numerador de un determinado valor

Parámetros `value` (*any*) – valor a numerar

Devuelve Número único del valor

Tipo del valor devuelto int

`list()`

Retorna la lista completa de valores, numeradores

Ejemplo:

```
>>> from openerm.Utils import *
>>> my_id = AutoNum()
>>> my_id.get("Prueba")
```

(continué en la próxima página)

(proviene de la página anterior)

```
1
>>> my_id.get("Otra cosa")
2
>>> my_id.get("Prueba")
1
>>> my_id.list()
[('Otra cosa', 2), ('Prueba', 1)]
```

Devuelve list

openerm.Utils.file_accessible(*filepath*, *mode*)
Verifica la accesibilidad de un archivo en un determinado modo de apertura.

Parámetros filepath (*string*) –

openerm.Utils.filesInPath(*path*, *pattern='*.*'*)
Retorna de forma recursiva los archivos que respetan un patrón

Parámetros

- path (*string*) – Path principal
- pattern (*string*) – (Opcional) patrón a buscar, por defecto "."

Ejemplo:

```
>>> for f in filesInPath("c:", "*.txt"):
>>>     print(f)
```

openerm.Utils.generate_filename(*mask*)
Genera un nombre de archivo en función a una máscara

Parámetros mask (*string*) – Mascara usada

openerm.Utils.get_values_from_byte(*byte*)
Retorna dos valores de un byte empaquetado

Parámetros byte – Entero que represta un byte

Return (v1, v2) Tupla con los dos valores enteros

openerm.Utils.set_byte_from_values(*value1*, *value2*)
Retorna un byte empaquetado a partir de dos valores

Parámetros

- value1 (*int*) – Entero 0 a 127
- value2 (*int*) – Entero 0 a 127

Devuelve byte

openerm.Utils.slugify(*text*, *delim='-'*)
Normaliza una cadena para ser usada como nombre de archivo.

Parámetros

- text (*str*) – String a normalizar
- delim (*str*) – Caracter de reemplazo de aquellos no válidos

Ejemplo:

```
>>> from openerm.Utils import *
>>> slugify("Esto, no es válido como nombre de Archivo!", "-")
'esto-no-es-valido-como-nombre-de-archivo'
```

openerm.Utils.str_to_list(*str_value*, *maxvalue*)
Devuelve una lista de enteros a partir de un string

Parámetros

- str_value (*string*) – Cadena de números separados por , o -
- maxvalue (*int*) – Máximo valor que puede tener la lista

Ejemplo:

```
>>> from openerm.Utils import *
>>> str_to_list("1,2,3,4", 10)
[1, 2, 3, 4]
>>> str_to_list("1-6,9, 12-14", 15)
[1, 2, 3, 4, 5, 6, 9, 12, 13, 14]
```

Config

Esta clase gestiona los archivos de configuración del proyecto Oerm. Se encarga de la carga de los archivos .cfg (formato yaml), realiza la validación de cada uno y devuelve un diccionario de datos para ser aprovechado luego.

class openerm.Config.Config(*configfile*, *schema=None*)
Bases: object

Clase base para el manejo de configuraciones «jerarquicas». :param configfile: Nombre del archivo físico de configuración a cargar :type configfile: string :param schema: Esquema de validación en formato yaml :type schema: string

Ejemplo

```
>>> from openerm.Database import Config
>>> cfg = Config("file.cfg", schema_yaml)
```

exception openerm.Config.ConfigLoadingException(*args)
Bases: Exception

Excepciones especiales para los cargadores de configuraciones

class openerm.Config.LoadConfig(configfile)
Bases: *openerm.Config.Config*

Clase base para el manejo de la configuración del proceso de carga.

class openerm.Config.ProcessorConfig(configfile)
Bases: *openerm.Config.Config*

Clase base para el manejo de la configuración del proceso de carga.

3.3.2 Internas - Contenedores

MetadataContainer

Un contenedor de metadatos es un tipo de «bloque» de un Database OpenErm, representa los atributos que describen el reporte. Cada reporte tiene un conjunto básico y fijo de atributos y es posible agregar los que deseemos.

Ver también:

- *openerm.PageContainer*
- *openerm.MetadataContainer*

class openerm.MetadataContainer.MetadataContainer(metadata=None)
Bases: object

Contenedeor de los «metadatos» del reporte

Parámetros metadata (*dict*) – Diccionario de los datos fundamentales de un reporte

add(extradata)
Agrega un diccionario de datos extra al contenedor

Parámetros extradata (*dict*) – Datos adicionales

Ejemplo:

```
>>> import datetime
>>> from openerm.MetadataContainer import MetadataContainer
>>> now = datetime.datetime.now()
>>> m = MetadataContainer("Reporte sin identificar", "n/a", "n/a",
↪ "n/a", now)
>>> extra_data = {"autor": "Ernesto Guevara", "Estado": "Draft"}
>>> m.add(extra_data)
```

dump()
Retorna en bytes el contenido de los metadatos ya listos para comprimir y

eventualmentre salvar físicamente en el Database Oerm. El «dump» termina siendo una representación plana de un diccionario de datos en formato JSON. Los datos pueden ser variables.

Ejemplo:

```
>>> import datetime
>>> from openerm.MetadataContainer import MetadataContainer
>>> now = datetime.datetime.now()
>>> m = MetadataContainer("Reporte sin identificar", "n/a", "n/a",
↪ "n/a", now)
>>> data = m.dump()
>>> print(data)
b'{"departamento": "n/a", "fecha": "20160914", "aplicacion": "n/a
↪", "sistema": "n/a", "reporte": "Reporte sin identificar"}'
```

Nota: El método *dump* entrega una estructura como el siguiente ejemplo:

```
+==============+
|    Datos     | --> Comprimibles
|  Json dump   |
+==============+
```

`load(`*data*`)`

Recupera de un conjunto de bytes los metadatos

Parámetros `data` – bytes

Ejemplo:

```
>>> import datetime
>>> from openerm.MetadataContainer import MetadataContainer
>>> now = datetime.datetime.now()
>>> m = MetadataContainer("Reporte sin identificar", "n/a", "n/a",
↪ "n/a", now)
>>> data = m.dump()
>>> print(data)
b'{"departamento": "n/a", "fecha": "20160914", "aplicacion": "n/a
↪", "sistema": "n/a", "reporte": "Reporte sin identificar"}'
>>> m2 = MetadataContainer()
>>> m2.load(data)
>>> print(m2)
{'departamento': 'n/a', 'fecha': '20160914', 'sistema': 'n/a',
↪'aplicacion': 'n/a', 'reporte': 'Reporte sin identificar'}
```

PageContainer

Un contenedor de páginas es un objeto que alberga de 1 a N páginas de un reporte. Es uno de los posibles tipos de Bloque que pueden salvarse en un Database **OpenErm**. La idea de agrupar varias páginas permite mejorar la performance de los compresores usados logrando mejores ratios de compresión y por ende reducir el tamaño de los reportes y el trafico de red o de disco.

Ver también:

- *openerm.MetadataContainer*
- *openerm.PageContainer*

class openerm.PageContainer.PageContainer(*max_page_count=1*)

Bases: object

Clase base para el manejo de un contenedor de páginas de reportes.Esta clase representa un contenedor de páginas, se agrupan las páginas en grupos para mejorar los niveles de compresión y de lectura/escritura.

Parámetros max_page_count (*int*) – Cantidad máxima de página en el contenedor

Ejemplo:

```
>>> from openerm.PageContainer import PageContainer
>>> p = PageContainer(10)
```

add(*page*)

Agrega una página al grupo. Esta rutina agrega un «string» que representa la página en el contenedor. La página queda registrada en una lista interna.

Parámetros page (*string*) – Texto completo de la página a incorporar

Error: ValueError: Cuando se superó la máxima cantidad de páginas del contenedor

Ejemplo:

```
>>> from openerm.PageContainer import PageContainer
>>> p = PageContainer(10)
>>> sample_page = "Pagina de una sola linea"
>>> p.add(sample_page)
>>> print(p)
[PageContainer]--->
Cantidad máxima de página: 10
Total de páginas                : 1
Pagina: 1:Pagina de una sola linea
[PageContainer]--->
```

clear()

Limpia la lista interna de páginas

Ejemplo:

```
>>> from openerm.PageContainer import PageContainer
>>> p = PageContainer(10)
>>> sample_page = "Pagina de una sola linea"
>>> p.add(sample_page)
>>> print(p)
[PageContainer]--->
Cantidad máxima de página: 10
Total de páginas         : 1
Pagina: 1:Pagina de una sola linea
[PageContainer]--->
>>> p.clear()
>>> print(p)
[PageContainer]--->
Cantidad máxima de página: 10
Total de páginas         : 0
[PageContainer]--->
```

dump()

Retorna en bytes el contenido de la lista de páginas del grupo. Este método es utilizado cuando construímos el «Bloque» que posteriormente salvaremos en un «Database»

Devuelve

- data (bytes) : Bytes completos de todas las páginas del grupo
- var_data (bytes) : Bytes estructurados con los datos para «desarmar» el grupo

Tipo del valor devuelto tuple

Ejemplo:

```
>>> from openerm import PageContainer
>>> p = PageContainer(10)
>>> sample_page = "Pagina de una sola linea"
>>> p.add(sample_page)
>>> (b'Pagina de una sola linea', b'\{\\\')
```

Nota: El método *dump* entrega una estructura como el siguiente ejemplo:

```
+==============+
|    Datos     | --> Comprimibles
+==============+
        +==============+
        | Pagina 1     |
        +==============+
```

(continué en la próxima página)

(proviene de la página anterior)

```
        +=============+
        | Pagina "N"  |
        +=============+
+=============+
|    Datos    | --> No comprimibles
| Adicionales |
+=============+
        +=============+================+    +================+
        | Cant.Paginas | Long. Pagina 1 | .. | Long. Pagina N |
        +=============+================+    +================+
```

---

get_page(*pagenum*)

Retorna una página determinada del grupo.

Parámetros pagenum (*int*) – Número de página a recupear

Devuelve Pagina o *None*

Tipo del valor devuelto (string)

Ejemplo

```
>>> import string
>>> import random
>>> from openerm.PageContainer import PageContainer
>>> def rnd_generator(size=1024, chars=string.ascii_uppercase +␣
↪string.digits):
                return ''.join(random.choice(chars) for _ in␣
↪range(size))
>>> pgroup = PageContainer(10)
>>> for i in range(1,11):
                random_text = rnd_generator(size=200*60)
                pgroup.add(random_text)
>>> print(pgroup.get_page(5))
```

load(*container_data*)

Recupera de un conjunto de bytes cada una de las páginas y construye la lista de las mismas

Parámetros container_data (*bytes*) – Bytes completos de todas las páginas del grupo. Ver *dump()*

Ejemplo:

```
>>> from openerm import PageContainer
>>> p = PageContainer(10)
>>> data = (b'Pagina de una sola linea', b'\{\\\')
>>> p.load(data)
```

3.3.3 Internas - Core

Block

Un «bloque» de datos. Un Block es la unidad miníma de un Database OERM. Hay dos tipos de bloque básico:

- *openerm.MetadataContainer* **para guardar los atributos de un** determinado reporte.
- *openerm.PageContainer* para guardar conjuntos de páginas

Ver también:

- *openerm.Database*
- *openerm.Compressor*
- *openerm.Cipher*

`class openerm.Block.Block(`*default_compress_method=1*, *default_compress_level=1*, *default_encription_method=0*`)`

Bases: `object`

Bloque de un archivo OpenERM. El bloque es la unidad mínima de información. Existen en está versión dos tipo de bloques básicos:

- Contenedores de metadatos
- Contenedores de páginas

Cada bloque puede o no ser comprimido y encriptado con algún algoritmo que dependerá de la implememntación OpenERM particular.

Parámetros

- `default_compress_method` (*int*) – Compresión por defecto del bloque (default = 1 - Gzip)
- `default_compress_level` (*int*) – Nivel de compresión por defecto del bloque (default = 1 Medium)
- `default_encription_method` (*int*) – Algoritmo de cifrado (0 = Ninguno)

`block_types = None`

Tipos de bloque 1 - Report metadata 2 - Pages container

`dump(`*tipo_bloque*, *data*, *variable_data=None*`)`

Convierte los datos en un bloque OpenErm que puede ser «salvable» en un archivo

Parámetros

- `tipo_bloque` (*int*) – Tipo de bloqe (1: metadatos, 2: páginas)
- `data` (*bytes*) – Bytes de los datos a salvar

- `variable_data` (*bytes*) – (opcional) Datos adicionales, es información que se salva en el bloque pero no se comprime

Ejemplo

```
>>> from openerm.Block import Block
>>> b = Block(default_compress_method=1, default_compress_level=1,␣
↪default_encription_method=0)
>>> data = b.dump(tipo_bloque=2, data=b"Esto hace las veces de una␣
↪pagina de reporte",variable_data=None)
>>> print(data)
b'\\\<}{\\\\1xs-.ÉWÈHLNUÈI,V(KMN-VHIU(ÍKT(HLÏR@^QjA~QI*\]Ê'
```

Nota: El método *dump* entrega una estructura como el siguiente ejemplo:

```
+========================+
| Long.Total del Bloque |    --> long (4 bytes)
+========================+
| Tipo de Bloque  |          --> int (1 bytes)
+=================+
| Alg. Compresión |          --> int (1 bytes)
+=================+
| Alg. Cifrado    |          --> int (1 bytes)
+========================+
| Long. de los Datos     |   --> long (4 bytes)
+========================+
|                        |
|        Datos           |   --> Longitud variable (datos comprimibles)
|                        |
+========================+
|                        |
|    Datos variables     |   --> (Opcional) Longitud variable (datos␣
↪NO comprimibles)
|                        |
+========================+
```

`load`(*data*)

Convierte un bloque OpenErm en datos lógicos

Parámetros `data` (*bytes*) – Bytes del bloque completo

Devuelve tuple

Se devuelve una «tupla» de 7 elementos:

Tipo	Detalle
long	Longitud total del bloque incluído este dato
int	Tipo de bloque
int	Tipo de compresión
int	Tipo de encriptado
long	longitud de los datos comprimidos
bytes	Datos
bytes	Datos adicionales

Ejemplo

```
>>> from openerm.Block import Block
>>> b = Block(default_compress_method=1, default_compress_level=1,␣
↪default_encription_method=0)
>>> data = b.dump(tipo_bloque=2, data=b"Esto hace las veces de una␣
↪pagina de reporte",variable_data=None)
>>> print(data)
b'\\\<}{\\\\1xs-.ÉWÈHLNUÈI,V(KMN-VHIU(ÍKT(HLÏR@^QjA~QI*\]Ê'
>>> print(b.load(data))
(60, 2, 1, 0, 49, b'Esto hace las veces de una pagina de reporte',␣
↪None)
```

Cipher

class openerm.Cipher.Cipher(*cipher_type=0*)

Bases: object

Clase base para el manejo de cifrado de «bytes».

Esta es una clase base para poder configurar distintos algoritmos de cifrado, pero que a la vez ofrezca una interfaz común para cifrar y descifrar.

Parámetros cipher_type (*int*) – Opcional, Tipo de cifrado por defecto (default=0-Ninguno)

Ejemplo

```
>>> from openerm.Cipher import Cipher
>>> c = Cipher(type=2)
>>> tmp = c.encode(b'esto es una prueba')
>>> print(tmp)
b'gAAAAABX2dYtQhe7Th3sA24606o_
↪747bts4n11Jm37gHW5SIRanP105loH4jPBJzEPrlBmhb9ai5FcXIBhTtUswHA_
↪H6yrzgVKOCPDigOiSKQjyfaWJryJI='
```

available_types

Retorna los tipos de cifrados disponibles.

Devuelve Lista de algoritmos cifrados.

Tipo del valor devuelto list

Ejemplo

```
>>> from openerm.Cipher import Cipher
>>> c = Cipher()
>>> c.available_types
[(0, 'Sin encriptación'), (1, 'Spritz'), (2, 'Fernet')]
```

decode(*data*)

Descifra un conjunto de bytes

Parámetros data (*bytes*) – conjunto de bytes cifrados

Devuelve datos descifrados.

Tipo del valor devuelto bytes

Ejemplo

```
>>> from openerm.Cipher import Cipher
>>> c = Cipher(type=2)
>>> tmp = c.encode(b'esto es una prueba')
>>> print(tmp)
b'gAAAAABX2dYtQhe7Th3sA24606o_
↪747bts4n11Jm37gHW5SIRanP105loH4jPBJzEPrlBmhb9ai5FcXIBhTtUswHA_
↪H6yrzgVKOCPDigOiSKQjyfaWJryJI='
>>> print(c.decode(tmp))
b'esto es una prueba'
```

encode(*data*)

Cifra conjunto de bytes

Parámetros data (*bytes*) – conjunto de bytes a cifrar

Devuelve datos cifrados.

Tipo del valor devuelto bytes

Ejemplo

```
>>> from openerm.Cipher import Cipher
>>> c = Cipher(type=2)
>>> tmp = c.encode(b'esto es una prueba')
>>> print(tmp)
b'gAAAAABX2dYtQhe7Th3sA24606o_
↪747bts4n11Jm37gHW5SIRanP105loH4jPBJzEPrlBmhb9ai5FcXIBhTtUswHA_
↪H6yrzgVKOCPDigOiSKQjyfaWJryJI='
```

`type`
: Tipo de cifrado.

`type_info(`*cipher_type*`)`
: Retorna la información de un determinado algoritmo de cifrado disponible.

  Returns: (tupla).

Compressor

Esta clase gestiona los algoritmos de compresión que se pueden utilizarse en un *openerm.Database* OERM. Este objeto ofrece dos rutinas básicas:

- `compress()`
- `decompress()`

y dos posibles formas de configurar estos metódos en la inicialización del objeto:

- **type**: Tipo de compresión
- **level**: Nivel de compresión 0=mínimo/rápido, 1=normal/estándar,2=máximo/lento

Tipos/Algoritmos de compresión

Tipo	Detalle
0	Sin compresión
1	**Gzip** es una abreviatura de GNU ZIP, un software libre GNU que reemplaza al programa compress de UNIX. gzip fue creado por Jean-loup Gailly y Mark Adler. Apareció el 31 de octubre de 1992 (versión 0.1). La versión 1.0 apareció en febrero de 1993. Gzip se basa en el algoritmo Deflate, que es una combinación del LZ77 y la codificación Huffman. Deflate se desarrolló como respuesta a las patentes que cubrieron LZW y otros algoritmos de compresión y limitaba el uso del compress.
2	**Bzip2** es un programa libre desarrollado bajo licencia BSD que comprime y descomprime ficheros usando los algoritmos de compresión de Burrows-Wheeler y de codificación de Huffman. El porcentaje de compresión alcanzado depende del contenido del fichero a comprimir, pero por lo general es bastante mejor al de los compresores basados en el algoritmo LZ77/LZ78 (gzip, compress, WinZip, pkzip,...). Como contrapartida, bzip2 emplea más memoria y más tiempo en su ejecución.
3	**LZMA** El Algoritmo de cadena de Lempel-Ziv-Markov o LZMA es un algoritmo de compresión de datos en desarrollo desde 1998. Se utiliza un esquema de compresión diccionario algo similar a LZ77, cuenta con una alta relación de compresión y una compresión de tamaño variable diccionario (de hasta 4 GB). Se utiliza en el formato 7z del archivador 7-Zip.
4	**LZ4** is lossless compression algorithm, providing compression speed at 400 MB/s per core (0.16 Bytes/cycle). It features an extremely fast decoder, with speed in multiple GB/s per core (0.71 Bytes/cycle). A high compression derivative, called LZ4_HC, is available, trading customizable CPU time for compression ratio. LZ4 library is provided as open source software using a BSD license.
5	**pyLZMA**, otra implementación de **LZMA**
6	**BLOSC** an extremely fast, multi-threaded, meta-compressor library
7	**Snappy** (previously known as Zippy) is a fast data compression and decompression library written in C++ by Google based on ideas from LZ77 and open-sourced in 2011. It does not aim for maximum compression, or compatibility with any other compression library; instead, it aims for very high speeds and reasonable compression
8	**Lzo** or Lempel–Ziv–Oberhumer is a lossless data compression algorithm that is focused on decompression speed
9	**Brotli** es una librería de compresión de datos de código abierto desarrollada por Jyrki Alakuijala y Zoltán Szabadka.1 2 Brotli está basado en una variante moderna del algoritmo LZ77, codificación Huffman y modelado de contexto de segundo orden.
10	**Zstandard**, or zstd as short version, is a fast lossless compression algorithm, targeting real-time compression scenarios at zlib-level and better compression ratios.

`class openerm.Compressor.Compressor(`*compress_type=1*, *level=1*`)`

Bases: `object`

Clase base para el manejo de compresión/descompresión de «bytes».

Esta es una clase base para poder configurar distintos algoritmos de compresión, pero que a la vez ofrezca una interfaz común para comprimir y descomprimir.

Parámetros

- `compress_type` (*int*) – Tipo de compresión
- `level` (*int*) – Nivel de compresión 0=mínimo, 1=normal, 2=máximo

Ejemplo

```
>>> from openerm.Compressor import Compressor
>>> c = Compressor(compress_type=1, level=1)
>>> tmp = c.compress(b"Esta es una prueba")
>>> print(tmp)
b'x^s-.ITH-V(ÍKT((*MMJ\<}'
```

`available_types`

Retorna los tipos de compresión disponibles.

Devuelve Lista de algoritmos disponibles.

Ejemplo

```
>>> from openerm.Compressor import Compressor
>>> c = Compressor()
>>> for c in c.available_types:
...     print(c)
...
(0, 'Sin compression')
(1, 'GZIP level=6 (1-9)')
(2, 'BZIP level=6 (1-9)')
(3, 'LZMA preset=6 (0-9) ')
(4, 'LZ4 nivel estándar')
(5, 'pyLZMA quality=1 (0-2)')
(6, 'BLOSC blosclz clevel=6 (1-9)')
(7, 'Snappy')
(8, 'Lzo level=2 (1-9)')
(9, 'Brotli quality=2 (1-11)')
(10, 'zstd level=3 (1-22)')
>>>
```

`compress(`*data*`)`

Comprime un conjunto de bytes

Parámetros `data` – (bytes) conjunto de bytes a comprimir

Devuelve bytes comprimidos.

Ejemplo

```
>>> from openerm.Compressor import Compressor
>>> c = Compressor(compress_type=1, level=1)
>>> tmp = c.compress(b"Esta es una prueba")
>>> print(tmp)
b'x^s-.ITH-V(ÍKT((*MMJ\<}'
```

`compression_type_info`(*compress_type*)

Retorna la información de un determinado algoritmo de compressión disponible.

Parámetros `compress_type` (*int*) – Id del algoritmo de compresión

Devuelve Información del algoritmo de compresión

Tipo del valor devuelto string

Ejemplo

```
>>> from openerm.Compressor import Compressor
>>> c = Compressor()
>>> print(c.compression_type_info(10))
zstd level=3 (1-22)
```

`decompress`(*data*)

Descomprime un conjunto de bytes

Parámetros `data` – (bytes) conjunto de bytes comprimidos

Devuelve bytes descomprimidos.

Ejemplo

```
>>> from openerm.Compressor import Compressor
>>> c = Compressor(compress_type=1, level=1)
>>> tmp = c.compress(b"Esta es una prueba")
>>> print(tmp)
b'x^s-.ITH-V(ÍKT((*MMJ\<}'
>>> print(c.decompress(tmp))
b'Esta es una prueba'
```

`level`

Nivel de compresión a utilizar

Ejemplo

```
>>> from openerm.Compressor import Compressor
>>> c = Compressor()
>>> print(c.level)
1
```

type
Tipo de compresión (algorimo) a utilizar.

Ejemplo

```
>>> from openerm.Compressor import Compressor
>>> c = Compressor()
>>> print(c.type)
1
```

Index

class openerm.Index.Index(*oermdb_file*)
Bases: object

add_container(*reporte_id*, *container_offset*)

add_report(*reporte*, *report_offset*, *pages_in_container*)

get_report(*reporte*)
Obtiene el id de un reporte en el índice

read()

write()

Pages

Clase «dummy», pensada eventualmente para el manejo de una colección de páginas.

`class openerm.Pages.Pages(`*file*, *index*`)`

Bases: `object`

Clase «Dummy» para el manejo de páginas

Report

Este objeto representa un Reporte almacenado en un *openerm.Database*.

Ejemplo:

Un reporte por su parte posee:

- Páginas
- Metadatos o atributos

Ver también:

- *openerm.Reports*
- *openerm.Database*

`class openerm.Report.Report(`*database*, *idrpt*`)`

Bases: `object`

Clase para el manejo de un Reporte OERM.

Parámetros

- `database` – Objeto *openerm.Database*
- `idrpt` (*int*) – Identificador único del reporte en el Database

Ejemplo

```
>>> fuerom openerm.Database import Database
>>> from openerm.Report import Report
>>> db = Database(file = "out/zstd-level-3-1-22.test.oerm", mode="rb")
>>> r = Report(db, 1)
>>> for page in r:
...     print(page[0:10])
...
Pagina 1 -
Pagina 2 -
Pagina 3 -
Pagina 4 -
Pagina 5 -
Pagina 6 -
```

(continué en la próxima página)

(proviene de la página anterior)

```
Pagina 7 -
Pagina 8 -
Pagina 9 -
Pagina 10
Pagina 11
```

data:

Tipo	Detalle
int	Id del reporte
string	Nombre del reporte
long	Offset al contenedor de metadatos
long	Max cantidad de páginas en los PageContainers
long	Offset al primer PageContainer
list	Lista de Offsets a los PageContainers

`find_text(`*text*`)`

Búsqueda de un texto dentro del reporte

Parámetros `text` (*string*) – Patrón de texto a buscar

Ejemplo

```
>>> from openerm.Database import Database
>>> from openerm.Report import Report
>>> db = Database(file = "out/.sin_compression_sin_encriptacion.oerm")
>>> r = Report(db, 1)
>>> report.find_text("IWY3")
[(2, 10, 991, 'AGH8B2NULTCTJOL-[IWY3]-4K6D8RRBYCRQCH')]
```

Devuelve

Lista de reportes y páginas

- Reporte id
- Página
- Posición en la página
- Extracto de la ocurrencia a modo de ejemplo

`get_page(`*pagenum*`)`

Retorna una pagina del reporte

Parámetros `pagenum` (*int*) – Número de página

Ejemplo

```
>>> from openerm.Database import Database
>>> from openerm.Report import Report
>>> db = Database(file = "out/zstd-level-3-1-22.test.oerm", mode="rb")
>>> r = Report(db, 1)
>>> p = r.get_page(5)
>>> print(p[0:30])
Pagina 5 -----------------
ZSV
>>>
```

Devuelve Texto completo de la página

Tipo del valor devuelto string

`id = None`
id del reporte

`metadata = None`
Metadatos del reporte

`nombre = None`
Nombre del reporte

`total_pages = None`
Cantidad total de páginas del reporte

Reports

Clase base para manejar una colección de reportes correspondientes a un Database OERM.

Ver también:

- *openerm.Report*

Nota:

- Esta clase no debiera usarse directamente
- En un futuro modificar la inicialización, para que reciba el archivo.oerm y los archivos de los

---

`class openerm.Reports.Reports(`*database*`)`
Bases: `object`

Clase base para manejar una colección de reportes correspondientes a un Database OERM.

Parámetros `database` (*openerm.Database*) – Objeto Database

Devuelve Iterador por la lista de Reportes

Tipo del valor devuelto iterable

Ejemplo

```
>>> from openerm.Database import Database
>>> from openerm.Reports import Reports
>>> db = Database(file = "out/zstd-level-3-1-22.test.oerm", mode="rb")
>>> for report in Reports(db):
...     print(report)
Report: Reporte 1
Report: Reporte 2
>>>
```

`find_text`(*text*, *search_in_reports=None*)

Búsqueda de un texto dentro de uno o más reportes

Parámetros

- `text` (*string*) – Patrón de texto a buscar
- `search_in_reports` (*list*) – Lista de id´s de reportes dónde buscar o None en todos

Ejemplo

```
>>> from openerm.Database import Database
>>> from openerm.Reports import Reports
>>> db = Database(file = "out/.sin_compression_sin_encriptacion.oerm")
>>> reports = Reports(db)
>>> reports.find_text("IWY3")
[(2, 10, 991, 'AGH8B2NULTCTJOL-[IWY3]-4K6D8RRBYCRQCH')]
```

Devuelve

Lista de reportes y páginas

- Reporte id
- Página
- Posición en la página
- Extracto de la ocurrencia a modo de ejemplo

`get_report`(*rptid*)

Retorna un objeto de la clase *openerm.Report* según el **id** del mismo

Parámetros `id` (*int*) – Número de reporte de la base Oerm

Ejemplo

```
>>> from openerm.Database import Database
>>> from openerm.Reports import Reports
>>> db= Database(file = "out/zstd-level-3-1-22.test.oerm", mode="rb")
>>> reports = Reports(db)
>>> reports.get_report(1)
>>> print(reports.get_report(1))
Report: Reporte 1
```

Devuelve Reporte solicitado

Tipo del valor devuelto *openerm.Report*

CAPÍTULO 4

Herramientas

4.1 splprocessor

splprocessor, es el monitor y procesador de archivos de colas de impresión (spool) del proyecto OpenErm. Su trabajo consta de:

- Monitorear una o más carpetas
- Detectar nuevos archivos en estas
- Detectar por patrones regulares nombres de archivo a procesar
- Verificar capacidad de bloqueo del archivo
- **Definir parámetros de procesamiento en función de:**
    - patrones regulares del nombre
    - patrones regulares dentro del contenido (limite de x bytes)
- Renombrado de los mismos a .locked
- Lectura de los archivos, proceso de paginas y generación de los reportes oerm
- Generación de log de proceso
- **Proceso final del Spool**
    - Borrado
    - Copiado a otra carpeta
    - Renombrado
    - Compresión en otra carpeta

splprocessor.init_argparse()
: init_argparse: Inicializar parametros del programa.

splprocessor.my_gettext(*s*)
: my_gettext: Traducir algunas cadenas de argparse.

4.2 spl2oerm

spl2oerm es una aplicación de línea de comandos, parte de la familia de herramientas del proyecto Openerm. Este comando se utiliza para procesar archivos de «spool» (salidas de las colas de impresión) y realiza las algunas de las siguientes operaciones:

- Lectura y decodificación del archivo
- Detección de páginas
- Detección de reportes
- Compresión y/o encriptación de páginas
- Almacenamiento final en Databases «oerm»

Básicamente esta herramienta es un «loader» o «cargador» de los reportes a la base de datos de un sistema Oerm para su posterior uso y consulta.

Una ejecución sin parámetros muestra esta ayuda:

```
uso: spl2oerm [-h] [--config-file CONFIGFILE] inputfile

Procesador de archivo de spool a Oerm (c) 2016, Patricio Moracho
<pmoracho@gmail.com>

argumentos posicionales:
inputfile                                Archivo a procesar

argumentos opcionales:
-h, --help                               mostrar esta ayuda y salir
--config-file CONFIGFILE, -f CONFIGFILE  Archivo de configuración␣
↪del proceso.
```

Esto claramente nos dice, que para ejecutar esta herramienta solo requerimos dos parámetros:

- El Archivo «spool» de entrada
- Un archivo de configuración del proceso que explicaremos a continuación

El proceso en sí requiere una serie de datos para funcionar y llevar a cabo exitosamente la carga de los reportes. Esta configuración se define en una archivo de texto escrito en formato yaml

Un ejemplo sencillo:

```
# vi: ft=yaml
# Openerm spool file load config file
#

load:
        #
        #  Definición del archivo de input
        #
        inputfile:
                encoding: cp500              # Codificación del␣
↪archivo de entrada (cp500=EBCDIC)
                record-length: 256           # Longitud del registro
                file-type: fixed             # Tipo de input fixed,␣
↪fcfc
                buffer-size: 102400          # Tamaño del buffer de␣
↪lectura

        #
        # Definiciones del proceso
        #
        process:
                EOP: NEVADO                  # Caracter o String que␣
↪define el salto de página
                report-cfg: ./reports.cfg    # Archivo de definición␣
↪de los reportes

        #
        # Definiciones de la salida
        #
        output:
                output-path: default
                compress-type: 10
                compress-level: 1
                cipher-type: 0
                pages-in-group: 10
```

- Toda línea o texto que comienza con el caracter # es considerada un comentario
- La primer sección *load* define completamente el proceso de carga
- **En *file* se configuran los parámetros para la interpretación básica del archivo**
    - *encoding* define la codificación del archivo, ver aqui la lista de psoibles codecs
    - *record-lenght*, para los tipos de archivo de registros de longitud fija, el tamaño de los mismos
    - *file-type* tipo de archivo, actualmente hay dos implementaciones *fixed* y *fcfc*

- *buffer-size*, tamaño del buffer de lectura, para los archivos de registros de tamño variable.

- **En *process* se configuran los parámetros inherentes al proceso lógico del spool**

  - *EOP* define el caracter o cadena que determina el cambio de página dentro del reporte.
  - *report-cfg* define el archivo de configuración de los reportes

- **En *output* se configuran los parámetros que definen el database físico a generar**

  - *output-path* define el archivo de configuración de los reportes
  - *compress-type* define el tipo de compresión a usar. Ver más documentación en *openerm.Compressor*
  - *compress-level* define el nivel de compresión a usar. Ver más documentación en *openerm.Compressor*
  - *cipher-type* define el cifrado de los archivos. Ver más documentación en *openerm.Cipher*
  - *pages-in-group* define la cantidad de páginas que manejara el contenedor correspondiente. Ver más documentación en `openerm.Pagecontainer`

`spl2oerm.init_argparse()`
: init_argparse: Inicializar parametros del programa.

`spl2oerm.my_gettext(`*s*`)`
: my_gettext: Traducir algunas cadenas de argparse.

4.3 catalogrepo

Esta herramienta realiza la indización o catálogo de los reportes contenidos en un repositorio **OpenErm**. Un repositorio no es más que una carpeta que contiene un conjunto de subcarpetas, y en estas un **Database** Oerm

`catalogrepo.init_argparse()`
: Inicializar parametros del programa.

`catalogrepo.procces_tree(`*path*, *update=False*`)`
: Procesa el path de un repositorio de datbases Oerm

  Parámetros

  - `path` (*string*) – Carpeta principal del repositorio
  - `update` (*bool*) – (Opcional) Se actualiza o regenera completamente el catalogo

4.4 checkoermdb

Verificación de databases Openermdb

`class checkoermdb.OermDataBase(`*inputfile*`)`
Bases: `object`

`checkoermdb.init_argparse()`
init_argparse: Inicializar parametros del programa.

`checkoermdb.my_gettext(`*s*`)`
my_gettext: Traducir algunas cadenas de argparse.

4.5 make module

Copyright (c) 2014 Patricio Moracho <pmoracho@gmail.com> # # make.py # # This program is free software; you can redistribute it and/or # modify it under the terms of version 3 of the GNU General Public License # as published by the Free Software Foundation. A copy of this license should # be included in the file GPL-3. # # This program is distributed in the hope that it will be useful, # but WITHOUT ANY WARRANTY; without even the implied warranty of # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the # GNU Library General Public License for more details. # # You should have received a copy of the GNU General Public License # along with this program; if not, write to the Free Software # Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

`class make.MyMake`
Bases: `object`

`build(`*tools*, *msg=True*, *debug=False*`)`

`check_packages(`*packages*, *msg=True*, *debug=False*`)`

`static check_tool(`*tool*, *command*, *find*, *debug=False*`)`

`check_virtualenv(`*msg=True*, *debug=False*`)`
Verifica la activación del entorno virtual del proyecto, ejecutando el activate.bat

Parámetros

- `msg` (*bool*) – Muestra los mensajes
- `debug` – (bool) : Muestra info adicional de la ejecución

Devuelve (bool) True si el entorno ha podido ser activado

`static clean(`*pattern*, *debug=False*`)`

`dev_install(`*msg=True*, *debug=False*`)`
Instalación del entorno de desarrollo en el proyecto

Parámetros

- msg (*bool*) – Muestra los mensajes
- debug – (bool) : Muestra info adicional de la ejecución

devcheck()

devinstall()

doc_install(*msg=True*, *debug=False*)
Instalación del entorno de documetación Sphinx

Parámetros

- msg (*bool*) – Muestra los mensajes
- debug – (bool) : Muestra info adicional de la ejecución

docinstall()

print_error()

run_devcheck(*msg=True*, *debug=False*)

run_tests(*msg=True*, *debug=False*)

test()

tools()

make.my_gettext(*s*)
my_gettext: Traducir algunas cadenas de argparse.

make.subprocess_cmd(*command*)

4.6 oerm_hostreprint_processor module

Copyright (c) 2014 Patricio Moracho <pmoracho@gmail.com> # # oerm_hostreprint_processor # # This program is free software; you can redistribute it and/or # modify it under the terms of version 3 of the GNU General Public License # as published by the Free Software Foundation. A copy of this license should # be included in the file GPL-3. # # This program is distributed in the hope that it will be useful, # but WITHOUT ANY WARRANTY; without even the implied warranty of # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the # GNU Library General Public License for more details. # # You should have received a copy of the GNU General Public License # along with this program; if not, write to the Free Software # Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

oerm_hostreprint_processor.Main()

oerm_hostreprint_processor.init_argparse()
init_argparse: Inicializar parametros del programa.

oerm_hostreprint_processor.my_gettext(*s*)
my_gettext: Traducir algunas cadenas de argparse.

oerm_hostreprint_processor.process_file(*configfile*, *inputfile*, *outputfile*, *compressiontype*, *complevel*, *ciphertype*, *testall*, *append*, *pagesingroups*)

4.7 Readoermdb

readoermdb.init_argparse()
: init_argparse: Inicializar parametros del programa.

readoermdb.my_gettext(*s*)
: my_gettext: Traducir algunas cadenas de argparse.

CAPÍTULO 5

Desarrollo

Algunos puntos importantes para el desarrollador

5.1 Requisitos iniciales

El proyecto **Openerm** esta construido usando el lenguaje **python** y varios «packages» o librerías adicionales para dicho lenguaje. Para poder construir las herramientas del proyecto es necesario preparar antes que nada, un entorno de desarrollo. A continuación expondremos en detalle cuales son los pasos para tener preparado el entorno de desarrollo. Este detalle esta orientado a la implementación sobre Windows 32 bits, los pasos para versiones de 64 bits son sustancialmente distintos, en particular por algunos de los «paquetes» que se construyen a partir de módulos en C o C++, de igual forma la instalación sobre Linux tiene sus grandes diferencias. Eventualmente profundizaremos sobre estos entornos, pero en principo volvemos a señalar que el siguiente detalle aplica a los ambientes Windows de 32 bits

- Obviamente en primer lugar necesitaremos Python, por ahora únicamente la versión 3.4. La correcta instalación se debe verificar desde la línea de comandos: `python --version`. Si todo se instaló correctamente se debe ver algo como esto `Python 3.4.0`, sino verificar que Python.exe se encuentre correctamente apuntado en el PATH.
- Es conveniente pero no mandatorio hacer upgrade de la herramienta pip: `python -m pip install --upgrade pip`
- Virutalenv:. Es la herramienta estándar para crear entornos «aislados» de python. En nuestro ejemplo **Openerm**, requiere de Ptython 3.4 y de varios «paquetes» adcionales de versiones especifícas. Para no tener conflictos de desarrollo lo que haremos mediante esta herramienta es crear un «entorno virtual» de python en

una carpeta del projecto (venv), dónde una vez «activado» dicho entorno podremos instalarle los paquetes que requiere el proyecto. Este «entorno virtual» contendrá una copia completa de Python y los paquetes mencionados, al activarlo se modifica el PATH al python.exe que apuntará ahora a nuestra carpeta del entorno y nuestras propias librerías, evitando cualquier tipo de conflicto con un entorno Python ya existente. La instalación de virtualenv se hara mediante `pip install virtualenv`

- Descargar el proyecto desde Bitbucket, se puede descargar desde la página el proyecto como un archivo Zip, o si contamos con Git sencillamente haremos un `git clone https://pmoracho@bitbucket.org/pmoracho/openerm.git`.
- El proyecto una vez descomprimido o luego del clonado del repositorio tendrá una estructura de directorios similar a la siguiente::

```
openerm
|-build
|-dist
|-doc
|-openerm
|-tests
|-tools
|-var
|-wheels
```

5.2 Preparación del entorno virtual local

Para poder ejecutar, o crear la distribución de la herramientas, lo primero que deberemos hacer es armar un entorno python «virtual» que alojaremos en una subcarpeta del directorio principal que llamarems «venv». En el proyecto incorporamos una herramienta de automatización de algunas tareas básicas. Se trata de `make.py`, la forma de ejecutarlo es la siguiente: `python tools\make.py` la ejecución si parámetros arrojará una salida como lo que sigue::

```
Automatización de tareas para el proyecto Openerm
(c) 2016, Patricio Moracho <pmoracho@gmail.com>

Uso: make <command> [<args>]

Los comandos más usados:
devcheck   Hace una verificación del entorno de desarrollo
devinstall Realiza la instalación del entorno de desarrollo virtual e instala␣
↪los requerimientos
clear      Elimina archivos innecesarios
test       Ejecuta todos los tests definidos del proyecto
tools      Construye la distribución binaria de las herramientas del proyecto

make.py: error: los siguientes argumentos son requeridos: command
```

Para preparar el entorno virtual simplemente haremos `python tools\make.py`

`devinstall`, este proceso si resulta exitoso deberá haber realizado las siguientes tareas:

- Creación de un entorno pyhton virtual en la carpeta «venv», invocable mediante `venv\Scripts\activate.bat` en Windows o `source venv/Scripts/activate` en entornos Linux o Cygwin/Mingw (en Windows)
- Instalado todas las dependencias necesarias

5.3 Notas:

- Hay dependecias que son fácilmente instalables mediante el comando `pip` y otras que no se instalan de la misma forma tan fácilmente. Estás últimas son librerías o proyectos en C o C++ que requieren de la compilación de distintos módulos, estos «paquetes», para poder instalarse mediante `pip` requieren que dispongamos de un compilador C/C++, algo que no siempre ocurre e incluso para ser más exactos, deberíamos contar con la misma versión del compilador que usa nuestra distribución python. Por esto hemos optado por incluir los paquetes ya compilados en su distribución binaria. Los requerimientos de este tipo podrán ser encontrados en la carpeta wheels.
- Es recomendable y cómodo, pero entiendo que no es mandatorio, contar con un entorno estilo «Linux», por ejemplo MinGW, tal como dice la página del proyecto: «MinGW provides a complete Open Source programming tool set which is suitable for the development of native MS-Windows applications, and which do not depend on any 3rd-party C-Runtime DLLs. (It does depend on a number of DLLs provided by Microsoft themselves, as components of the operating system; most notable among these is MSVCRT.DLL, the Microsoft C runtime library. Additionally, threaded applications must ship with a freely distributable thread support DLL, provided as part of MinGW itself).» De este entorno requerimos algunas herramientas de desarrollo: Bash para la línea de comandos y Make para la automatización de varias tareas del proyecto.

5.4 Otras consideraciones

- Usar «soft tabs»: Con cualquier editor que usemos configurar el uso del <tab> en vez de los espacios, yo prefiero el <tab> puro al espacio, entiendo que es válido el otro criterio pero ya los fuentes están con esta configuración, por lo que para evitar problemas al compilar los .py recomiendo seguir usando este criterio. Asimismo configurar en 4 posiciones estos <tab>.

CAPÍTULO 6

Indices y tablas

- genindex
- modindex
- search

Índice de Módulos Python

Índice

D

E

F

G

I

L

M

N

O

P

Q

R

S

T

W