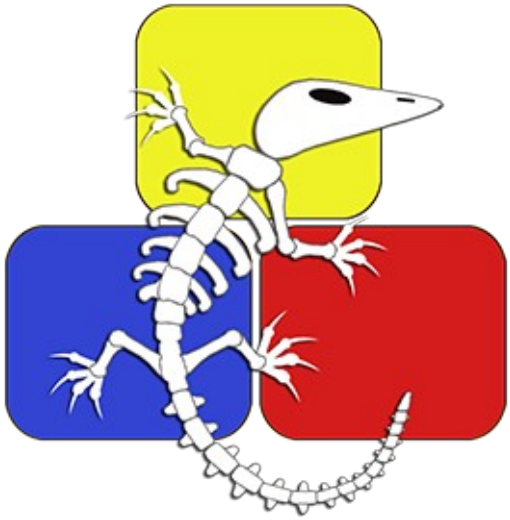


SOCKELETOM

Índice

- class_eseguridad_procesos.php
- class_edatos_procesos.php
- class_eimplementacion_procesos.php
- class_einterfaz_datos.php
- trait_egeco.php
- class_eherramientas.php
- class_eutiles.php
- class_gedee_epadre.php
- class_gedee_enucleo.php
- class_gedee_ecomun.php
- class_e_errores.php
- class_eerrores.php
- class_gedege_txt_emod
- class_herramienta_earreglo.php
- class_he_datfortxt.php

Licencia

Copyright

SOCKELETOM

¿QUÉ ES EL SOCKELETOM?

Es un framework que como todo framework brinda estructuras de directorios-archivos y estructuras de datos, junto con herramientas para la gestión y control de estas estructuras, en palabras más simples, mucho código que hará un porciento grande del trabajo necesario para que su aplicación o sistema se ejecute versátil y satisfactoriamente.

Este framework trata los scripts php como procesos y les permite una amplia intercomunicación entre ellos, no enmarcando así los scripts en modelos, todo lo contrario, permite el acoplamiento entre diferentes scripts o aplicaciones sin que estos se enmarquen en un patrón, modelo o filosofía determinada. Toda esta gestión en forma de procesos se realiza apoyada en herramientas de control y configuración a varios niveles, muy modular, muy versátil en el acople de scripts y/o aplicaciones, y escrito para el intercambio de datos entre scripts y/o aplicaciones de forma muy dinámica y ágil.

Como núcleo es base para constituir un framework mayor, un cms o un simple portal-sitio, todo depende de la funcionalidad de los procesos y la selección de estos que se haga para constituir su sockeletom, de esto también se desprende la posibilidad de hacer un Sockeletom personalizado.

¿Diferencias con los demás frameworks?

La diferencia con los demás frameworks es como están conformadas estas estructuras y que facilidades nos brindan estas herramientas de gestión. Además entre las diferencias está el poder ver las aplicaciones o scripts como procesos o sistemas de procesos, y entre estos procesos puede existir un intercambio de datos sin que los procesos tengan que conocer las estructuras básicas donde se encuentran estos datos para el intercambio.

Nos permite que estos procesos puedan ser ellos mismos frameworks con modelos, estructuras de datos y ficheros, y funcionamientos, diferentes, y que se puedan comunicar, pedir servicios entre ellos, utilizando las prestaciones que para ello brinda el Sockeletom. Además no es muy difícil o traumática la adaptación al Sockeletom de aplicaciones o frameworks ya programados y vigentes.

Para tener una mejor idea de lo explicado anteriormente aclaramos que podemos comunicar datos entre una aplicación con diseño MVC y otra sin este tipo de modelo y muy diferente, sin que haya que adaptar una a la otra, solo con pequeños cambios en cada una de ellas se hace la transferencia de datos entre una y otra a través de las herramientas que el Sockeletom pone a su disposición. Además lo anterior puede suceder no solo entre dos sino entre varias aplicaciones en solo una ejecución del motor php o petición de opción a una aplicación.

Es ideal para el intercambio de datos entre aplicaciones con mucho tiempo de creadas y otras de nueva creación.

Su dinámica de trabajo se basa en la filosofía socket-conector a la hora de diseñar aplicaciones, fue pensado para una alta modularidad y acople entre módulos, las aplicaciones que corran sobre Sockeletom pueden realizar su gestión específica y a la vez realizar gestiones como servicios para otras aplicaciones, sin dejar de ser independientes y/o autónomas en sus configuraciones, estructuras de ficheros, tipos de modelos o filosofías de diseño y funcionamientos, estilos de programación, contenido, forma, gestión, etc

También permite cierto control y seguridad del núcleo Sockeletom sobre las aplicaciones o procesos y algunas herramientas que sobre él se ejecutan, dando una posibilidad de administración a nivel de sistema Sockeletom sobre determinadas interioridades como pueden ser filtrado de datos por post y get, filtrado de ip, accionar o veto de ejecución de alguna aplicación o proceso, implementación de firewall de algún tipo, condicionamiento de variables del motor php, resumiendo: cualquier condicionamiento que permita el lenguaje php y que sea factible en el enfoque de procesos que este framework o motor de códigos da a sus códigos.

Los elementos globales del Sockeletom como sistema son, una estructura de archivos , una configuración del sistema y proceso núcleo , una estructura de datos dinámica, y un espacio reservado para los procesos y sus módulos cliente.

El Sockeletom está basado desde su nacimiento en la filosofía modular, está programado para ser lo más modular posible, dados los beneficios de esta filosofía.

En cuanto al tratamiento de código cliente, el Sockeletom orienta su tratamiento en tres direcciones o capas, la primera son los Gestores de Estructuras de Datos en Entidades Esquelemod (GEDEE), la segunda son los procesos que se programan utilizando como plataforma un GEDEE, es decir la creación-programación de un proceso, y la tercera son las aplicaciones que se programan utilizando como plataforma un proceso. Todas estas capas estarán siendo controladas por un proceso núcleo del sistema Sockeletom,

Los ficheros o archivos correspondientes al núcleo se encuentran en el path raíz_sistema_Sockeletom/esquelemod/e_modulos/nucleo, dentro de este path está el fichero e_nucleo_control.php encargado de hacer el arranque y control del proceso núcleo, los demás ficheros o archivos que componen el núcleo del sistema, y que en su mayoría se componen de clases y funciones, se encuentran en este mismo path dentro del directorio nucleo_bib_funciones, en el path raíz_sistema_Sockeletom/esquelemod/e_modulos/nucleo/nucleo_bib_funciones.

Requerimientos Técnicos

El sistema Sockeletom requiere de php5.4 o superior y 4 MB en hdd para su instalación.

Instalando SOCKELETOM

Se obtiene el paquete de instalación del Sockeletom y se copia en la raíz del servidor web instalado en su SO, el paquete no es más que un directorio con la estructura de ficheros del Sockeletom entre las que se encuentra un fichero index.php; Este fichero index.php es el que debe ser llamado desde su browser o navegador, eso es todo para ver la página por defecto que trae el sistema Sockeletom, ya está funcionando, el Sockeletom está hecho sobre la base de no necesitar para su funcionamiento nada más que no fuera el motor php, sus ficheros de configuración son .txt con estructuras para ser parseados por parsers YAML. Todo lo anterior es solo para el funcionamiento básico del sistema, los procesos, scripts, aplicaciones que se hagan para correr sobre el Sockeletom tienen toda la libertad de utilizar otras herramientas como gestores de BD etc.

Si todo va bien al llamar el index.php el browser o navegador le mostrará algo como esto:

```
BIENVENIDOS A SOCKELETOM *
```

Creando nuestro Hola mundo o primer proceso

Para crear nuestro hola mundo en el Sockeletom lo haremos como un proceso que estará formado solo por un script, iremos en la estructura de directorios del paquete Sockeletom hasta el directorio esquelemod/e_modulos/procesos y ahí crearemos un directorio de nombre hmundo quedando de esta forma el path esquelemod/e_modulos/procesos/hmundo; Dentro de este directorio crearemos un fichero con extensión php de nombre hola_mundo quedando de esta forma el path esquelemod/e_modulos/procesos/hmundo/hola_mundo.php y dentro de este fichero escribimos

```php
<?php
      echo " Hola Mundo ";
?>
```

ya terminamos con el proceso, ahora solo queda declararlo en la configuración del sistema para que este sepa que existe y lo ejecute.

Accedemos al fichero con path esquelemod/e_sistema/e_sistema_config.cnf que no es mas que un fichero txt con estructura yaml, y dentro de el buscamos la sección

```
##############################################
#  Valores referentes a Bloques de procesos  #
##############################################
```

y dentro de esta la subsección: `bloques_procesos:`
en esta subsección se verá una estructura como esta:

```
bloque_defecto:
```

```
                apertura:
                            gedee_proceso:
                                        namespace: \Emod\Nucleo\Gedees
                                        clase: GedeeEComun
                                        id_entidad: GedeeEComun
                            propiedades_implementacion_proceso:
                                        path_raiz: apertura
                                        path_arranque: apertura_control.php
                                        obligatoriedad: 1
```

que podemos reemplazar por esta otra:

```
bloque_defecto:
                miprocesohm:
                            gedee_proceso:
                                         namespace: \Emod\Nucleo\Gedees
                                         clase: GedeeEComun
                                         id_entidad: GedeeEComun
                            propiedades_implementacion_proceso:
                                        path_raiz: hmundo
                                        path_arranque: hola_mundo.php
                                        obligatoriedad: 1
```

`miprocesohm`: es el identificador que le voy a dar al proceso en el sistema.
`hmundo`: es el directorio que creamos para este proceso.
`hola_mundo.php`: es el fichero que creamos y ejecutaremos como control de este proceso.
`gedee_proceso:` es un bloque para las propiedades namespace, clase, e id de la entidad de la clase correspondientes al GEDEE al que se acoje el proceso correspondiente.
`propiedades_implementacion_proceso:` es un bloque para las propiedades `path_raiz`, `path_arranque`, y `obligatoriedad` correspondientes al proceso, es el lugar o path de los ficheros del proceso y la obligatoriedad es la prioridad para la ejecución del fichero que arranca el proceso siendo 1,2,3,4,5 los valores para include, include_once, require, require_once, eval respectivamente.

En este documento existe una sección que explica con detalle cada parte de la configuración del sistema, y que son los GEDEE.

Ahora guardamos los cambios al fichero de configuración del Sockeletom y ejecutamos la petición al index.php del Sockeletom en el browser o navegador, y si todo se hizo correctamente ya tenemos nuestro hola mundo.

Lo del reemplazo de una estructura por otra es opcional, puede dejar la estructura del proceso apertura y crear debajo la del hola mundo, respetando las normas del yaml para que quede el arreglo que genera el parser de yaml con una estructura entendible por el sistema Sockeletom; Al dejar ambas estructuras tendrán el resultado de ambos procesos, para comprender en su totalidad el funcionamiento de los procesos tenemos toda la descripción del funcionamiento de estos en este mismo documento, así como también las secciones referentes a ellos en el fichero de configuración.

Ahora haremos otra demostración tipo hola mundo pero explotando un poco más las potencialidades del Sockeletom, en lugar de hacer un proceso que emita el hola mundo haremos que para ello se comparta la gestión del string entre dos procesos, uno aportará el string 'hola' y el otro aportará el string 'mundo'; Para ello crearemos dos procesos, el proceso de nombre o identificador 'hola', y el proceso de nombre o identificador 'mundo'. Se hace de la siguiente manera:

Iremos en la estructura de directorios del paquete Sockeletom hasta el directorio esquelemod/e_modulos/procesos y ahí crearemos un directorio de nombre proceso_hola quedando de esta forma el path esquelemod/e_modulos/procesos/proceso_hola.

Ahora iremos en la estructura de directorios del paquete Sockeletom hasta el directorio esquelemod/e_modulos/procesos/proceso_hola y ahí crearemos un directorio de nombre configuracion quedando de esta forma el path esquelemod/e_modulos/procesos/proceso_hola/configuracion ; Dentro de este directorio crearemos un fichero con extensión php de nombre configuracion_proceso_hola.php quedando de esta forma el path esquelemod/e_modulos/procesos/proceso_hola/configuracion/configuracion_proceso_hola.php y dentro de este fichero escribimos:

```
# Fichero de configuracion del proceso hola
# version YAML

# Propiedades del proceso hola
 propiedades_proceso:
                      clave_proceso: hola
#
# Datos de seguridad del proceso hola
 datos_seguridad:
                acceso_datos:
                             ambito: restrictivo
                             procesos:
                                     \Emod\Nucleo\Gedees:
                                                          GedeeEComun:
                                                                      mundo: leer
```

El Sockeletom pone a disposición de los procesos varios contenedores llamados entidades esquelemod, cada uno de estos contenedores tienen un objetivo específico y su utilidad la mostramos en este hola mundo. El fichero de configuración que anteriormente elaboramos, consta de dos secciones, primera las propiedades del proceso, que no es mas que un grupo de datos descriptivos y otros con otras funcionalidades que se verán mas adelante, y segunda los datos de seguridad del proceso hola, estos datos de seguridad permitirán declarar qué procesos pueden acceder a datos escritos por el proceso hola en la entidad(contenedor) EEoDatos, la estructura

```
datos_seguridad:
                acceso_datos:
                             ambito: restrictivo
                             procesos:
```

```
                                        \Emod\Nucleo\Gedees:
                                                    GedeeEComun:
                                                                mundo: leer
```

se desglosa de la siguiente forma:
son los datos de seguridad(`datos_seguridad:`), para el acceso a datos en la entidad(contenedor) EEoDatos(`acceso_datos:`), el ámbito restrictivo indica que se restringen todos los procesos excepto los que se declaran en la estructura procesos(`ambito:`), procesos se compone de los procesos que tendrán acceso a los datos(`procesos:`), estos procesos deben ser declarados con el namespace de su GEDEE(`\Emod\Nucleo\Gedees:`), la clase de su GEDEE(`GedeeEComun`) y finalmente el identificador o nombre del proceso(`mundo:`) con el permiso de las acciones que puede ejercer sobre los datos a acceder(`leer`).

El término GEDEE tiene toda una sección para su explicación en este documento, solo adelantarles que es una interfaz para el manejo de los datos en las entidades esquelemod.

Continuamos creando dentro del directorio esquelemod/e_modulos/procesos/proceso_hola un fichero con extensión php de nombre control_proceso_hola.php quedando de esta forma el path esquelemod/e_modulos/procesos/proceso_hola/control_proceso_hola.php y dentro de este fichero escribimos:

```
<?php

 //proceso hola

 $dato = array( "hola");

 $La_fich_interfaz_config_proc_hola['Ls_pathfich_interfaz'] = 'hereda' ;
 $La_fich_datos_config_proc_hola['Ls_pathfich_datos'] = 'configuracion_proceso_hola.php' ;
 $La_fich_datos_config_proc_hola['Ls_path_base'] = $this->EEoNucleo->pathDirEsquelemod().'/e_modulos/'.$this->EEoNucleo-
 >pathDirRaizProcesos().'/proceso_hola/configuracion' ;

 $datos_configuracion_phola = $this->EEoInterfazDatos->gestionEjecucionInterfazSalida( $this->EEoNucleo-
 >idProcesoEjecucion() , $La_fich_interfaz_config_proc_hola , $La_fich_datos_config_proc_hola , 'statu_procesos' ) ;
 //segundo inicializo los datos de configuracion de este proceso
 if ( !empty( $datos_configuracion_phola ) )
 {
  $this->EEoConfiguracion->iniciarDatosConfiguracionProceso( $datos_configuracion_phola ) ;
  $this->EEoSeguridad->iniciarDatosSeguridadProceso( $datos_configuracion_phola['datos_seguridad'] ) ;
 }

 $this->EEoDatos->iniciarDatosSalidaProceso( $dato );

?>
```

El código que acabamos de elaborar consta de:

- declaración de la variable `$dato` y su valor, este será el dato que el proceso hola pondrá en la entidad EEoDatos.

- declaración de las variables $La_fich_interfaz_config_proc_hola['Ls_pathfich_interfaz'] , $La_fich_datos_config_proc_hola['Ls_pathfich_datos'] y $La_fich_datos_config_proc_hola['Ls_path_base'] y su valores, necesarios como parámetros para auxiliarnos de la entidad EEoInterfazDatos que se dedica a la gestión de datos incluyendo parsers, orígenes de datos y caches de datos entre otras cosas, esta entidad tiene una sección para su explicación en este documento.
- Gestión de datos de configuración del proceso hola mediante la entidad EEoInterfazDatos $datos_configuracion_phola = $this->EEoInterfazDatos->gestionEjecucionInterfazSalida($this->EEoNucleo->idProcesoEjecucion() , $La_fich_interfaz_config_proc_hola , $La_fich_datos_config_proc_hola , 'statu_procesos') ;
- Escritura de datos de configuración del proceso hola en la entidad EEoConfiguracion $this->EEoConfiguracion-iniciarDatosConfiguracionProceso($datos_configuracion_phola) , esta acción no es obligatoria para el objetivo final que perseguimos en el hola mundo.
- Escritura de datos de seguridad del proceso hola en la entidad EEoSeguridad $this->EEoSeguridad->iniciarDatosSeguridadProceso($datos_configuracion_phola['datos_seguridad']) , esta acción si es obligatoria para el objetivo final que perseguimos en el hola mundo porque deja declarado el permiso a otros procesos al acceso de datos escritos por el proceso hola en la entidad EEoDatos.
- Escritura de datos del proceso hola en la entidad EEoDatos para ser leídos posteriormente por otros procesos. $this->EEoDatos->iniciarDatosSalidaProceso($dato);

Continuamos ahora elaborando el proceso que llamaremos 'mundo', identificador o nombre descriptivo de la función que va a realizar, para ello iremos en la estructura de directorios del paquete Sockeletom hasta el directorio esquelemod/e_modulos/procesos y ahí crearemos un directorio de nombre proceso_mundo quedando de esta forma el path esquelemod/e_modulos/procesos/proceso_mundo.

Crearemos dentro del directorio esquelemod/e_modulos/procesos/proceso_mundo un fichero con extensión php de nombre control_proceso_mundo.php quedando de esta forma el path esquelemod/e_modulos/procesos/proceso_mundo/control_proceso_mundo.php y dentro de este fichero escribimos:

```
<?php

 //proceso mundo

 $dato_proceso_hola = $this->EEoDatos->accederDatosSalidaProceso( 'hola','hereda','hereda','hereda' ) ;

 echo " $dato_proceso_hola[0] mundo ";

?>
```

El código que acabamos de elaborar consta de:

- Lectura de datos por parte del proceso de identificador 'mundo' en la entidad EEoDatos, el proceso 'mundo' lee el dato que escribió el proceso 'hola' en la estructura de datos de la entidad EEoDatos

$dato_proceso_hola = $this->EEoDatos->accederDatosSalidaProceso('hola','hereda','hereda','hereda') ;

- Emisión del string 'hola mundo ' echo " $dato_proceso_hola[0] mundo ";

Ya tenemos los dos procesos hechos ahora debemos ponerlos en cola de procesos para su ejecución, esto lo haremos declarándolo en la configuración del sistema para que este sepa que existe y lo ejecute.

Accedemos al fichero con path esquelemod/e_sistema/e_sistema_config.cnf que no es mas que un fichero txt con estructura yaml, y dentro de el buscamos la sección

```
##############################################
#  Valores referentes a Bloques de procesos  #
##############################################
```

y dentro de esta la subsección: `bloques_procesos:`
en esta subsección se verá una estructura como esta:

```
bloque_defecto:
                apertura:
                          gedee_proceso:
                                       namespace: \Emod\Nucleo\Gedees
                                       clase: GedeeEComun
                                       id_entidad: GedeeEComun
                          propiedades_implementacion_proceso:
                                       path_raiz: apertura
                                       path_arranque: apertura_control.php
                                       obligatoriedad: 1
```

que podemos reemplazar por esta otra:

```
bloque_defecto:
                hola:
                      gedee_proceso:
                                    namespace: \Emod\Nucleo\Gedees
                                    clase: GedeeEComun
                                    id_entidad: GedeeEComun
                      propiedades_implementacion_proceso:
                                       path_raiz: proceso_hola
                                       path_arranque: control_proceso_hola.php
                                       obligatoriedad: 1
                mundo:
                      gedee_proceso:
                                    namespace: \Emod\Nucleo\Gedees
                                    clase: GedeeEComun
                                    id_entidad: GedeeEComun
                      propiedades_implementacion_proceso:
                                      path_raiz: proceso_mundo
                                      path_arranque: control_proceso_mundo.php
                                      obligatoriedad: 1
```

donde:
`hola` y `mundo`: son los identificadores que le voy a dar a los procesos en el sistema.

`gedee_proceso:` es un bloque para las propiedades namespace, clase, e id de la entidad de la clase correspondientes al GEDEE al que se acoje el proceso correspondiente.
`propiedades_implementacion_proceso:` es un bloque para las propiedades `path_raiz`, `path_arranque`, y `obligatoriedad` correspondientes al proceso, es el lugar o path de los ficheros del proceso y la obligatoriedad es la prioridad para la ejecución del fichero que arranca el proceso siendo 1,2,3,4,5 los valores para include, include_once, require, require_once, eval respectivamente.

En este documento existe una sección que explica con detalle cada parte de la configuración del sistema, y que son los GEDEE.

Ahora guardamos los cambios al fichero de configuración del Sockeletom y ejecutamos la petición al index.php del Sockeletom en el browser o navegador, y si todo se hizo correctamente ya tenemos nuestro hola mundo.

A continuación explicamos qué son los procesos, el núcleo, los GEDEEs, las prestaciones, etc. en términos de códigos y estructuras, que nos brinda el sistema.

PROCESOS

Llámese proceso a cualquier script o conjunto de ellos que tengan comienzo y puede o no tener final, el proceso puede concluir y dar paso a la ejecución de otros procesos sin que concluya el interprete php. También puede comprender desde un script hasta toda una aplicación o sistema. Siempre en el ámbito del lenguaje php.

El sistema Sockeletom está creado para la ejecución de procesos o incluso listas de procesos, declaradas en el fichero de configuración del sistema Sockeletom o invocados desde el interior de un proceso en ejecución, es importante comprender estas dos vías de ejecución, la primera es implementada en el fichero de configuración del sistema mientras que la segunda es a petición de un proceso que se esté ejecutando y necesite los servicios de otro proceso; Estos procesos serán registrados en un historial de procesos junto con algunas características de estos, y estarán sometidos al control de un proceso principal o núcleo del sistema Sockeletom, y a una serie de restricciones o permisiones desde la administración del sistema Sockeletom y plasmado en el fichero de configuración del sistema Sockeletom.

Cuando un proceso es ejecutado puede o no terminar la ejecución de php, en el caso de no terminar la ejecución del interprete php, el control del interprete regresa al proceso que lanzó el proceso terminado,es decir el proceso hijo devuelve el control al proceso padre, siendo en ultima instancia el proceso núcleo, el padre o raíz de el árbol de procesos que se gestione, los procesos hijos tienen acceso a todas las prestaciones del motor Sockeleton al igual que sus procesos padres, al igual que están sometidos a los mismos controles desde la seguridad declarada en el motor Sockeleton.

Los procesos se localizan o tienen como raíz el directorio raíz_sistema_Sockeletom/esquelemod/e_modulos/(path_raiz_procesos), (path_raiz_procesos) es el valor definido en la variable path_raiz_procesos en el fichero de configuración del sistema Sockeletom, en este path existirá una estructura de directorios que responderá a cada proceso específico, y que:

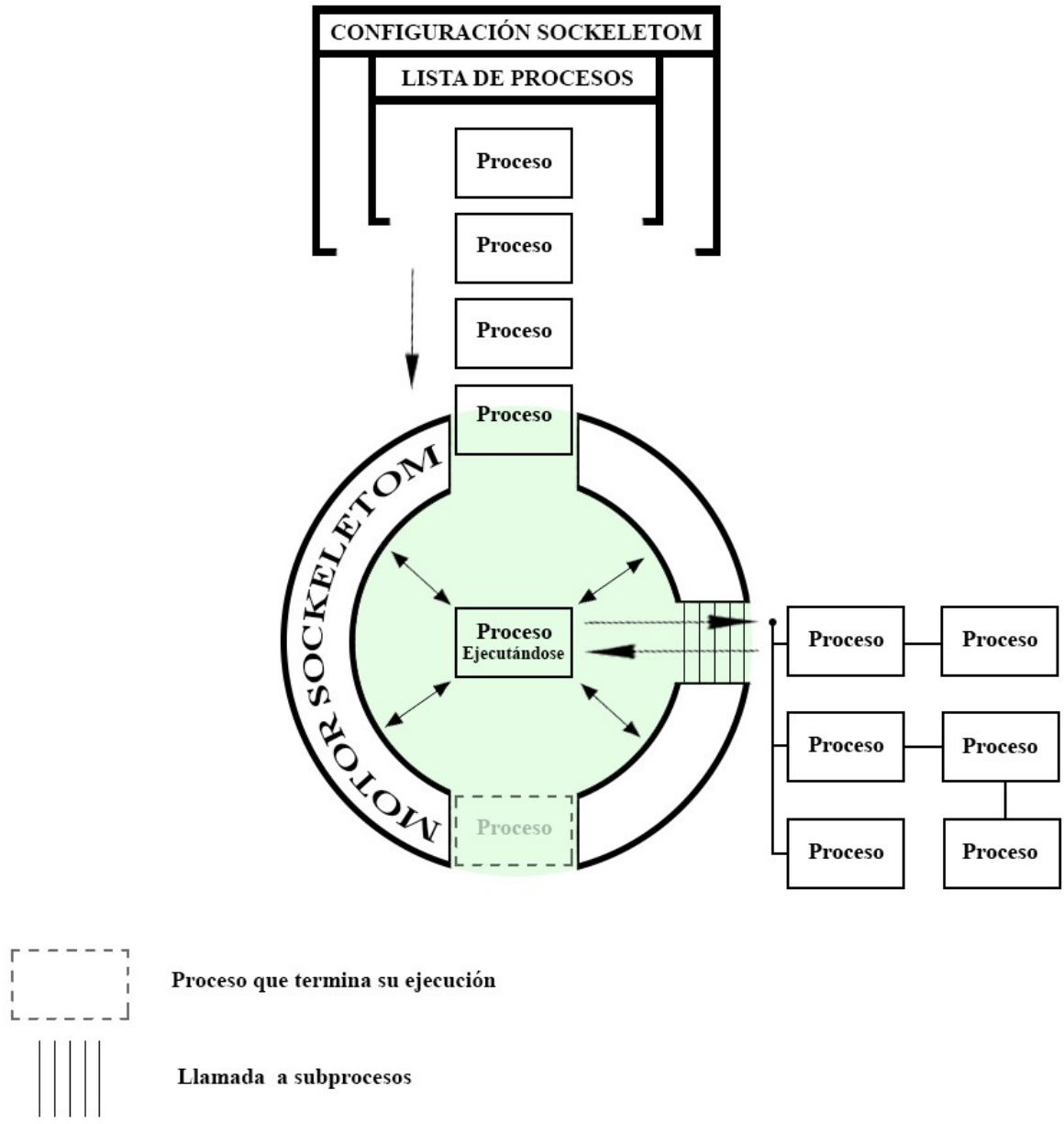
En caso de ser ejecutado el proceso desde la lista de procesos que pertenece a la configuración del sistema, entonces coincidirá con el valor del elemento:

propiedades_implementacion_proceso: path_arranque

elemento que pertenece a la configuración del sistema en la sección de ¨ Valores referentes a Procesos a ejecutar en cada bloque ¨.

En caso de ser ejecutado el proceso desde otro proceso en ejecución, esta estructura de directorios será brindado en el parámetro $La_entrada_datos_proceso['path_arranque'] del procedimiento cargaControlProcesos al que tiene acceso todo proceso. La ayuda de este procedimiento se encuentra en class_eimplementacion_procesos.php.

A continuación un gráfico para ayudar a comprender la dinámica básica de los procesos



A menudo verán en sus estructuras, procedimientos, nombres de clases, objetos etc. la palabra Esquelemod, Emod o la sigla E, esto es debido a que Sockeletom en sus inicios se llamó esquelemod contracción de Esqueleto Modular y se conservaron el nombre y sus siglas en la dinámica del núcleo del sockeletom.

Los procesos pueden tener sus estructuras propias de datos y los espacios para contener y gestionar estos, además están los espacios y prestaciones en entidades clases y objetos, para la gestión de

datos y sus estructuras que brinda el sistema Sockeletom, todos ellos al alcance de cualquier proceso desde su ámbito de ejecución, estos espacios y prestaciones son:

- Prestaciones que brinda el núcleo Sockeletom para el trabajo con los procesos. Estas prestaciones vienen dadas por un grupo de procedimintos públicos accesibles en el objeto núcleo, este objeto esta accesible desde cualquier ámbito (global o local) en la variable $EEoNucleo en caso de necesitar hacer una petición mas directa de una referencia a este objeto se puede hacer implementando el siguiente procedimiento y sus parámetros:
  \Emod\Nucleo\CropNucleo::referenciarObjeto('EEoNucleo' , 'NucleoControl' , 'Emod\Nucleo')
  Una explicación más detallada de la estructura de este objeto se encuentra en $EEoNucleo en este mismo documento.

- Prestaciones que brinda la configuración del sistema Sockeletom, lugar donde existe una sección dedicada a los procedimientos, y otras secciones como la declaración de GEDEEs, útiles, herramientas, gestión de errores, entre otros. Una explicación más detallada de la configuración del sistema Sockeletom se encuentra en la descripción de e_sistema_config.cnf en este mismo documento.

- Prestaciones que vienen dadas por un grupo de procedimientos públicos accesibles en un objeto para la gestión y caché de datos de forma general, en diferentes fuentes de datos, este objeto esta accesible desde cualquier ámbito (global o local) en la variable $EEoInterfazDatos en caso de necesitar hacer una petición mas directa de una referencia a este objeto se puede hacer implementando el siguiente procedimiento y sus parámetros:
  \Emod\Nucleo\CropNucleo::referenciarObjeto('EEoInterfazDatos' , 'InterfazDatos' , 'Emod\Nucleo\Herramientas')
  Una explicación más detallada de la estructura de este objeto se encuentra en $EEoInterfazDatos en este mismo documento.

- Prestaciones que vienen dadas por un grupo de procedimientos públicos accesibles en un objeto para la gestión y contención de datos de configuración, este objeto esta accesible desde cualquier ámbito (global o local) en la variable $EEoConfiguracion en caso de necesitar hacer una petición mas directa de una referencia a este objeto se puede hacer implementando el siguiente procedimiento y sus parámetros:
  \Emod\Nucleo\CropNucleo::referenciarObjeto('EEoConfiguracion' , 'ConfiguracionProcesos' , 'Emod\Nucleo')
  Una explicación más detallada de la estructura de este objeto se encuentra en $EEoConfiguracion en este mismo documento.

- Prestaciones que vienen dadas por un grupo de procedimientos públicos accesibles en un objeto para la gestión y contención de datos de seguridad, este objeto esta accesible desde cualquier ámbito (global o local) en la variable $EEoSeguridad en caso de necesitar hacer una petición mas directa de una referencia a este objeto se puede hacer implementando el siguiente procedimiento y sus parámetros:
  \Emod\Nucleo\CropNucleo::referenciarObjeto('EEoSeguridad' , 'SeguridadProcesos' , 'Emod\Nucleo')

Una explicación más detallada de la estructura de este objeto se encuentra en $EEoSeguridad en este mismo documento.

- Prestaciones que vienen dadas por un grupo de procedimientos públicos accesibles en un objeto para la gestión y contención de datos para el trabajo interno del proceso, o para el intercambio con otros procesos, este objeto esta accesible desde cualquier ámbito (global o local) en la variable $EEoDatos en caso de necesitar hacer una petición mas directa de una referencia a este objeto se puede hacer implementando el siguiente procedimiento y sus parámetros:

\Emod\Nucleo\CropNucleo::referenciarObjeto('EEoDatos' , 'DatosProcesos' , 'Emod\Nucleo')

Una explicación más detallada de la estructura de este objeto se encuentra en $EEoDatos en este mismo documento.

- Prestaciones que vienen dadas por un grupo de procedimientos públicos accesibles en un objeto para la gestión y ejecución de procesos, este objeto esta accesible desde cualquier ámbito (global) en la variable $EEoImplementacionProcesos, y (local) todos los procesos se ejecutan dentro del objeto $EEoImplementacionProcesos por lo que tienen acceso a sus procedimientos y propiedades, en caso de necesitar hacer una petición mas directa de una referencia a este objeto se puede hacer implementando el siguiente procedimiento y sus parámetros:

\Emod\Nucleo\CropNucleo::referenciarObjeto('EEoImplementacionProcesos ' , 'ImplementacionProcesos ' , 'Emod\\Nucleo')

Una explicación más detallada de la estructura de este objeto se encuentra en $EEoImplementacionProcesos en este mismo documento.

- Prestaciones que vienen dadas por un grupo de procedimientos públicos accesibles en una clase de procedimientos estáticos para la gestión y contención de entidades GEDEEs, esta clase esta accesible desde cualquier ámbito (global) (local), para utilizarla se implementaría de la siguiente forma:

Emod\Nucleo\GEDEEs

Una explicación más detallada de la estructura de este objeto se encuentra en GEDEEs en este mismo documento.

- Prestaciones que vienen dadas por un grupo de procedimientos públicos accesibles en clases y objetos para la gestión, tratamiento y contención de entidades gestoras de errores, estas clases y objetos estan accesibles desde cualquier ámbito (global) (local). Una explicación más detallada sobre la gestión y tratamiento de errores se encuentra en Gestión y tratamiento de Errores en este mismo documento.

- Prestaciones que vienen dadas por un grupo de procedimientos públicos accesibles en una clase de procedimientos estáticos para la gestión y contención de entidades Herramientas, Utiles y otros, estas clases estan accesibles desde cualquier ámbito (global) (local), Una explicación más detallada de la estructura de este objeto se encuentra en Herramientas, Utiles y otros en este mismo documento.

NÚCLEO

El núcleo del Sockeletom está compuesto de varios elementos entre los que se encuentran:

- Proceso núcleo
- Entidades Esquelemod
- GEDEEs
- Gestión y Tratamiento de Errores
- Configuración del Sistema Sockeletom

Proceso núcleo

El núcleo del Sockeletom es un proceso especial, que es el encargado de controlar la ejecución de cualquier pedido que se haga al sistema Sockeletom, entiéndase todas las opciones que se alojen en el sistema y sus diferentes capas, estructuras y códigos cliente.

Este proceso tiene como soporte:

Controlador

- un script (fichero) con el nombre de e_nucleo_control.php que hace de controlador.

El controlador es quien hace el arranque del proceso núcleo, carga los ficheros de clases patrones y padres, carga las clases de los objetos que conforman el núcleo del sistema, crea los objetos que conforman el núcleo del sistema, inicia el objeto núcleo, carga la configuración del sistema, y le ordena al objeto núcleo la ejecución de la cola de procesos que se encuentra en la configuración del sistema.

Una descripción detallada del código se encuentra en e_nucleo_control.php.

Este proceso cuenta con un GEDEE programado específicamente para las gestiones de un proceso núcleo, el GEDDE que comentamos tiene como identificador ENucleo.

Luego de hacer su tarea, el controlador pasa su control a la entidad EEoNucleo, quien ejerce a partir de ese momento el control del sistema Sockeletom, se puede ejecutar otro proceso de tipo núcleo que puede hacer su gestión utilizando las Entidades Esquelemod, es decir este nuevo

proceso de tipo núcleo a ejecutar no tiene que reemplazar o modificar las Entidades Esquelemod, de hacerlo debe tenerse en cuenta que se modifica toda la lógica del sistema actual.

Por el momento y sujeto a estudio solo puede lanzar un proceso núcleo el proceso núcleo actual, y para que esto suceda además el proceso núcleo a lanzar tiene que tener el mismo namespace y clase de GEDEE que el núcleo actual, también debe estar listado como permitido en la sección Valores referentes a Bloques de procesos/procesos/seguridad_procesos/permision_nucleo del fichero de configuración del sistema Sockeletom. El nuevo proceso núcleo puede no matar la ejecución de php, de ser así el termino de este proceso le cede el control al proceso núcleo anterior.

Programador núcleo: La definición y control de la ejecución de procesos está implementada en la función controlEjecucionProcesos() en la clase ImplementacionProcesos. Cuando un proceso núcleo ejecuta otro proceso núcleo se mantienen el el objeto EEoNucleo las variables lsNamespaceGedeeProcesoEjecucion y lsClaseGedeeProcesoEjecucion porque es obligatorio que el proceso núcleo a ejecutar sea del mismo namespace y clase de GEDEE que el proceso núcleo actual(ejecutándose), solo se actualizan las variables lsIdProcesoNucleo y lsIdGedeeProcesoNucleo, todo este intercambio se realiza en los procedimientos actualizacionComienzoProceso() y actualizacionFinalizacionProceso del objeto EEoNucleo.

Programador núcleo: La seguridad de los datos de los procesos tipo núcleo que se gestionan en las Entidades Esquelemod también se simplificó, se obligó a que la gestión de esos datos sea estrictamente gestionados si el proceso que los solicita es un proceso de tipo núcleo, estas obligaciones se declaran en el GEDEE, los procedimientos de seguridad de este GEDEE se mantienen existiendo solo su nombre y parámetros pero con una gestión nula, esto se debe a que como solo puede acceder a los datos de todo tipo y ejecución de un proceso núcleo, otro proceso núcleo, la seguridad no es configurable ya viene predeterminada en los demás procedimientos de este GEDEE. Esta decisión está sujeta a discusión ya que puede ser flexibilizada y funcionar parecido a como lo hacen los procesos con GEDEE Ecomun, es por eso que existe código comentado en:

e_nucleo_control.php: seccion inicializacion de los datos de seguridad del proceso núcleo del sistema Sockeletom.

Class_enucleo_conrol.php: public function iniciacionDatosSeguridadEsquelemod()

también están involucrados en esta decisión de seguridad los procedimientos y ficheros mencionados en el comentario para programador núcleo anterior a este.

Entidades Esquelemod

Las entidades esquelemod son objetos que conforman el núcleo del sistema, son visibles desde cualquier parte del Sockeletom y están al servicio de cualquier código que se ejecute en el ambiente Sockeletom. Cada uno de estos objetos tiene un objetivo específico, responsable de las

prestaciones fundamentales de este framework o motor de procesos, estas entidades y sus prestaciones son:

Entidad EEoNucleo.

Prestaciones que brinda el núcleo Sockeletom para el trabajo con los procesos. Estas prestaciones vienen dadas por un grupo de procedimientos públicos accesibles en el objeto núcleo, este objeto esta accesible desde cualquier ámbito (global o local) en la variable $EEoNucleo en caso de necesitar hacer una petición mas directa de una referencia a este objeto se puede hacer implementando el siguiente procedimiento y sus parámetros:

```
\Emod\Nucleo\CropNucleo::referenciarObjeto( 'EEoNucleo' , 'NucleoControl' , 'Emod\Nucleo' )
```

La instanciación de la clase (fichero), clase con nombre NucleoControl y fichero donde se aloja la clase con nombre class_enucleo_control.php. instanciada por el control del núcleo, y con patrón singlenton, generará un objeto de nombre EEoNucleo que contiene propiedades y procedimientos para el control de los procesos que harán uso del Sockeletom como plataforma, y también brinda datos que pueden ser de interés de estos procesos para su gestión. Este objeto y sus propiedades y métodos públicos estarán al alcance de todos los códigos clientes en forma de la variable $EEoNucleo.

A continuación mostramos una lista de las propiedades y procedimientos que este objeto pone a disposición de los códigos clientes del Sockeletom, se seccionaran de acuerdo a la posibilidad real de ejecutarlos satisfactoriamente, es decir existen procedimientos que por ser públicos no están al alcance de una ejecución satisfactoria para cualquiera que lo invoque ya que están sujetos a factores determinantes para su ejecución:

Dominio de todos

- procedimiento pathDirEsquelemod
- procedimiento pathDirModulos
- procedimiento idProcesoNucleo
- procedimiento gedeeProcesoNucleo
- procedimiento idGedeeProcesoNucleo
- procedimiento namespaceGedeeProcesoNucleo
- procedimiento claseGedeeProcesoNucleo
- procedimiento accesoPropiedadesEsquelemod
- procedimiento permisionProcesoNucleo
- procedimiento permisionProcesoCliente
- procedimiento bloqueEjecucionProcesos
- procedimiento pathDirRaizProcesos
- procedimiento pathDirRaizProcesoEjecucion
- procedimiento pathAbsolutoDirRaizProcesoEjecucion

- procedimiento idProcesoEjecucion
- procedimiento gedeeProcesoEjecucion
- procedimiento idGedeeProcesoEjecucion
- procedimiento namespaceGedeeProcesoEjecucion
- procedimiento claseGedeeProcesoEjecucion
- procedimiento accederHistorialArbolProcesos
- procedimiento accederPropiedadesProcesoPadre

Dominio del controlador del proceso núcleo
- procedimiento iniciacionImplementacionNucleo
- procedimiento gestionIniciacionConfiguracionEsquelemod
- procedimiento iniciacionDatosSeguridadEsquelemod
- procedimiento ejecutarBloquesProcesos

Dominio del objeto EEoImplementacionProcesos
- procedimiento actualizacionComienzoProceso
- procedimiento actualizacionFinalizacionProceso

Una descripción detallada del código se encuentra en class_enucleo_control.php.

Entidad EEoConfiguracion.

Prestaciones que vienen dadas por un grupo de procedimientos públicos accesibles en un objeto para la gestión y contención de datos de configuración, este objeto esta accesible desde cualquier ámbito (global o local) en la variable $EEoConfiguracion en caso de necesitar hacer una petición mas directa de una referencia a este objeto se puede hacer implementando el siguiente procedimiento y sus parámetros:
\Emod\Nucleo\CropNucleo::referenciarObjeto('EEoConfiguracion' , 'ConfiguracionProcesos' , 'Emod\Nucleo')
Una explicación más detallada de la estructura de este objeto se encuentra en en este mismo documento.

La instanciación de la clase (fichero), clase con nombre ConfiguracionProcesos y fichero donde se aloja la clase con nombre class_econfiguracion_procesos.php, instanciada por el núcleo, y con patrón singlenton, generará un objeto de nombre EEoConfiguracion que está compuesto por una propiedad que alojará los datos de las configuraciones de los procesos, y procedimientos para la gestión interna de estos datos. Este objeto y sus propiedades y métodos públicos estarán al alcance de todos los códigos clientes en forma de la variable $EEoConfiguracion.

A continuación mostramos una lista de las propiedades y procedimientos que este objeto pone a disposición de los códigos clientes del Sockeletom:

Dominio de todos

- procedimiento iniciarDatosConfiguracionProceso
- existenciaIdProceso
- accederDatosConfiguracionProceso

Una descripción detallada del código se encuentra en class_econfiguracion_procesos.php.

Entidad EEoSeguridad.

Prestaciones que vienen dadas por un grupo de procedimientos públicos accesibles en un objeto para la gestión y contención de datos de seguridad, este objeto esta accesible desde cualquier ámbito (global o local) en la variable $EEoSeguridad en caso de necesitar hacer una petición mas directa de una referencia a este objeto se puede hacer implementando el siguiente procedimiento y sus parámetros:

```
\Emod\Nucleo\CropNucleo::referenciarObjeto('EEoSeguridad' , 'SeguridadProcesos' , 'Emod\Nucleo')
```

Una explicación más detallada de la estructura de este objeto se encuentra en $EEoSeguridad en este mismo documento.

La instanciación de la clase (fichero), clase con nombre SeguridadProcesos y fichero donde se aloja la clase con nombre class_eseguridad_procesos.php, instanciada por el núcleo, y con patrón singlenton, generará un objeto de nombre EEoSeguridad que está compuesto por una propiedad que alojará los datos de seguridad de los procesos, y procedimientos para la gestión interna de estos datos. Este objeto y sus propiedades y métodos públicos estarán al alcance de todos los códigos clientes en forma de la variable $EEoSeguridad.

A continuación mostramos una lista de las propiedades y procedimientos que este objeto pone a disposición de los códigos clientes del Sockeletom:

Dominio de todos

- procedimiento iniciarDatosSeguridadProceso
- existenciaIdProceso
- accederDatosSeguridadProceso
- clienteEjecucionProceso
- clienteAccesoDatosProceso
- clienteAccesoConfiguracionProceso

Una descripción detallada del código se encuentra en class_eseguridad_procesos.php.

Entidad EEoDatos.

Prestaciones que vienen dadas por un grupo de procedimientos públicos accesibles en un objeto para la gestión y contención de datos para el trabajo interno del proceso, o para el intercambio con otros procesos, este objeto esta accesible desde cualquier ámbito (global o local) en la variable $EEoDatos en caso de necesitar hacer una petición mas directa de una referencia a este objeto se puede hacer implementando el siguiente procedimiento y sus parámetros:
\Emod\Nucleo\CropNucleo::referenciarObjeto('EEoDatos' , 'DatosProcesos' , 'Emod\Nucleo')
Una explicación más detallada de la estructura de este objeto se encuentra en $EEoDatos en este mismo documento.

La instanciación de la clase (fichero), clase con nombre DatosProcesos y fichero donde se aloja la clase con nombre class_edatos_procesos.php instanciada por el núcleo, y con patrón singlenton, generará un objeto de nombre EEoDatos que está compuesto por una propiedad que alojará los datos que estimen convenientes los procesos, y procedimientos para la gestión interna de estos datos. Este objeto y sus propiedades y métodos públicos estarán al alcance de todos los códigos clientes en forma de la variable $EEoDatos.

A continuación mostramos una lista de las propiedades y procedimientos que este objeto pone a disposición de los códigos clientes del Sockeletom:

Dominio de todos
- procedimiento iniciarDatosSalidaProceso
- existenciaIdProceso
- accederDatosSalidaProceso

Una descripción detallada del código se encuentra en class_edatos_procesos.php.

Entidad EEoImplementacionProcesos.

Prestaciones que vienen dadas por un grupo de procedimientos públicos accesibles en un objeto para la gestión y ejecución de procesos, este objeto esta accesible desde cualquier ámbito (global) en la variable $EEoImplementacionProcesos, y (local) todos los procesos se ejecutan dentro del objeto $EEoImplementacionProcesos por lo que tienen acceso a sus procedimientos y propiedades, en caso de necesitar hacer una petición mas directa de una referencia a este objeto se puede hacer implementando el siguiente procedimiento y sus parámetros:
\Emod\Nucleo\CropNucleo::referenciarObjeto('EEoImplementacionProcesos ' , 'ImplementacionProcesos ' , 'Emod\Nucleo')
Una explicación más detallada de la estructura de este objeto se encuentra en $EEoImplementacionProcesos en este mismo documento.

La instanciación de la clase (fichero), clase con nombre ImplementacionProcesos y fichero donde se aloja la clase con nombre class_eimplementacion_procesos.php. Instanciada por el núcleo, y con patrón singlenton, generará un objeto de nombre EEoImplementacionProcesos que está compuesto por propiedades, y procedimientos para la ejecución de procesos, esta ejecución ocurre de forma encapsulada en este objeto. Este objeto y sus propiedades y métodos estarán al alcance de todos los códigos clientes, ya que el código cliente se ejecuta como parte de este objeto.

A continuación mostramos una lista de las propiedades y procedimientos que este objeto pone a disposición de los códigos clientes del Sockeletom:

Dominio de todos
- procedimiento cargaControlProcesos

Una descripción detallada del código se encuentra en class_eimplementacion_procesos.php.¨

Entidad EEoInterfazDatos.

Prestaciones que vienen dadas por un grupo de procedimientos públicos accesibles en un objeto para la gestión y caché de datos de forma general, en diferentes fuentes de datos, este objeto esta accesible desde cualquier ámbito (global o local) en la variable $EEoInterfazDatos en caso de necesitar hacer una petición mas directa de una referencia a este objeto se puede hacer implementando el siguiente procedimiento y sus parámetros:
\Emod\Nucleo\CropNucleo::referenciarObjeto('EEoInterfazDatos' , 'InterfazDatos' , 'Emod\Nucleo\Herramientas')
Una explicación más detallada de la estructura de este objeto se encuentra en $EEoInterfazDatos en este mismo documento.

La instanciación de la clase (fichero), clase con nombre InterfazDatos y fichero donde se aloja la clase con nombre class_einterfaz_datos.php. Instanciada por el núcleo, y con patrón singlenton, generará un objeto de nombre EEoInterfazDatos que está compuesto por propiedades, y procedimientos para la gestión de datos en diferentes formatos y la implementación de cache de estos datos. La gestión de datos es de lectura y también de modificación de estos datos. Este objeto y sus propiedades y métodos estarán al alcance de todos los códigos clientes.

A continuación mostramos una lista de las propiedades y procedimientos que este objeto pone a disposición de los códigos clientes del Sockeletom:

Dominio de todos
- constructor
- iniciacionImplementacionInterfazDatos
- gestionEjecucionInterfazSalida

Una descripción detallada del código se encuentra en class_einterfaz_datos.php.

Existe otra entidad esquelemod que por motivos de organización no se encuentra en esta sección pero que no deja de ser tan importante como las demás, esta entidad es EEoErrores y se encuentra su explicación en la sección de Gestión y Tratamiento de Errores en este mismo documento.

GEDEEs

Gestor de Estructuras de Datos en Entidades Esquelemod (GEDEE)

El uso en sus diferentes formas de la frase GEDEEs en el ambiente Sockeletom, se debe a la necesidad de crear un servicio versátil en la creación y soporte de gestores para el tratamiento de estructuras de datos, tratamiento por parte de los procesos, sistemas o aplicaciones que tengan como plataforma el sistema Sockeletom. Se refiere a poder crear o utilizar gestores que manejen estructuras de datos hechas específicamente para un tipo de proceso, estos gestores, manejadores de estructuras de datos, permitirán la coexistencia de procesos con estructuras de datos diferente y con la posibilidad de intercambio de datos entre estos diferentes procesos, todo sobre una misma plataforma y sus herramientas y espacios para el desempeño e intercambio entre procesos.

El sistema Sockeletom pone a disposición de los procesos que se ejecutaran sobre él, un grupo de entidades objetos como EEoNucleo, EEoConfiguracion, EEoSeguridad, EEoDatos, que son espacios contenedores de datos (propiedades de estos objetos), y las herramientas (procedimientos de estos objetos) para la gestión de estos datos. Un GEDEE controla y dirige como es la estructura de los datos que guardarán en esos contenedores y como se accede a esas estructuras de datos, esto permite que coexistan diferentes formas de estructuras de datos en un mismo contenedor (objeto); Y que procesos con diferentes naturaleza de GEDEEs, puedan intercambiar datos entre ellos, siendo sus diferencias en cuanto a estructuras de datos, transparentes para los procesos.

A los GEDEEs se les puede considerar o llamar interfaces entre los procesos y las estructuras de datos que se guardan y gestionan en las diferentes entidades objetos antes mencionados, y que nos brinda el núcleo Sockeletom.

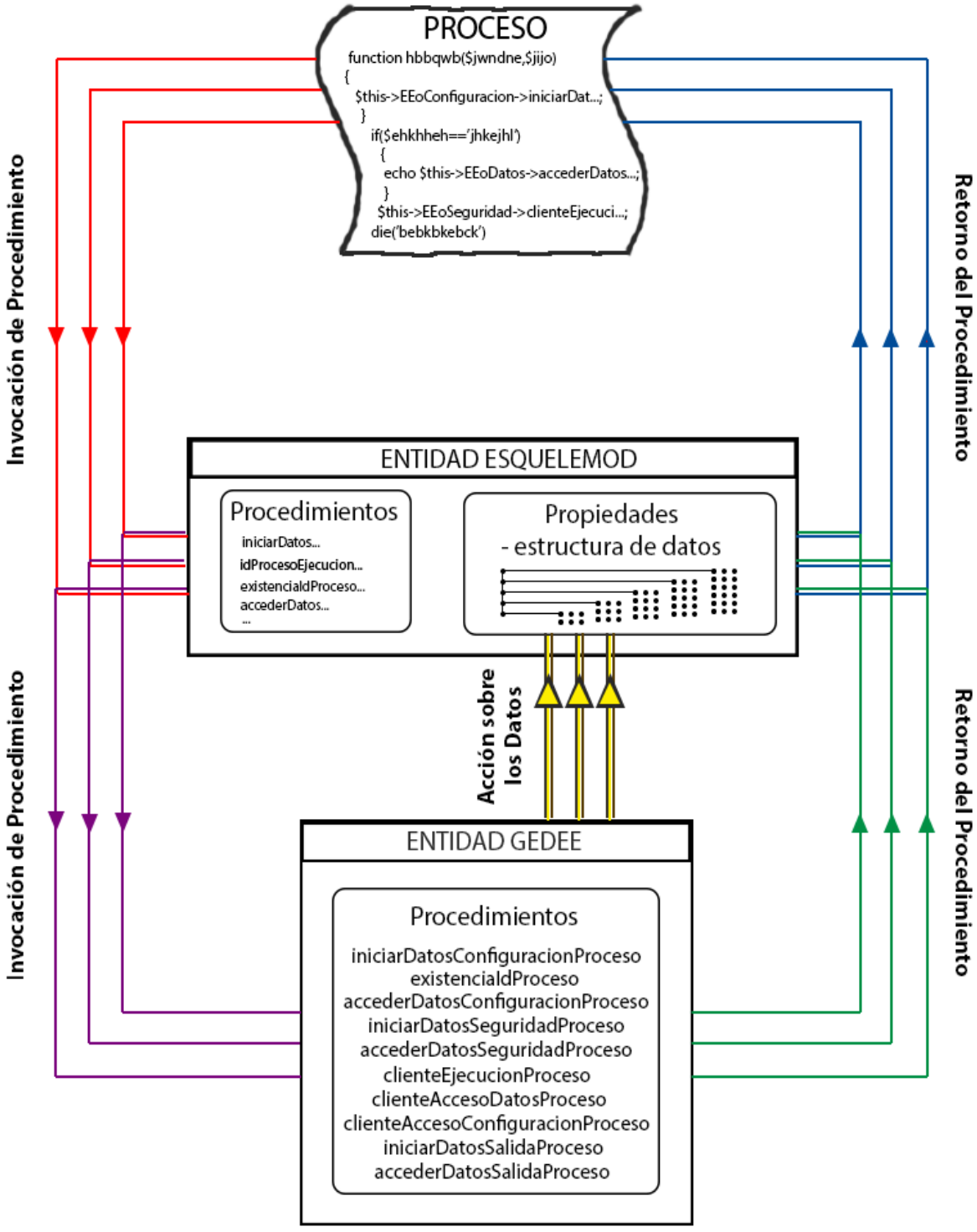
A los GEDEEs les corresponde un lugar en la estructura de directorios del sistema Sockeletom, este lugar tiene como raíz el directorio raíz_sistema_Sockeletom/esquelemod/e_modulos, y a partir de esta raíz el path declarado en la configuración del sistema Sockeletom, en el elemento
gedees: path_dir_raiz
que contiene como valor por defecto: gedees. Cada GEDEE tendrá como raíz la expuesta anteriormente, y a partir de esta raíz es que se guardan los directorios contenedores de cada GEDEE específico, es decir si el GEDEE se encuentra en ‘mi_gedee/sub_gedee’, entonces el path absoluto quedaría:
raíz_sistema_Sockeletom/esquelemod/e_modulos/gedees/mi_gedee/sub_gedee/.

Es en la configuración del sistema Sockeletom donde se declaran los GEDEEs y sus aspectos, que se gestionaran por parte del sistema Sockeletom, si no existe el GEDEE declarado en la configuración y además permitido su ejecución, no es un GEDEE válido o permitido para el sistema y por tanto no se gestionará. En la configuración del sistema Sockeletom existe una sección para la declaración de los GEDEEs y sus propiedades, para un mejor entendimiento de esta sección se debe consultar configuración del sistema Sockeletom en este mismo documento.

El sistema Sockeletom trae por defecto varios GEDEEs de los que se puede copiar su estructura de directorios y ficheros y modificar estos ficheros según los requerimientos de un nuevo GEDEE.

Cada GEDEE tendrá procedimientos con nombres idénticos a los procedimientos que contienen las entidades EEoConfiguracion, EEoSeguridad, EEoDatos, y un parámetro como entrada de datos por referencia que sería para facilitar al procedimiento la estructura de datos de las entidades EEoConfiguracion, EEoSeguridad, EEoDatos, con el objetivo de que quien hace realmente la gestión de datos(sub estructura de datos) en la estructura de datos de la entidad EEoConfiguracion, EEoSeguridad, EEoDatos es el procedimiento del GEDEE, de esa forma pueden coexistir diferentes sub estructuras de datos y múltiples procedimientos con un nombre común para gestionar estas sub estructuras de datos. Para un mejor entendimiento consultar el GEDEE GedeeEComun en este mismo documento.

La imagen a continuación describe lo anterior:


PROCESO
function hbbqwb($jwndne,$jijo)
{
$this->EEoConfiguracion->iniciarDat...;
}
if($ehkhheh=='jhkejhl')
{
echo $this->EEoDatos->accederDatos...;
}
$this->EEoSeguridad->clienteEjecuci...;
die('bebkbkebck')
Invocación de Procedimiento
Retorno del Procedimiento
ENTIDAD ESQUELEMOD
Procedimientos
iniciarDatos...
idProcesoEjecucion...
existencialdProceso...
accederDatos...
...
Propiedades
- estructura de datos
Acción sobre los Datos
Invocación de Procedimiento
Retorno del Procedimiento
ENTIDAD GEDEE
Procedimientos
iniciarDatosConfiguracionProceso
existencialdProceso
accederDatosConfiguracionProceso
iniciarDatosSeguridadProceso
accederDatosSeguridadProceso
clienteEjecucionProceso
clienteAccesoDatosProceso
clienteAccesoConfiguracionProceso
iniciarDatosSalidaProceso
accederDatosSalidaProceso

Los GEDEEs son clases estáticas u objetos instancia de una clase que pueden ser iniciados, chequeados su estado de gestión inicial, o controlados por lo que se ha llamado, por parte de los desarrolladores del Sockeletom, un fichero control, puede ser directamente a través de algunos datos obligatorios o a través de un fichero control, ambas formas aportan los datos necesarios a quien realmente gestionaran los GEDEEs, que es la clase Emod\Nucleo\GEDEEs

Existe una clase abstracta para definir un grupo de procedimientos obligados en la implementación de GEDEEs, una explicación de esta se encuentra mas adelante. Un ejemplo de GEDEE se encuentra por defecto en el Sockeletom, este es el GEDEE con nombre de clase GedeeEComun cuya estructura y funcionamiento se encuentran explicados con detalle en class_gedee_ecomun.php.

Las propiedades de los GEDEEs y sus entidades no son manejadas directamente por el objeto EEoNucleo , ya que existe una clase de métodos estáticos que se encarga del trabajo con los GEDEEs, esta clase se subordina al objeto EEoNucleo y se encarga de la gestión y administración de los GEDEEs, la clase de nombre GEDEEs y namespace Emod\Nucleo se encuentra en el fichero de nombre class_gedees.php del directorio esquelemod/e_modulos/núcleo/nucleo_bib_funciones del sistema Sockeletom.

A continuación mostramos una lista de las propiedades y procedimientos que esta clase pone a disposición de los códigos clientes del Sockeletom:

Dominio de todos

- iniciación
- pathRaizEntidades()
- actualizarDatosSeguridadEntidades
- permisionEntidad
- gestionControlIngresoEntidad
- gestionIngresoEntidad
- gestionIngresosEntidades
- existenciaEntidad
- ingresarEntidad
- entidad
- datosEntidad
- idEntidades
- eliminarEntidad

Una descripción detallada del código se encuentra en class_gedees.php

Como se explicó anteriormente el sistema Sockeletom trae consigo algunos GEDEEs que pone a disposición de los procesos, a continuación los explicamos:

GedeeEPadre

Este GEDEE y la clase que lo implementa son una guía obligatoria para la creación de GEDEEs, puesto que define de forma abstracta los procedimientos básicos a implementar por cualquier GEDEE que se cree, además responde de forma lineal o equivalente a los nombres de procedimientos soportados por las entidades EEoConfiguracion, $EEoSeguridad, $EEoDatos partes del núcleo Sockeletom.

Una descripción detallada de la clase y fichero que implementan el GedeeEPadre la encontramos en class_gedee_epadre.php

A continuación mostramos una lista de las propiedades y procedimientos que esta clase pone a disposición de los códigos clientes del Sockeletom:

Dominio de todos

- existenciaIdProceso
- iniciarDatosConfiguracionProceso
- accederDatosConfiguracionProceso
- iniciarDatosSeguridadProceso
- accederDatosSeguridadProceso
- clienteEjecucionProceso
- clienteAccesoDatosProceso
- clienteAccesoConfiguracionProceso
- iniciarDatosSalidaProceso
- accederDatosSalidaProceso

GedeeENucleo

Es un GEDEE concebido con una estructura de datos y permisos para su uso por el o los proceso núcleo del sistema Sockeletom. A continuación explicaremos las estructuras de datos para la que está concebida este GEDEE, es decir, las estructuras de datos que los procedimientos de este GEDEE gestionan, comprenden, esperan se les de como parámetro para una correcta ejecución. Las estructuras solo constaran de un namespace y una clase puesto que la concepción actual del sistema Sockeletom admite la ejecución de procesos tipo núcleo de un solo namespace\clase GEDEE. A continuación las estructuras:

• para los procedimientos con función núcleo que gestionan sobre la entidad esquelemod EEoConfiguracion:

```
array:
        namespace_gedee_proceso:
```

clase_gedee:
id_proceso1:
(datos de configuración)
id_procesoN:
(datos de configuración)

namespace_gedee_proceso: es el namespace del GEDEE con el que trabaja el proceso que pretende tener sus datos de configuración en esta estructura.

clase_gedee: es la clase del GEDEE con el que trabaja el proceso que pretende tener sus datos de configuración en esta estructura.

id_proceso: nombre o identificador de proceso que pretende tener sus datos de configuración en esta estructura.

datos de configuración: estructura de datos de configuración.

- para los procedimientos que gestionan sobre la entidad esquelemod EEoSeguridad:

Los procedimientos sobre la entidad esquelemod EEoSeguridad de este GEDEE se mantienen existiendo solo su nombre y parámetros pero con una gestión nula, esto se debe a que como solo puede acceder a los datos de todo tipo y ejecución de un proceso núcleo, otro proceso núcleo, la seguridad no es configurable ya viene predeterminada en los demás procedimientos de este GEDEE, cada procedimiento gestiona solo si su cliente es un proceso núcleo. Los procedimientos de seguridad existen para no generar errores en caso de que los soliciten, pero siempre retornarán un valor null.

- para los procedimientos que gestionan sobre la entidad esquelemod EEoDatos:

array:
namespace_gedee_proceso:
clase_gedee:
id_proceso1:
(datos de salida)
id_procesoN:
(datos de salida)

namespace_gedee_proceso: es el namespace del GEDEE con el que trabaja el proceso que pretende tener sus datos de salida o trabajo en esta estructura.

clase_gedee: es la clase del GEDEE con el que trabaja el proceso que pretende tener sus datos de salida o trabajo en esta estructura.

id_proceso: nombre o identificador de proceso que pretende tener sus datos de salida o trabajo en esta estructura.

datos de salida: estructura de datos de salida o trabajo.

Después de haber explicado cada estructura de datos le brindamos el acceso a la explicación de la clase con sus propiedades y los procedimiento que implementa toda la gestión de este GEDEE, esta clase se encuentra en el fichero class_gedee_enucleo.php descrita en este documento.

Gedee EComun

Es un GEDEE concebido con una estructura de datos simple para su uso por procesos que no tengan necesidades específicas en las estructuras de datos de su GEDEE. A continuación explicaremos las estructuras de datos para la que está concebida este GEDEE, es decir, las estructuras de datos que los procedimientos de este GEDEE gestionan, comprenden, esperan se les de como parámetro para una correcta ejecución. A continuación las estructuras:

- para los procedimientos que gestionan sobre la entidad esquelemod EEoConfiguracion:

```
array:
    namespace_gedee_proceso1:
                                clase_gedee1:
                                            id_proceso1:
                                                        (datos de configuración)
                                             id_procesoN:
                                                         (datos de configuración)
                                clase_gedeeN:
                                             id_proceso1:
                                                        (datos de configuración)
                                             id_procesoN:
                                                        (datos de configuración)
    namespace_gedee_procesoN:
                                 clase_gedee1:
                                             id_proceso1:
                                                        (datos de configuración)
                                             id_procesoN:
                                                        (datos de configuración)
                                 clase_gedeeN:
                                             id_proceso1:
                                                        (datos de configuración)
```

```
                                                id_procesoN:
                                                        (datos de configuración)
```

namespace_gedee_proceso: es el namespace del GEDEE con el que trabaja el proceso que pretende tener sus datos de configuración en esta estructura.

clase_gedee: es la clase del GEDEE con el que trabaja el proceso que pretende tener sus datos de configuración en esta estructura.

id_proceso: nombre o identificador de proceso que pretende tener sus datos de configuración en esta estructura.

datos de configuración: estructura de datos de configuración.

- para los procedimientos que gestionan sobre la entidad esquelemod EEoSeguridad:

```
array:
    namespace_gedee_proceso1:
        clase_gedee_proceso1:
            id_proceso1:
              permiso_ejecucion:
                    ambito:
                    procesos:
                        namespace_gedee_proceso_cliente1:
                            clase_gedee_proceso_cliente1:
                                - id_proceso_cliente1
                                - id_proceso_clienteN
                            clase_gedee_proceso_clienteN:***
                        namespace_gedee_proceso_clienteN:***
              acceso_seguridad:
                    ambito:
                    procesos:
                        namespace_gedee_proceso_cliente1:
                            clase_gedee_proceso_cliente1:
                                id_proceso_cliente1:
                                d_proceso_clienteN:
                            clase_gedee_proceso_clienteN:***
                        namespace_gedee_proceso_clienteN:***
              acceso_configuracion:
                   ambito:
                    procesos:
                        namespace_gedee_proceso_cliente1:
```

```
                                                clase_gedee_proceso_cliente1:
                                                        id_proceso_cliente1:
                                                        id_proceso_clienteN:
                                                clase_gedee_proceso_clienteN:***
                                        namespace_gedee_proceso_clienteN:***
                        acceso_datos:
                                ambito:
                                procesos:
                                        namespace_gedee_proceso_cliente1:
                                                clase_gedee_proceso_cliente1:
                                                        id_proceso_cliente1:
                                                        id_proceso_clienteN:
                                                clase_gedee_proceso_clienteN:***
                                        namespace_gedee_proceso_clienteN:***
                        id_procesoN:***
                clase_gedee_procesoN:***
        namespace_gedee_procesoN: ***
```

namespace_gedee_proceso: es el namespace del GEDEE con el que trabaja el proceso que pretende tener sus datos de seguridad en esta estructura.

clase_gedee_proceso: es la clase del GEDEE con el que trabaja el proceso que pretende tener sus datos de seguridad en esta estructura.

id_proceso: nombre o identificador de proceso que pretende tener sus datos de seguridad en esta estructura.

permiso_ejecucion: arreglo asociativo donde se guardarán datos relacionados con el permiso de ejecución del id_proceso de gedee namespace_gedee_proceso\clase_gedee, en esta estructura se definen los procesos que tienen o no, permiso de ejecutar este id_proceso, esta estructura tiene varios elementos que son:
1- ambito es un string cuyos valores posibles son 'permisivo' y 'restrictivo', este elemento es una condicionante del elemento procesos, si el valor es 'permisivo' se declara que todos los procesos excepto los contenidos en el elemento procesos, tienen el permiso de ejecutar este id_proceso, si el valor es 'restrictivo' se declara que todos los procesos excepto los contenidos en el elemento procesos, no tienen el permiso de ejecutar este id_proceso.

2- procesos es un array asociativo con identificador 'procesos' que contiene los procesos que pueden o no ejecutar este id_proceso. Este elemento está condicionado por el elemento ambito.

3- namespace_gedee_proceso_cliente es un array asociativo con identificador nombre del namespace del gedee que utiliza el proceso que puede o no ejecutar este id_proceso.

4- clase_gedee_proceso_cliente es un array asociativo con identificador nombre de la clase del gedee que utiliza el proceso que puede o no ejecutar este id_proceso.

5- id_proceso_cliente es un string identificador del proceso que puede o no ejecutar este id_proceso, si el primer elemento id_proceso_cliente es un '*', el permiso se extiende a todos los procesos del namespace_gedee_proceso_cliente\clase_gedee_proceso_cliente correspondiente.

acceso_seguridad: arreglo asociativo donde se guardarán datos relacionados con el permiso de acceso a datos de seguridad del id_proceso de gedee namespace_gedee_proceso\clase_gedee, en esta estructura se definen los procesos que tienen o no, permiso de leer, modificar_escribir, o eliminar datos de seguridad pertenecientes a este id_proceso, esta estructura tiene varios elementos que son:
1- ambito es un string cuyos valores posibles son 'permisivo' y 'restrictivo', este elemento es una condicionante del elemento procesos, si el valor es 'permisivo' se declara que todos los procesos excepto los contenidos en el elemento procesos, tienen el permiso de leer, modificar_escribir, y eliminar los datos de seguridad de este id_proceso, si el valor es 'restrictivo' se declara que todos los procesos excepto los contenidos en el elemento procesos, no tienen el permiso de leer, modificar_escribir, y eliminar los datos de seguridad de este id_proceso. En el elemento id_proceso_cliente se precisan algunas especificidades.

2- procesos es un array asociativo con identificador 'procesos' que contiene los procesos que pueden o no acceder a leer, modificar_escribir, y/o eliminar los datos de seguridad de este id_proceso. Este elemento está condicionado por el elemento ambito.

3- namespace_gedee_proceso_cliente es un array asociativo con identificador nombre del namespace del gedee que utiliza el proceso que puede o no acceder a leer, modificar_escribir, y/o eliminar los datos de seguridad de este id_proceso.Si este elemento es un '*', el permiso se extiende a todos los procesos del namespace_gedee_proceso_cliente\ correspondiente

4- clase_gedee_proceso_cliente es un array asociativo con identificador nombre de la clase del gedee que utiliza el proceso que puede o no acceder a leer, modificar_escribir, y/o eliminar los datos de seguridad de este id_proceso.si este elemento es un '*', el permiso se extiende a todos los procesos del namespace_gedee_proceso_cliente\ clase_gedee_proceso_cliente correspondiente

5- id_proceso_cliente es un string identificador del proceso que puede o no acceder a leer, modificar_escribir, y/o eliminar los datos de seguridad de este id_proceso, a este identificador se le asignan los valores correspondientes al tipo de acceso, estos valores pueden ser una combinación de los string leer, escribir y eliminar, separados por el string ::. un ejemplo sería 'leer::escribir::eliminar'. si el primer elemento id_proceso_cliente es un '*', el permiso se extiende a todos los procesos del namespace_gedee_proceso_cliente\ clase_gedee_proceso_cliente correspondiente

acceso_configuracion: arreglo asociativo donde se guardarán datos relacionados con el permiso de acceso a datos de configuración del id_proceso de gedee namespace_gedee_proceso\clase_gedee, en esta estructura se definen los procesos que tienen o no, permiso de leer, modificar_escribir, o eliminar datos de configuración pertenecientes a este id_proceso, esta estructura tiene varios elementos que son:
1- ambito es un string cuyos valores posibles son 'permisivo' y 'restrictivo', este elemento es una condicionante del elemento procesos, si el valor es 'permisivo' se declara que todos los procesos excepto los contenidos en el elemento procesos, tienen el permiso de leer, modificar_escribir, y eliminar los datos de configuración de este id_proceso, si el valor es 'restrictivo' se declara que todos los procesos excepto los contenidos en el elemento procesos, no tienen el permiso de leer, modificar_escribir, y eliminar los datos de configuración de este id_proceso. En el elemento id_proceso_cliente se precisan algunas especificidades.

2- procesos es un array asociativo con identificador 'procesos' que contiene los procesos que pueden o no acceder a leer, modificar_escribir, y/o eliminar los datos de configuración de este id_proceso. Este elemento está condicionado por el elemento ambito.

3- namespace_gedee_proceso_cliente es un array asociativo con identificador nombre del namespace del gedee que utiliza el proceso que puede o no acceder a leer, modificar_escribir, y/o eliminar los datos de configuración de este id_proceso. Si este elemento es un '*', el permiso se extiende a todos los procesos del namespace_gedee_proceso_cliente\ correspondiente

4- clase_gedee_proceso_cliente es un array asociativo con identificador nombre de la clase del gedee que utiliza el proceso que puede o no acceder a leer, modificar_escribir, y/o eliminar los datos de configuración de este id_proceso.si este elemento es un '*', el permiso se extiende a todos los procesos del namespace_gedee_proceso_cliente\ clase_gedee_proceso_cliente correspondiente

5- id_proceso_cliente es un string identificador del proceso que puede o no acceder a leer, modificar_escribir, y/o eliminar los datos de configuración de este id_proceso, a este identificador se le asignan los valores correspondientes al tipo de acceso, estos valores pueden ser una combinación de los string leer, escribir y eliminar, separados por el string ::. un ejemplo sería 'leer::escribir::eliminar'. si el primer elemento id_proceso_cliente es un '*', el permiso se extiende a todos los procesos del namespace_gedee_proceso_cliente\ clase_gedee_proceso_cliente correspondiente.

acceso_datos: arreglo asociativo donde se guardarán datos relacionados con el permiso de acceso a datos de salida o trabajo del id_proceso de gedee namespace_gedee_proceso\clase_gedee, en esta estructura se definen los procesos que tienen o no, permiso de leer, modificar_escribir, o eliminar datos de salida o trabajo pertenecientes a este id_proceso, esta estructura tiene varios elementos que son:
1- ambito es un string cuyos valores posibles son 'permisivo' y 'restrictivo', este elemento es una condicionante del elemento procesos, si el valor es 'permisivo' se declara que todos los procesos excepto los contenidos en el elemento procesos, tienen el permiso de leer, modificar_escribir, y

eliminar los datos de salida o trabajo de este id_proceso, si el valor es 'restrictivo' se declara que todos los procesos excepto los contenidos en el elemento procesos, no tienen el permiso de leer, modificar_escribir, y eliminar los datos de salida o trabajo de este id_proceso. En el elemento id_proceso_cliente se precisan algunas especificidades.

2- procesos es un array asociativo con identificador 'procesos' que contiene los procesos que pueden o no acceder a leer, modificar_escribir, y/o eliminar los datos de salida o trabajo de este id_proceso. Este elemento está condicionado por el elemento ambito.

3- namespace_gedee_proceso_cliente es un array asociativo con identificador nombre del namespace del gedee que utiliza el proceso que puede o no acceder a leer, modificar_escribir, y/o eliminar los datos de salida o trabajo de este id_proceso. Si este elemento es un '*', el permiso se extiende a todos los procesos del namespace_gedee_proceso_cliente\ correspondiente

4- clase_gedee_proceso_cliente es un array asociativo con identificador nombre de la clase del gedee que utiliza el proceso que puede o no acceder a leer, modificar_escribir, y/o eliminar los datos de salida o trabajo de este id_proceso.si este elemento es un '*', el permiso se extiende a todos los procesos del namespace_gedee_proceso_cliente\ clase_gedee_proceso_cliente correspondiente

5- id_proceso_cliente es un string identificador del proceso que puede o no acceder a leer, modificar_escribir, y/o eliminar los datos de salida o trabajo de este id_proceso, a este identificador se le asignan los valores correspondientes al tipo de acceso, estos valores pueden ser una combinación de los string leer, escribir y eliminar, separados por el string ::. un ejemplo sería 'leer::escribir::eliminar'. si el primer elemento id_proceso_cliente es un '*', el permiso se extiende a todos los procesos del namespace_gedee_proceso_cliente\ clase_gedee_proceso_cliente correspondiente

- para los procedimientos que gestionan sobre la entidad esquelemod EEoDatos:

```
array:
        namespace_gedee_proceso1:
                                        clase_gedee1:
                                                        id_proceso1:
                                                                (datos de salida)
                                                        id_procesoN:
                                                                (datos de salida)
                                        clase_gedeeN:
                                                        id_proceso1:
                                                                (datos de salida)
                                                        id_procesoN:
                                                                (datos de salida)
        namespace_gedee_procesoN:
```

```
                    clase_gedee1:
                                id_proceso1:
                                            (datos de salida)
                                id_procesoN:
                                            (datos de salida)
                    clase_gedeeN:
                                id_proceso1:
                                            (datos de salida)
                                id_procesoN:
                                            (datos de salida)
```

namespace_gedee_proceso: es el namespace del GEDEE con el que trabaja el proceso que pretende tener sus datos de salida o trabajo en esta estructura.

clase_gedee: es la clase del GEDEE con el que trabaja el proceso que pretende tener sus datos de salida o trabajo en esta estructura.

id_proceso: nombre o identificador de proceso que pretende tener sus datos de salida o trabajo en esta estructura.

datos de salida: estructura de datos de salida o trabajo.

Después de haber explicado cada estructura de datos le brindamos el acceso a la explicación de la clase con sus propiedades y los procedimiento que implementa toda la gestión de este GEDEE, esta clase se encuentra en el fichero class_gedee_ecomun.php descrita en este documento.

Gestión y Tratamiento de Errores

El sistema Sockeletom cuenta con un grupo de herramientas destinadas al manejo de errores, con alcance a todas las áreas del sistema exceptuando el código que se ejecuta en el arranque del sistema y que se encuentra antes de la ejecución del módulo y motor gestor de errores.

Este gestor de errores puede ser invocado o no por los códigos clientes ya que no obstaculiza la utilización de otras formas de gestión de errores implantadas por códigos clientes.

Para el tratamiento de errores existe una sección en la configuración del sistema Sockeletom, para esta sección en la configuración del sistema Sockeletom existe en este mismo documento una explicación detallada, como también existe la explicación de cada clase y sus procedimientos que conforman el tratamiento de errores en este mismo documento, a continuación exponemos las herramientas que conforman la gestión de errores en el Sockeletom.

Entidad Errores

Comenzamos con la exposición de una clase de procedimientos estáticos y de nombre Errores, al alcance de todos los códigos clientes en la forma \Emod\Nucleo\Errores, que servirá como gestor y contenedor de entidades errores asequible desde cualquier parte del sistema Sockeletom y sus códigos clientes, antes debemos explicar que el grupo de desarrollo del Sockeletom llamará "entidades errores" a las clases de procedimientos estáticos, u objetos instancias de clase, que se gestionen como gestores de errores en esta clase "Errores" que a continuación exponemos:

Una clase (fichero), clase con nombre Errores y fichero donde se aloja la clase con nombre class_e_errores.php

Esta clase está compuesta por propiedades, y procedimientos estáticos para la gestión de entidades errores, desde la inclusión del script o clase como fichero hasta el retorno de referencias o clonaciones a objetos gestores de errores, gestiona y sirve como contenedor de estas entidades, una opción clara para la inyección de dependencias. En su gestión las entidades gestores de errores no son incluidas o instanciadas hasta el momento que se hace la petición de la instancia para su uso.

A continuación mostramos una lista de las propiedades y procedimientos que esta clase pone a disposición de los códigos clientes del Sockeletom:

Dominio de todos

- iniciacion
- pathRaizEntidades
- actualizarDatosSeguridadEntidades
- permisionEntidad
- gestionControlIngresoEntidad
- gestionIngresoEntidad
- gestionIngresosEntidades
- existenciaEntidad
- ingresarEntidad
- entidad
- datosEntidad
- idEntidades
- eliminarEntidad
- implantarGestorErrores
- implantarGestorErroresAnterior
- implantarGestorErroresPila
- gestorErroresEjecucion

Una descripción detallada del código se encuentra en: class_e_errores.php.

Entidad EEoErrores

Continuamos con un objeto al alcance de todos los códigos clientes procesos en la variable $this->EEoErrores, y para todos los códigos clientes en la forma variable global $EEoErrores o también haciendo uso de \Emod\Nucleo\CropNucleo::referenciarObjeto('EEoErrores', 'EErrores' , 'Emod\Nucleo\Errores'). Este objeto define el manejo de cada error específico, así como la fuente y las propiedades del error, su formato, y el destino errorlog junto a las propiedades y formato del errorlog; es quién ya iniciado y configurado nos permite con solo poner un identificador de error, hacer el tratamiento del error en sus acciones de implantar un manejador de errores personalizado, buscar las características del error en la fuente de estos, formatear el mensaje, mostrar el mensaje, detener o no el script en dependencia del tipo de error, formatear el mensaje de error log y ponerlo en su destino, implantar el manejador de errores anterior al actual(personalizado), estas y otras son las operaciones básicas que nos permite automatizar este objeto gestor de errores.

A continuación mostramos una lista de las propiedades y procedimientos que este objeto pone a disposición de los códigos clientes del Sockeletom:

Dominio de todos

- iniciacion
- gestorErroresEmod
- filtrarDatosError
- crearDatosError
- eliminarDatosError
- leerDatosError
- modificarDatosError
- crearFuenteDatosError
- eliminarFuenteDatosError
- filtrarDatosErrorlog
- crearDatosErrorLog
- eliminarDatosErrorLog
- leerDatosErrorLog
- modificarDatosErrorLog
- crearFuenteDatosErrorLog
- eliminarFuenteDatosErrorLog
- error

Una descripción detallada del código se encuentra en: class_eerrores.php.

GEDEGE

Por último se expone el concepto GEDEGE, que significa Gestor de Estructuras de Datos para Entidades Gestoras de Errores. Las entidades que clasifican dentro de este concepto se dedican a gestionar datos como intermediario entre la entidad que hará el tratamiento del error (gestor de errores) y la fuente de los datos del error. El Sockeletom tiene por defecto un objeto GEDEGE.

A continuación mostramos una lista de las propiedades y procedimientos que este objeto pone a disposición de los códigos clientes del Sockeletom:

Dominio de todos

- __construct
- filtrarDatosError
- crearDatosError
- eliminarDatosError
- leerDatosError
- modificarDatosError
- crearFuenteDatosError
- eliminarFuenteDatosError
- filtrarDatosErrorlog
- crearDatosErrorLog
- eliminarDatosErrorLog
- leerDatosErrorLog
- modificarDatosErrorLog
- crearFuenteDatosErrorLog
- eliminarFuenteDatosErrorLog

Una descripción detallada del código se encuentra en: class_gedege_txt_emod.

Configuración

En el sistema Sockeletom también tenemos una configuración, esta permite administrar un grupo de decisiones y prestaciones del sistema, administración de tipo implementación de prestaciones como también de tipo seguridad. Una descripción detallada del código se encuentra en e_sistema_config.cnf. A continuación una muestra de sus secciones:

identificador_referencia_local

gedees

sistema

herramientas

utiles

errores

procesos

PRESTACIONES

class_epatron_multiton

- (herramienta-patrones) una clase (fichero), clase con nombre Multiton y fichero donde se aloja la clase con nombre class_epatron_multiton.php.

Los patrones de diseño son bien conocidos en el mundo de la programación, en este caso se implementa el patrón singlenton, pero se implementa de forma múltiple en una sola clase tomando el nombre de Multiton.

A continuación mostramos una lista de las propiedades y procedimientos que esta clase pone a disposición de los códigos clientes :

Dominio de todos

- instanciar

Una descripción detallada del código se encuentra en class_epatron_multiton.php.

class_ecro_permanente

- (herramienta) una clase(fichero), clase con nombre CroPermanente y fichero donde se aloja la clase con nombre class_ecro_permanente.php

Esta clase abstracta, con métodos y procedimientos estáticos implementa contenedores de referencias a objetos sin opción de eliminar las referencias que ingresan al contenedor, el Sockeletom la utiliza como clase padre de otras clases más específicas.

A continuación mostramos una lista de las propiedades y procedimientos que esta clase pone a disposición de los códigos clientes :

Dominio de todos
- ingresarNuevoObjeto
- referenciarObjeto
- clonarObjeto
- existenciaObjeto

Una descripción detallada del código se encuentra en class_ecro_permanente.php

class_ecro_variable

- (herramienta) una clase(fichero), clase con nombre CroVariable y fichero donde se aloja la clase con nombre class_ecro_variable.php

Esta clase abstracta, con métodos y procedimientos estáticos implementa contenedores de referencias a objetos, el Sockeletom la utiliza como clase padre de otras clases más específicas.

A continuación mostramos una lista de las propiedades y procedimientos que esta clase pone a disposición de los códigos clientes del Sockeletom:

Dominio de todos
- ingresarNuevoObjeto
- referenciarObjeto
- clonarObjeto
- eliminarReferenciaObjeto

Una descripción detallada del código se encuentra en class_ecro_variable.php.

class_ecrop_nucleo

- una clase(fichero), clase con nombre CropNucleo y fichero donde se aloja la clase con nombre class_ecrop_nucleo.php

Esta clase con métodos y procedimientos estáticos sirve de contenedor de referencias permanentes a objetos del núcleo del sistema Sockeletom, hereda de la clase CroPermanente, y modifica el procedimiento clonarObjeto.

A continuación mostramos una lista de las propiedades y procedimientos que esta clase pone a disposición de los códigos clientes :

Dominio de todos

- ingresarNuevoObjeto
- referenciarObjeto
- clonarObjeto (con modificaciones)
- existenciaObjeto

Una descripción detallada del código se encuentra en class_ecrop_nucleo.php

trait_dependencias_entidades_emod

- un trait (fichero), trait con nombre DependenciasEntidadesEmod y fichero donde se aloja la clase con nombre trait_dependencias_entidades_emod.php.

Los objetos que conforman el núcleo del sistema Sockeletom comparten características, propiedades y procedimientos que por lógica de POO se agrupan en clases padres como esta.

A continuación mostramos una lista de las propiedades y procedimientos que esta clase pone a disposición de los códigos clientes :

Dominio de todos

- cargarObjetosDependencia

Una descripción detallada del código se encuentra en trait_dependencias_entidades_emod.php.

class_enucleo_entidad_base

- una clase (fichero), clase con nombre NucleoEntidadBase y fichero donde se aloja la clase con nombre class_enucleo_entidad_base.php.

Los objetos que conforman el núcleo del sistema Sockeletom comparten características, propiedades y procedimientos que por lógica de POO se agrupan en clases padres como esta.

Una descripción detallada del código se encuentra en class_enucleo_entidad_base.php.

class_enucleo_control

- una clase (fichero), clase con nombre NucleoControl y fichero donde se aloja la clase con nombre class_enucleo_control.php.

La instanciación de esta clase , con patrón singlenton, generará un objeto que contiene propiedades y procedimientos para el control de procesos al estilo Sockeletom como plataforma, y también brinda datos que pueden ser de interés de estos procesos para su gestión.

A continuación mostramos una lista de las propiedades y procedimientos que las instancias de esta clase ponen a disposición de los códigos clientes de esta, se seccionaran de acuerdo a la posibilidad real de ejecutarlos satisfactoriamente, es decir existen procedimientos que por ser públicos no están al alcance de una ejecución satisfactoria para cualquiera que lo invoque ya que están sujetos a factores determinantes para su ejecución:

Dominio de todos

- procedimiento pathDirEsquelemod
- procedimiento pathDirModulos
- procedimiento idProcesoNucleo
- procedimiento gedeeProcesoNucleo
- procedimiento idGedeeProcesoNucleo
- procedimiento namespaceGedeeProcesoNucleo
- procedimiento claseGedeeProcesoNucleo
- procedimiento accesoPropiedadesEsquelemod
- procedimiento permisionProcesoNucleo
- procedimiento permisionProcesoCliente
- procedimiento bloqueEjecucionProcesos
- procedimiento pathDirRaizProcesos
- procedimiento pathDirRaizProcesoEjecucion
- procedimiento pathAbsolutoDirRaizProcesoEjecucion
- procedimiento idProcesoEjecucion
- procedimiento gedeeProcesoEjecucion
- procedimiento idGedeeProcesoEjecucion
- procedimiento namespaceGedeeProcesoEjecucion
- procedimiento claseGedeeProcesoEjecucion
- procedimiento accederHistorialArbolProcesos
- procedimiento accederPropiedadesProcesoPadre
- procedimiento iniciacionImplementacionNucleo
- procedimiento gestionIniciacionConfiguracionEsquelemod
- procedimiento iniciacionDatosSeguridadEsquelemod
- procedimiento ejecutarBloquesProcesos
- procedimiento actualizacionComienzoProceso
- procedimiento actualizacionFinalizacionProceso

Una descripción detallada del código se encuentra en class_enucleo_control.php.

class_econfiguracion_procesos

- una clase(fichero), clase con nombre ConfiguracionProcesos y fichero donde se aloja la clase con nombre class_econfiguracion_procesos.php

La instanciación de esta clase, con patrón singlenton, generará un objeto que está compuesto por una propiedad que alojará los datos de las configuraciones de los procesos, y procedimientos para la gestión interna de estos datos, al estilo Sockeletom.

A continuación mostramos una lista de las propiedades y procedimientos que las instancias de esta clase ponen a disposición de los códigos clientes:

Dominio de todos

- procedimiento iniciarDatosConfiguracionProceso
- existenciaIdProceso
- accederDatosConfiguracionProceso

Una descripción detallada del código se encuentra en class_econfiguracion_procesos.php.

class_eseguridad_procesos

- una clase(fichero), clase con nombre SeguridadProcesos y fichero donde se aloja la clase con nombre class_eseguridad_procesos.php

La instanciación de esta clase, con patrón singlenton, generará un objeto que está compuesto por una propiedad que alojará los datos de seguridad de los procesos, y procedimientos para la gestión interna de estos datos, al estilo Sockeletom.

A continuación mostramos una lista de las propiedades y procedimientos que las instancias de esta clase ponen a disposición de los códigos clientes:

Dominio de todos

- procedimiento iniciarDatosSeguridadProceso
- existenciaIdProceso
- accederDatosSeguridadProceso
- clienteEjecucionProceso
- clienteAccesoDatosProceso
- clienteAccesoConfiguracionProceso

Una descripción detallada del código se encuentra en class_eseguridad_procesos.php.

class_edatos_procesos

- una clase(fichero), clase con nombre DatosProcesos y fichero donde se aloja la clase con nombre class_edatos_procesos.php

La instanciación de esta clase, con patrón singlenton, generará un objeto que está compuesto por una propiedad que alojará los datos que estimen convenientes los procesos, y procedimientos para la gestión interna de estos datos, al estilo Sockeletom.

A continuación mostramos una lista de las propiedades y procedimientos que las instancias de esta clase ponen a disposición de los códigos clientes:

Dominio de todos

- procedimiento iniciarDatosSalidaProceso
- existenciaIdProceso
- accederDatosSalidaProceso

Una descripción detallada del código se encuentra en class_edatos_procesos.php.

class_eimplementacion_procesos

- una clase (fichero), clase con nombre ImplementacionProcesos y fichero donde se aloja la clase con nombre class_eimplementacion_procesos.php.

La instanciación de esta clase, con patrón singlenton, generará un objeto que está compuesto por propiedades, y procedimientos para la ejecución de procesos, esta ejecución ocurre de forma encapsulada en este objeto, al estilo Sockeletom.

A continuación mostramos una lista de las propiedades y procedimientos que las instancias de esta clase ponen a disposición de los códigos clientes:

Dominio de todos

- procedimiento cargaControlProcesos

Una descripción detallada del código se encuentra en class_eimplementacion_procesos.php.¨

class_einterfaz_datos

- (herramienta) una clase(fichero), clase con nombre InterfazDatos y fichero donde se aloja la clase con nombre class_einterfaz_datos.php

La instanciación de esta clase, con patrón singlenton, generará un objeto compuesto por propiedades, y procedimientos para la gestión de datos en diferentes formatos y la implementación de cache de estos datos. La gestión de datos es de lectura y también de modificación de estos datos

A continuación mostramos una lista de las propiedades y procedimientos que las instancias de esta clase ponen a disposición de los códigos clientes:

Dominio de todos

- constructor
- iniciacionImplementacionInterfazDatos
- gestionEjecucionInterfazSalida

Una descripción detallada del código se encuentra en class_einterfaz_datos.php.

trait_egeco

- un trait(fichero), trait con nombre GECO y fichero donde se aloja la clase con nombre trait_egeco.php

Este trait con métodos y procedimientos estáticos es un Gestor de Entidades Clases y Objetos, se encarga desde la inclusión del script o clase como fichero hasta el retorno de referencias o clonaciones a objetos instancias, y clases, gestiona y sirve como contenedor de estas entidades, una opción clara para la inyección de dependencias. En su gestión las entidades no son incluidas o instanciadas hasta el momento que se hace la petición de la instancia para su uso.

A continuación mostramos una lista de las propiedades y procedimientos que esta clase pone a disposición de los códigos clientes:

Dominio de todos

- iniciacion
- pathRaizEntidades
- actualizarDatosSeguridadEntidades
- permisionEntidad
- gestionControlIngresoEntidad
- gestionIngresoEntidad
- gestionIngresosEntidades
- existenciaEntidad
- ingresarEntidad
- entidad
- datosEntidad

- idEntidades
- eliminarEntidad

Una descripción detallada del código se encuentra en: trait_egeco.php.

class_eherramientas

Antes de pasar a esta clases debemos especificar que considera el sistema Sockeletom una herramienta.

Herramienta: objeto, clase o función que se utiliza con un fin determinado en el desempeño de un código de programación, y que posteriormente a ser utilizado se prescinde de este hasta una nueva necesidad, sin que haya variado el estado inicial del objeto, clase o función. En el sistema Sockeletom existe un gestor de herramientas, donde se considera que cada herramienta parte de la implementación de una clase y puede ser accedida como clase estática, o como referencia o clon en caso de ser instancias de una clase, y cada acceso se hace sobre una misma entidad existente en el gestor contenedor de herramientas; cada clase permite solo una entidad a ser accedida.

Comenzaremos con la exposición de la clase de procedimientos estáticos y de nombre Herramientas, que servirá como gestor y contenedor de herramientas asequible desde cualquier parte del sistema Sockeletom y sus códigos clientes, antes debemos explicar que el grupo de desarrollo del Sockeletom llamará “entidades herramientas” a las clases de procedimientos estáticos, u objetos instancias de clase, que se gestionen como herramientas en esta clase “Herramientas” que a continuación exponemos:

Una clase (fichero), clase con nombre Herramientas y fichero donde se aloja la clase con nombre class_eherramientas.php

Esta clase está compuesta por propiedades, y procedimientos estáticos para la gestión de entidades herramientas, desde la inclusión del script o clase como fichero hasta el retorno de referencias o clonaciones a objetos herramientas, gestiona y sirve como contenedor de estas entidades, una opción clara para la inyección de dependencias. En su gestión las entidades herramientas no son incluidas o instanciadas hasta el momento que se hace la petición de la instancia para su uso.

A continuación mostramos una lista de las propiedades y procedimientos que este objeto pone a disposición de los códigos clientes :

Dominio de todos
- iniciacion
- pathRaizHerramientas
- actualizarDatosSeguridadHerramientas
- permisionHerramienta

- gestionControlIngresoHerramienta
- gestionIngresoHerramienta
- gestionIngresosHerramientas
- existenciaEntidadHerramienta
- entidadHerramienta
- datosEntidadHerramienta
- idEntidadHerramienta
- eliminarEntidadHerramienta

Una descripción detallada del código se encuentra en: class_eherramientas.php.

class_eutiles

Antes de pasar a estas clases y objetos debemos especificar que considera el sistema Sockeletom un útil.

Útil: objeto, clase o función que se utiliza con un fin determinado en el desempeño de un código de programación, y que al ser utilizado forma parte o no desde ese momento de la estructura lógica y de datos del código que lo utiliza; pero al ser utilizado se pueda o no cambiar el estado inicial del objeto, clase o función. En el sistema Sockeletom existe un gestor de útiles, donde se considera que cada útil parte de la implementación de una clase y puede ser accedida como clase estática, o como referencia o clon en caso de ser instancias de una clase, y cada acceso se hace sobre una entidad de las n entidades que puedan existir producto de una misma clase existente en el gestor contenedor de útiles; cada clase permite n entidades a ser accedida.

Continuamos con la exposición de la clase de procedimientos estáticos y de nombre Utiles, que servirá como gestor y contenedor de útiles asequible desde cualquier parte del sistema Sockeletom y sus códigos clientes, antes debemos explicar que el grupo de desarrollo del Sockeletom llamará "entidades útiles" a las clases de procedimientos estáticos, u objetos instancias de clase, que se gestionen como útiles en esta clase "Utiles" que a continuación exponemos:

Una clase (fichero), clase con nombre Utiles y fichero donde se aloja la clase con nombre class_eutiles.php

Esta clase está compuesta por propiedades, y procedimientos estáticos para la gestión de entidades útiles, desde la inclusión del script o clase como fichero hasta el retorno de referencias o clonaciones a objetos útiles, gestiona y sirve como contenedor de estas entidades, una opción clara para la inyección de dependencias. En su gestión las entidades útiles no son incluidas o instanciadas hasta el momento que se hace la petición de la instancia para su uso.

A continuación mostramos una lista de las propiedades y procedimientos que este objeto pone a disposición de los códigos clientes:

Dominio de todos
- iniciacion
- pathRaizEntidades
- actualizarDatosSeguridadEntidades
- permisionEntidad
- gestionControlIngresoEntidad
- gestionIngresoEntidad
- gestionIngresosUtiles
- existenciaEntidades
- entidades
- datosEntidad
- idEntidades
- eliminarEntidad

Una descripción detallada del código se encuentra en: class_eutiles.php.

class_herramienta_earreglo

- (herramienta) una clase(fichero), clase con nombre EArreglo y fichero donde se aloja la clase con nombre class_herramienta_earreglo.php

Esta clase con métodos y procedimientos estáticos sirve de contenedor de procedimientos para el tratamiento de arreglos, según el objetivo del procedimiento.

A continuación mostramos una lista de las propiedades y procedimientos que esta clase pone a disposición de los códigos clientes:

Dominio de todos
- arregloModificarRecursivo
- arregloEliminarRecursivo

Una descripción detallada del código se encuentra en class_herramienta_earreglo.php.

class_he_datfortxt

- (herramienta) una clase(fichero), clase con nombre EDatosFormatoTxt y fichero donde se aloja la clase con nombre class_he_datfortxt.php

Esta clase con métodos y procedimientos estáticos sirve de contenedor de procedimientos para el tratamiento de datos con formatos, en ficheros txt, según el objetivo del procedimiento.

A continuación mostramos una lista de las propiedades y procedimientos que esta clase pone a disposición de los códigos clientes del Sockeletom:

Dominio de todos

- crearNuevaLinea
- filtrarEliminarLineaDatos
- filtrarLeerLineaDatos
- fltrarModificarLineaDatos

Una descripción detallada del código se encuentra en class_he_datfortxt.php.

A continuación explicaremos algunos directorios y ficheros con sus códigos, del árbol de directorios del sistema Sockeletom.

Directorio Esquelemod

Es el directorio que contiene todo lo referente al sistema Sockeletom, es decir todo el código que es responsable del funcionamiento del Sockeletom como plataforma para aplicaciones o códigos cliente, contiene el código del núcleo Sockeletom, GEDEEs, los procesos y los útiles y herramientas que estarán a disposición de los códigos clientes, estos códigos clientes se encontrarán ubicados en otro directorio que no pertenece al contenido de este directorio esquelemod.

Configuración del sistema

La configuración del sistema Sockeletom se encuentra en
raíz_sistema_Sockeletom/esquelemod/e_sistema/e_sistema_config.cnf

Es un fichero con extensión .cnf para distinguirlo de las demás extensiones, pero es en esencia un txt, por defecto su contenido está estructurado para ser interpretado por un parser YAML, a continuación mostramos el contenido de la configuración y explicaremos cada una de sus partes.

```
# Fichero de configuracion del Sistema  Sockeletom
# version YAML
#
##############################################################################
#  Valor referente al identificador de  referencia local en la estructura de datos de este fichero#
##############################################################################
```

```
#mucho cuidado de no hacer referencias cíclicas porque traen problemas, este identificador tiene que cumplir las reglas de expresiones regulares para funciones PCRE
#
 identificador_referencia_local: localis_
#
###Identificador de referencia local (identificador_referencia_local) es una opción para reutilizar o utilizar una misma variable en múltiples lugares de este fichero de configuración si tener que escribirla nuevamente, es decir escribir una sola vez una variable y luego cuando se necesite invocarla en otro lugar de esta configuración solo como referencia.

El valor de esta variable (identificador_referencia_local) es un string que define que se está haciendo uso de una referencia a variable, seguido de este string en el lugar donde se hace la referencia debe existir también en forma de string la estructura de datos que contiene la variable referenciada, a continuación un ejemplo donde identificador_referencia_local tiene como valor localis_ :

identificador_referencia_local: localis_

Mi_arreglo:
        Elemento1: a
        Elemento2: b
        Elemento3: c

Arreglox:
        Elemento1: d
        Elemento2: localis_['Mi_arreglo']['Elemento3']
        Elemento3: e

En este ejemplo $Arreglox['Elemento2'] = $Mi_arreglo['Elemento3'] = c
```

```
##############################################################################
#  Valores referentes a los Gestores de Estructuras de Datos en Entidades Esquelemod #
##############################################################################

 gedees:

### path_dir_raiz Este será el fragmento de path que completará el path del directorio base de los GEDEEs, del directorio que el sistema propone como contenedor de los GEDEEs  que se utilicen por el núcleo del sistema y los demás códigos clientes del Sockeletom. Este parámetro tiene además como path raíz al directorio esquelemod, pertenecientes a la estructura de directorios del sistema Sockeletom. Tomando como ejemplo el valor ¨ gedees ¨, quedará como resultado el path:
 raíz_directorio_Sockeletom/e_modulos/gedees
###
```

path_dir_raiz: gedees

seguridad (array)(asociativo), define dentro de este bloque gedees, una serie de parámetros que aportan niveles de seguridad a la gestión y administración de GEDEEs, la gestión y administración de gedees en el sistema Sockeletom se hace por parte de una clase de procedimientos estáticos creada para este fin específico, para un mejor entendimiento de estos parámetro y su función se debe consultar la sección de ayuda sobre GEDEEs que existe en este documento, a continuación se exponen los elementos pertenecientes a esta sección:

```
seguridad:
        gestion_seguridad: true
        propietario_entidad: true
        ambito_seguridad: permisivo o restrictivo
        actualizar_datos_seguridad: false
        datos_seguridad:
                nombre_namespace1:
                        nombre_clase1
                        nombre_clase2
                        nombre_claseN
                nombre_namespaceN:
                        nombre_clase1
                        nombre_clase2
                        nombre_claseN
```

- gestion_seguridad (boolean) es quien define si se toman o no, en cuenta los parámetros de seguridad para la gestión y administración de GEDEEs.

- propietario_entidad (boolean) en true define si al ingresar una entidad GEDEE en el gestor de GEDEEs se registra el proceso que hiso el ingreso del GEDEE, esto se hace con el objetivo de que la opción de eliminar la entidad GEDEE sea controlada y solo pueda eliminarla el proceso que la creó, de lo contrario cualquier proceso puede eliminar la entidad GEDEE. Este elemento es el único que aunque este desactivado el control de seguridad (gestion_seguridad: false) funciona según su valor.

- ambito_seguridad (string) es el ámbito de seguridad por el que se desarrollará el chequeo de la lista de GEDEEs, que se admiten o deniegan su ingreso en el gestor de GEDEEs. Sus valores posibles son *permisivo* o *restrictivo*, permisivo permite todos y restringe solo los elementos de la lista, restrictivo restringe todos y permite solo los elementos de la lista.

- actualizar_datos_seguridad (boolean) define la posibilidad de que cualquier proceso pueda modificar el elemento datos_seguridad, este elemento con valor true permite que cualquier proceso

modifique la lista de procesos permitidos o denegados su gestión por parte de esta clase dedicada a los GEDEEs.

- datos_seguridad (array)(asociativo) es donde se listan las entidades GEDEEs que se permitirán o denegarán su gestión por parte de la clase GEDEEs, la estructura de este arreglo está dada por las clases y sus namespace, que serán chequeados, la gestión y administración de GEDEEs en el sistema Sockeletom se hace por parte de una clase de procedimientos estáticos creada para este fin específico, para un mejor entendimiento de estos parámetro y su función se debe consultar la sección de ayuda sobre gestión de GEDEEs que existe en este documento, a continuación un ejemplo de la estructura

```
datos_seguridad:
        namespace_gedee1:
                clase_gedee1
                clase_gedee2
                clase_gedeeN
        namespace_gedeeN:
                clase_gedee1
                clase_gedee2
                clase_gedeeN
```

namespace_gedee (string) es el nombre del namespace al que pertenece la entidad GEDEE. (obligatorio)

clase_gedee (string) es el nombre de la clase a la que pertenece la entidad GEDEE. (obligatorio)

###existentes_sistema es la declaración de un grupo (arreglo) de GEDEEs y datos necesarios para su implementación por parte del sistema, los elementos llave de este arreglo son los namespaces de los GEDEEs que el sistema Sockeletom carga desde sus inicios, ejemplo: \Emod\Nucleo\GEDEEs.
la llave namespace representa a su vez un arreglo con las clases y algunas características de estas clases que finalmente son los GEDEEs declarados, un ejemplo de estas clases es: GedeeEComun
la llave clase representa a su vez un arreglo con algunas características necesarias para la correcta gestión de los GEDEEs

Para comprender más a fondo esta sección puede remitirse a la clase Emod\Nucleo\GEDEEs. A continuación un ejemplo###

```
existentes_sistema:
        namespace_gedee1:
                clase_gedee1:
                        path_entidad_clase:
                        referencia_path_entidad:
```

```
                path_control:
                referencia_path_control:
                tipo_entidad:
                iniciacion:
                instancias:
                    id_gedee1:
                        datos:
                        parametros_iniciacion:
                    id_gedee2
                    id_gedee3:
                        parametros_iniciacion:
                    id_gedee4:
                        parametros_iniciacion:
                        datos:

namespace_gedee2:
        clase_gedee1:
                path_entidad_clase:
                referencia_path_entidad:
                path_control:
                referencia_path_control:
                tipo_entidad:
                iniciacion:
                instancias:
                    id_gedee1:
                        parametros_iniciacion:
                        datos:
                    id_gedee2:
                        datos:
                    id_gedee3
                    id_gedee4
        clase_gedee2:
                path_entidad_clase:
                referencia_path_entidad:
                path_control:
                referencia_path_control:
                tipo_entidad:
                iniciacion:
                instancias:
                    id_gedee1:
                        parametros_iniciacion:
                        datos:
                    id_gedee2:
```

```
                    datos:
            id_gedee3:
                    parametros_iniciacion:
                    datos:
            id_gedee4:
                    datos:
```

Los elementos de este arreglo se explican a continuación:

- namespace_gedee (string) es el nombre del namespace al que pertenece el GEDEE. (obligatorio)

- clase_gedee (string) es el nombre de la clase a la que pertenece el GEDEE. (obligatorio)

- path_entidad_clase (string) es el path del fichero donde se encuentra la clase del GEDEE. (obligatorio)

- referencia_path_entidad (string) es la forma que se hace referencia al path_entidad_clase, tiene dos posibles valores, *relativo*, *relativo_esquelemod*, y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor path_dir_raiz declarado en esta misma sección gedees, de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

- path_control es el path de un fichero que controlará condiciones y/o el inicio o inclusión de una entidad GEDEE, el gestor de GEDEEs espera un retorno de este control con un arreglo asociativo donde se encuentren los datos namespace_gedee, clase_gedee, id_gedee, path_entidad_clase, referencia_path_entidad, tipo_entidad, datos, o combinaciones de estos, estos datos serán procesados por la clase Emod\Nucleo\GEDEEs que es quien gestiona y administra los GEDEEs en el sistema Sockeletom. (no obligatorio)

- referencia_path_control (string) es la forma que se hace referencia al path_control, tiene dos posibles valores, *relativo*, *relativo_esquelemod*, y *absoluto*. Relativo es cuando el sistema debe tomar el path_control relativo al valor path_dir_raiz declarado en la sección gedees, de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_control relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_control de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (no obligatorio)(valor defecto *relativo*)

- tipo_entidad (string) es si la entidad GEDEE a referenciar es un objeto o una clase de procedimientos estáticos, los valores posibles son *clase* para una clase de procedimientos estáticos, y *objeto* para una instancia de la clase. (obligatorio)

- iniciacion: (string) es la forma como se iniciará una entidad GEDEE, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de $Ls_iniciacion son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de GEDEEs al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de GEDEEs le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de GEDEEs le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad útil sea de tipo clase ($Ls_tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez el primer GEDEE del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento [$Ls_namespace][$Ls_clase][$La_instancias] [$Ls_id_gedee]['parametros_iniciacion'] de la clase base, por lo que cada GEDEE puede tener sus valores propios, excepto en caso de que la entidad GEDEE sea de tipo clase ($Ls_tipo_entidad = 'clase') que con el primer id_gedee se inicia la clase base. Los elementos [$La_instancias] [$Ls_id_gedee]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

- instancias (array) es un arreglo con los identificadores de GEDEEs a reservar desde este procedimiento, este arreglo asociativo tiene como llaves los id_gedee

id_gedee (string) es un nombre o identificador de cada GEDEE instancia de la clase correspondiente. (obligatorio), y a su vez este id_gedee es un arreglo asociativo al que se puede insertar una llave de nombre parametros_iniciacion y otra de nombre datos, posteriormente este arreglo id_gedee se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_gedee como sea necesario ya que el GEDEEs puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades.

parametros_iniciacion: (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación del GEDEE (id_gedee), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base del GEDEE a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_gedee declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez este GEDEE es que se buscará el elemento 'parametros_iniciacion' en su estructura . Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

datos (array) es un contenedor de cualquier dato que se decida poner para necesidades determinadas, su estructura interna la decide quien ingrese el id_gedee al que pertenecen los datos. (no obligatorio)

- se aconseja por parte del grupo de desarrollo del núcleo Sockeletom que los namespaces se declaren de forma absoluta, comenzando con '\'

```
################################################
#  Valores referentes al Sistema Sockeletom                #
################################################

 sistema:
```

propiedades_proceso: propiedades descriptivas del sistema, pueden ser consultadas al objeto núcleo mediante un procedimiento que existe para ello, de obligatoria existencia son los siguientes:

id_proceso: identificativo para el proceso núcleo, será el identificador de nombre del proceso núcleo mientras corre el sistema Sockeletom.

namespace_gedee: es el namespace al que pertenece la entidad GEDEE conque trabaja el proceso núcleo, será el namespace de la entidad GEDEE conque trabaja el proceso núcleo mientras corre el sistema Sockeletom.

clase_gedee: es la clase a la que pertenece la entidad GEDEE conque trabaja el proceso núcleo, será la clase a la que pertenece la entidad GEDEE conque trabaja el proceso núcleo mientras corre el sistema Sockeletom.

id_gedee: es la entidad GEDEE conque trabaja el proceso núcleo, será la entidad GEDEE conque trabaja el proceso núcleo mientras corre el sistema Sockeletom.

###.

```
        #Propiedades del sistema
        propiedades_proceso:
                                version_sistema: macro 0.0.1
                                version_fich_configuracion: YAML 0.0.1
                                id_proceso: NucleoEsquelemod
                                namespace_gedee: \Emod\Nucleo\Gedees
                                clase_gedee: GedeeENucleo
```

```
                    id_gedee: GedeeENucleo
                    version: 0.0.1
                    dependencias:
                    conflictos:

### Datos de seguridad del proceso núcleo Sockeletom(forman parte de la estructura del elemento sistema) estos son los datos de seguridad que se le definen al proceso núcleo del sistema Sockeletom, estos datos son de obligatoria definición.###
        datos_seguridad:
                    permiso_ejecucion:
                                ambito: restrictivo
                                procesos:

                    acceso_seguridad:
                                ambito: restrictivo
                                procesos:

                    acceso_configuracion:
                                ambito: restrictivo
                                procesos:

                    acceso_datos:
                                ambito: restrictivo
                                procesos:
```

```
################################################
#  Valores referentes a las Herramientas del Sockeletom #
################################################

 herramientas:

### ejecucion (boolean), donde se define si se ejecutará la gestión de herramientas que ofrece el sistema Sockeletom, es decir, con su gestor de herramientas y esta configuración. Sus posibles valores son true o false, si es false se obvia la gestión de herramientas propuesta por el sistema Sockeletom y toda la sección de configuración de herramientas. Hay que tener en cuenta que dando a este parámetro el valor true puede traer consecuencias en el desarrollo de la ejecución del sistema Sockeletom y sus códigos clientes, en caso de tener valor true el sistema emitirá un E_USER_WARNING ###

        ejecucion: true
```

path_dir_raiz Este será el fragmento de path que completará el path del directorio base de las herramientas, del directorio que el sistema propone como contenedor de las herramientas que se utilicen por el núcleo del sistema y los demás códigos clientes del Sockeletom. Este parámetro tiene además como path raíz al directorio esquelemod, pertenecientes a la estructura de directorios del sistema Sockeletom. Tomando como ejemplo el valor ¨ herramientas ¨, quedará como resultado el path:
raíz_directorio_Sockeletom/e_modulos/herramientas
###

```
path_dir_raiz: herramientas
```

seguridad (array)(asociativo), define dentro de este bloque herramientas, una serie de parámetros que aportan niveles de seguridad a la gestión y administración de herramientas, la gestión y administración de herramientas en el sistema Sockeletom se hace por parte de una clase de procedimientos estáticos creada para este fin específico, para un mejor entendimiento de estos parámetro y su función se debe consultar la sección de ayuda sobre gestión de herramientas que existe en este documento, a continuación se exponen los elementos pertenecientes a esta sección:

```
seguridad:
        gestion_seguridad: true
        propietario_entidad: true
        ambito_seguridad: permisivo o restrictivo
        actualizar_datos_seguridad: false
        datos_seguridad:
                nombre_namespace1:
                        nombre_clase1
                        nombre_clase2
                        nombre_claseN
                nombre_namespaceN:
                        nombre_clase1
                        nombre_clase2
                        nombre_claseN
```

- gestion_seguridad (boolean) es quien define si se toman o no, en cuenta los parámetros de seguridad para la gestión y administración de herramientas.

- propietario_entidad (boolean) en true define si al ingresar una herramienta en el gestor de herramientas se registra el proceso que hizo el ingreso de la herramienta, esto se hace con el objetivo de que la opción de eliminar la herramienta sea controlada y solo pueda eliminarla el proceso que la creó, de lo contrario cualquier proceso puede eliminar la herramienta. Este elemento es el único que aunque este desactivado el control de seguridad (gestion_seguridad: false) funciona según su valor.

- ambito_seguridad (string) es el ámbito de seguridad por el que se desarrollará el chequeo de la lista de herramientas, que se admiten o deniegan su ingreso en el gestor de herramientas. Sus valores posibles son *permisivo* o *restrictivo*, permisivo permite todos y restringe solo los elementos de la lista, restrictivo restringe todos y permite solo los elementos de la lista.

- actualizar_datos_seguridad (boolean) define la posibilidad de que cualquier proceso pueda modificar el elemento datos_seguridad, este elemento con valor true permite que cualquier proceso modifique la lista de procesos permitidos o denegados su gestión por parte de esta clase dedicada a las herramientas.

- datos_seguridad (array)(asociativo) es donde se listan las herramientas que se permitirán o denegarán su gestión por parte de la clase herramientas, la estructura de este arreglo está dada por las clases y sus namespace, que serán chequeados, la gestión y administración de herramientas en el sistema Sockeletom se hace por parte de una clase de procedimientos estáticos creada para este fin específico, para un mejor entendimiento de estos parámetro y su función se debe consultar la sección de ayuda sobre gestión de herramientas que existe en este documento, a continuación un ejemplo de la estructura

```
datos_seguridad:
        namespace_herramientas1:
                clase_herramientas1
                clase_herramientas2
                clase_herramientasN
        namespace_herramientasN:
                clase_herramientas1
                clase_herramientas2
                clase_herramientasN
```

namespace_herramientas (string) es el nombre del namespace al que pertenece la herramienta. (obligatorio)

clase_herramientas (string) es el nombre de la clase a la que pertenece la herramienta. (obligatorio)

###existentes_sistema es la declaración de un grupo (arreglo) de herramientas y datos necesarios para su implementación por parte del sistema, los elementos llave de este arreglo son los namespaces de las herramientas que el sistema Sockeletom carga desde sus inicios, ejemplo: \Emod\Nucleo\ Herramientas.
la llave namespace representa a su vez un arreglo con las clases y algunas características de estas clases que finalmente son las herramientas declarados, un ejemplo de estas clases es: EArreglo
la llave clase representa a su vez un arreglo con algunas características necesarias para la correcta gestión de las herramientas

Para comprender más a fondo esta sección puede remitirse a la clase Emod\Nucleo\Herramientas. A continuación un ejemplo###

```
existentes_sistema:
        namespace_herramienta1:
                clase_herramienta1:
                        path_entidad_clase:
                        referencia_path_entidad:
                        path_control:
                        referencia_path_control:
                        tipo_entidad:
                        iniciacion:
                        instancias:
                                id_herramienta1:
                                        datos:
                                        parametros_iniciacion:
                                id_herramienta2
                                id_herramienta3:
                                        parametros_iniciacion:
                                id_herramienta4:
                                        parametros_iniciacion:
                                        datos:

        namespace_herramienta2:
                clase_herramienta1:
                        path_entidad_clase:
                        referencia_path_entidad:
                        path_control:
                        referencia_path_control:
                        tipo_entidad:
                        iniciacion:
                        instancias:
                                id_herramienta1:
                                        parametros_iniciacion:
                                        datos:
                                id_herramienta2:
                                        datos:
                                id_herramienta3
                                id_herramienta4
                clase_herramienta2:
                        path_entidad_clase:
                        referencia_path_entidad:
                        path_control:
```

```
                                    referencia_path_control:
                                    tipo_entidad:
                                    iniciacion:
                                    instancias:
                                        id_herramienta1:
                                            parametros_iniciacion:
                                            datos:
                                        id_herramienta2:
                                            datos:
                                        id_herramienta3:
                                            parametros_iniciacion:
                                            datos:
                                        id_herramienta4:
                                            datos:
```

los elementos de este arreglo se explican a continuación:

- namespace_herramienta (string) es el nombre del namespace al que pertenece la herramienta. (obligatorio)

- clase_herramienta (string) es el nombre de la clase a la que pertenece la herramienta. (obligatorio)

- path_entidad_clase (string) es el path del fichero donde se encuentra la clase de la herramienta. (obligatorio)

- referencia_path_entidad (string) es la forma que se hace referencia al path_entidad_clase, tiene dos posibles valores, *relativo*, *relativo_esquelemod*, y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor path_dir_raiz declarado en esta misma sección herramientas, de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

- path_control es el path de un fichero que controlará condiciones y/o el inicio o inclusión de una herramienta, el gestor de herramientas espera un retorno de este control con un arreglo asociativo donde se encuentren los datos namespace_herramienta, clase_herramienta, id_herramienta, path_entidad_clase, referencia_path_entidad, tipo_entidad, datos, o combinaciones de estos, estos datos serán procesados por esta clase Emod\Nucleo\Herramientas que es quien gestiona y administra las herramientas en el sistema Sockeletom. (no obligatorio)

- referencia_path_control (string) es la forma que se hace referencia al path_control, tiene dos posibles valores, *relativo*, *relativo_esquelemod*, y *absoluto*. Relativo es cuando el sistema debe tomar el path_control relativo al valor path_dir_raiz declarado en la sección herramientas, de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_control relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_control de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (no obligatorio)(valor defecto *relativo*)

- tipo_entidad (string) es si la entidad herramienta a referenciar es un objeto o una clase de procedimientos estáticos, los valores posibles son *clase* para una clase de procedimientos estáticos, y *objeto* para una instancia de la clase. (obligatorio)

- iniciacion: (string) es la forma como se iniciará una entidad herramienta, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de $Ls_iniciacion son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de herramientas al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de herramientas le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de herramientas le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad herramienta sea de tipo clase ($Ls_tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera herramienta del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento [$Ls_namespace][$Ls_clase][$La_instancias][$Ls_id_herramienta]['parametros_iniciacion'] de la clase base, por lo que cada herramienta puede tener sus valores propios, excepto en caso de que la entidad herramienta sea de tipo clase ($Ls_tipo_entidad = 'clase') que con el primer id_ herramienta se inicia la clase base. Los elementos [$La_instancias][$Ls_id_herramienta]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

- instancias (array) es un arreglo con los identificadores de herramientas a reservar desde este procedimiento, este arreglo asociativo tiene como llaves los id_herramienta.

id_ herramienta (string) es un nombre o identificador de cada herramienta instancia de la clase correspondiente. (obligatorio), y a su vez este id_ herramienta es un arreglo asociativo al que se puede insertar una llave de nombre parametros_iniciacion y otra de nombre datos, posteriormente este arreglo id_herramienta se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_herramienta como sea necesario ya que la herramienta puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades.

parametros_iniciacion: (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la herramienta (id_ herramienta), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la herramienta a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_ herramienta declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta herramienta es que se buscará el elemento 'parametros_iniciacion' en su estructura . Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

datos (array) es un contenedor de cualquier dato que se decida poner para necesidades determinadas, su estructura interna la decide quien ingrese la id_herramienta al que pertenecen los datos. (no obligatorio)

se aconseja por parte del grupo de desarrollo del núcleo Sockeletom que los namespaces se declaren de forma absoluta, comenzando con '\'

```
########################################
#  Valores referentes a los Utiles del Sockeletom #
########################################

 utiles:

### ejecucion (boolean), donde se define si se ejecutará la gestión de útiles que ofrece el sistema Sockeletom, es decir, con su gestor de útiles y esta configuración. Sus posibles valores son true o false, si es false se obvia la gestión de útiles propuesta por el sistema Sockeletom y toda la sección de configuración de útiles. Hay que tener en cuenta que dando a este parámetro el valor true puede traer consecuencias en el desarrollo de la ejecución del sistema Sockeletom y sus códigos clientes, en caso de tener valor true el sistema emitirá un E_USER_WARNING ###

        ejecucion: true

### path_dir_raiz Este será el fragmento de path que completará el path del directorio base de los útiles, del directorio que el sistema propone como contenedor de los útiles que se utilicen por el núcleo del sistema y los demás códigos clientes del Sockeletom. Este parámetro tiene además como path raíz al directorio esquelemod, pertenecientes a la estructura de directorios del sistema Sockeletom. Tomando como ejemplo el valor ¨ utiles¨, quedará como resultado el path:
 raíz_directorio_Sockeletom/e_modulos/utiles
###

              path_dir_raiz: utiles
```

seguridad (array)(asociativo), define dentro de este bloque útiles, una serie de parámetros que aportan niveles de seguridad a la gestión y administración de útiles, la gestión y administración de útiles en el sistema Sockeletom se hace por parte de una clase de procedimientos estáticos creada para este fin específico, para un mejor entendimiento de estos parámetro y su función se debe consultar la sección de ayuda sobre gestión de útiles que existe en este documento, a continuación se exponen los elementos pertenecientes a esta sección:

```
seguridad:
        gestion_seguridad: true
        propietario_entidad: true
        ambito_seguridad: permisivo o restrictivo
        actualizar_datos_seguridad: false
        datos_seguridad:
                nombre_namespace1:
                        nombre_clase1
                        nombre_clase2
                        nombre_claseN
                nombre_namespaceN:
                        nombre_clase1
                        nombre_clase2
                        nombre_claseN
```

- gestion_seguridad (boolean) es quien define si se toman o no, en cuenta los parámetros de seguridad para la gestión y administración de útiles.

- propietario_entidad (boolean) en true define si al ingresar un útil en el gestor de útiles se registra el proceso que hiso el ingreso del útil, esto se hace con el objetivo de que la opción de eliminar el útil sea controlada y solo pueda eliminarla el proceso que la creó, de lo contrario cualquier proceso puede eliminar el útil. Este elemento es el único que aunque este desactivado el control de seguridad (gestion_seguridad: false) funciona según su valor.

- ambito_seguridad (string) es el ámbito de seguridad por el que se desarrollará el chequeo de la lista de útiles, que se admiten o deniegan su ingreso en el gestor de útiles. Sus valores posibles son *permisivo* o *restrictivo*, permisivo permite todos y restringe solo los elementos de la lista, restrictivo restringe todos y permite solo los elementos de la lista.

- actualizar_datos_seguridad (boolean) define la posibilidad de que cualquier proceso pueda modificar el elemento datos_seguridad, este elemento con valor true permite que cualquier proceso modifique la lista de procesos permitidos o denegados su gestión por parte de esta clase dedicada a los útiles.

- datos_seguridad (array)(asociativo) es donde se listan los útiles que se permitirán o denegarán su gestión por parte de la clase útiles, la estructura de este arreglo está dada por las

clases y sus namespace, que serán chequeados, la gestión y administración de útiles en el sistema Sockeletom se hace por parte de una clase de procedimientos estáticos creada para este fin específico, para un mejor entendimiento de estos parámetro y su función se debe consultar la sección de ayuda sobre gestión de útiles que existe en este documento, a continuación un ejemplo de la estructura

```
datos_seguridad:
        namespace_utiles1:
                clase_utiles1
                clase_utiles2
                clase_utilesN
        namespace_utilesN:
                clase_utiles1
                clase_utiles2
                clase_utilesN
```

namespace_utiles (string) es el nombre del namespace al que pertenece el útil. (obligatorio)

clase_utiles (string) es el nombre de la clase a la que pertenece el útil. (obligatorio)

###existentes_sistema es la declaración de un grupo (arreglo) de útiles y datos necesarios para su implementación por parte del sistema, los elementos llave de este arreglo son los namespaces de los útiles que el sistema Sockeletom carga desde sus inicios, ejemplo: \Emod\Nucleo\Utiles.
la llave namespace representa a su vez un arreglo con las clases y algunas características de estas clases que finalmente son los útiles declarados, un ejemplo de estas clases es: EArreglo
la llave clase representa a su vez un arreglo con algunas características necesarias para la correcta gestión de los útiles

Para comprender más a fondo esta sección puede remitirse a la clase Emod\Nucleo\Utiles. A continuación un ejemplo###

```
existentes_sistema:
        namespace_util1:
                clase_util1:
                        path_entidad_clase:
                        referencia_path_entidad:
                        path_control:
                        referencia_path_control:
                        tipo_entidad:
                        iniciacion:
                        instancias:
                                id_util1:
```

```
                                        datos:
                                        parametros_iniciacion:
                                id_util2
                                id_util3:
                                        parametros_iniciacion:
                                id_util4:
                                        parametros_iniciacion:
                                        datos:

        namespace_util2:
                clase_util1:
                        path_entidad_clase:
                        referencia_path_entidad:
                        path_control:
                        referencia_path_control:
                        tipo_entidad:
                        iniciacion:
                        instancias:
                                id_util1:
                                        parametros_iniciacion:
                                        datos:
                                id_util2:
                                        datos:
                                id_util3
                                id_util4
                clase_util2:
                        path_entidad_clase:
                        referencia_path_entidad:
                        path_control:
                        referencia_path_control:
                        tipo_entidad:
                        iniciacion:
                        instancias:
                                id_util1:
                                        parametros_iniciacion:
                                        datos:
                                id_util2:
                                        datos:
                                id_util3:
                                        parametros_iniciacion:
                                        datos:
                                id_util4:
                                        datos:
```

Los elementos de este arreglo se explican a continuación:

- namespace_util (string) es el nombre del namespace al que pertenece el útil. (obligatorio)

- clase_util (string) es el nombre de la clase a la que pertenece el útil. (obligatorio)

- path_entidad_clase (string) es el path del fichero donde se encuentra la clase del útil. (obligatorio)

- referencia_path_entidad (string) es la forma que se hace referencia al path_entidad_clase, tiene dos posibles valores, *relativo*, *relativo_esquelemod*, y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor path_dir_raiz declarado en esta misma sección útiles, de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

- path_control es el path de un fichero que controlará condiciones y/o el inicio o inclusión de un útil, el gestor de útiles espera un retorno de este control con un arreglo asociativo donde se encuentren los datos namespace_utiles, clase_utiles, id_util, path_entidad_clase, referencia_path_entidad, tipo_entidad, datos, o combinaciones de estos, estos datos serán procesados por esta clase Emod\Nucleo\Utiles que es quien gestiona y administra los útiles en el sistema Sockeletom. (no obligatorio)

- referencia_path_control (string) es la forma que se hace referencia al path_control, tiene dos posibles valores, *relativo*, *relativo_esquelemod*, y *absoluto*. Relativo es cuando el sistema debe tomar el path_control relativo al valor path_dir_raiz declarado en la sección útiles, de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_control relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_control de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (no obligatorio)(valor defecto *relativo*)

- tipo_entidad (string) es si la entidad útil a referenciar es un objeto o una clase de procedimientos estáticos, los valores posibles son *clase* para una clase de procedimientos estáticos, y *objeto* para una instancia de la clase. (obligatorio)

- iniciacion: (string) es la forma como se iniciará una entidad útil, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles

valores de $Ls_iniciacion son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de útiles al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de útiles le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de útiles le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad útil sea de tipo clase ($Ls_tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez el primer útil del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento [$Ls_namespace][$Ls_clase][$La_instancias][$Ls_id_util] ['parametros_iniciacion'] de la clase base, por lo que cada útil puede tener sus valores propios, excepto en caso de que la entidad útil sea de tipo clase ($Ls_tipo_entidad = 'clase') que con el primer id_util se inicia la clase base. Los elementos [$La_instancias][$Ls_id_util] ['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

- instancias (array) es un arreglo con los identificadores de útiles a reservar desde este procedimiento, este arreglo asociativo tiene como llaves los id_util

id_util (string) es un nombre o identificador de cada útil instancia de la clase correspondiente. (obligatorio), y a su vez este id_util es un arreglo asociativo al que se puede insertar una llave de nombre parametros_iniciacion y otra de nombre datos, posteriormente este arreglo id_util se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_util como sea necesario ya que el útil puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades.

parametros_iniciacion: (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación del útil (id_util), si existe este parámetro, es arreglo y no es vacio entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base del útil a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_util declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez este útil es que se buscará el elemento 'parametros_iniciacion' en su estructura . Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

datos (array) es un contenedor de cualquier dato que se decida poner para necesidades determinadas, su estructura interna la decide quien ingrese el id_util al que pertenecen los datos. (no obligatorio)

se aconseja por parte del grupo de desarrollo del núcleo Sockeletom que los namespaces se declaren de forma absoluta, comenzando con '\'

```
#########################################################
#  Valores referentes a Tratamiento de Errores en el Sockeletom #
#########################################################
```
Primero mostramos la estructura completa y luego se explicará cada elemento de esta estructura.

```
errores:
        ejecucion: true
        path_dir_raiz: errores
        seguridad:
                gestion_seguridad: true
                propietario_entidad: true
                ambito_seguridad: permisivo
                actualizar_datos_seguridad: false
                datos_seguridad:
                        nombre_namespace1:
                                                - nombre_clase1
                                                - nombre_clase2
                                                - nombre_claseN
                        nombre_namespaceN:
                                                - nombre_clase1
                                                - nombre_clase2
                                                - nombre_claseN

        existentes_sistema:
                        namespace_errores1:
                                clase_errores1:
                                        path_entidad_clase:
                                        referencia_path_entidad:
                                        path_control:
                                        referencia_path_control:
                                        tipo_entidad:
                                        iniciacion:
                                        instancias:
                                                id_errores1:
                                                        datos:
                                                        parametros_iniciacion:
                                                id_errores2
                                                id_errores3:
                                                        parametros_iniciacion:
                                                id_errores4:
                                                        parametros_iniciacion:
                                                        datos:
```

```
            namespace_errores2:
                clase_errores1:
                    path_entidad_clase:
                    referencia_path_entidad:
                    path_control:
                    referencia_path_control:
                    tipo_entidad:
                    iniciacion:
                    instancias:
                        id_errores1:
                            parametros_iniciacion:
                            datos:
                        id_errores2:
                            datos:
                        id_errores3
                        id_errores4
                clase_errores2:
                    path_entidad_clase:
                    referencia_path_entidad:
                    path_control:
                    referencia_path_control:
                    tipo_entidad:
                    iniciacion:
                    instancias:
                        id_errores1:
                            parametros_iniciacion:
                            datos:
                        id_errores2:
                            datos:
                        id_errores3:
                            parametros_iniciacion:
                            datos:
                        id_errores4:
                            datos:

entidad_errores_emod:
            \Emod\Nucleo\Errores:
                EErrores:
                    path_entidad_clase: e_modulos/nucleo/nucleo_bib_funciones/class_eerrores.php
                    referencia_path_entidad: relativo_esquelemod
                    tipo_entidad: objeto
                    iniciacion: construct
```

```
                    instancias:
                                EEoErrores:
                                    parametros_iniciacion:
                                        - localis_['errores']['entidad_gedege_emod']
                                        - localis_['errores']['funcion_gestor_errores']
                                        - localis_['errores']['elementos_error']
                                        - localis_['errores']['formato_mensaje']
                                        - localis_['errores']['fuente_datos_error']
                                        - localis_['errores']['formato_filtrado']
                                        - localis_['errores']['tipos_error']
                                    datos:
funcion_gestor_errores:
        identificador: gestorErroresEmod
        nombre_funcion: gestorErroresEmod
entidad_gedege_emod:
        Emod\Nucleo\Errores\Gedege:
            GedegeTxtEmod:
                path_entidad_clase: gedeges/gedege_txt_emod/gedege_bib_funciones/class_gedege_txt_emod.php
                referencia_path_entidad: relativo
                tipo_entidad: objeto
                instancias:
                        GedegeTxtEmod:
                                datos:
elementos_error:
            miembros_errorlog:
                        - tiempo
                        - fichero
                        - linea
                        - proceso
                        - gedee_proceso
            miembros_error:
                        - id
                        - tipo
                        - mensaje
fuente_datos_error:
             path_fich_error: error.txt
             referencia_path_error: relativo
             path_fich_errorlog: errorlog.txt
             referencia_path_errorlog: relativo
formato_filtrado:
             formato:
                    separador: "::"
             filtrado:
```

```
                    marca_ausente: true
    formato_mensaje:
            formato_mensaje_error:
                    - id
                    - tipo
                    - mensaje
            formato_mensaje_errorlog:
                    - id
                    - tipo
                    - mensaje
                    - tiempo
                    - fichero
                    - linea
                    - proceso
                    - gedee_proceso
    tipos_error: E_ALL
```

A continuación explicamos cada elemento de esta sección errores:

errores (arreglo), contendrá toda la estructura de configuración referente a la gestión de errores en el sistema Sockeletom.

errores:

ejecucion (boolean), donde se define si se ejecutará la gestión de errores que ofrece el sistema Sockeletom, es decir, con su gestor de errores y esta configuración. Sus posibles valores son true o false, si es false se obvia la gestión de errores propuesta por el sistema Sockeletom y toda la sección de configuración de errores. Hay que tener en cuenta que dando a este parámetro el valor true puede traer consecuencias en el desarrollo de la ejecución del sistema Sockeletom y sus códigos clientes, en caso de tener valor true el sistema emitirá un E_USER_WARNING

```
    ejecucion: true
```

path_dir_raiz Este será el fragmento de path que completará el path del directorio base de los errores, del directorio que el sistema propone como contenedor de los recursos errores que se utilicen por el núcleo del sistema y los demás códigos clientes del Sockeletom. Este parámetro tiene además como path raíz al directorio esquelemod, pertenecientes a la estructura de directorios del sistema Sockeletom. Tomando como ejemplo el valor ¨ errores¨, quedará como resultado el path: raíz_directorio_Sockeletom/e_modulos/errores
###

```
    path_dir_raiz: errores
```

seguridad (array)(asociativo), define dentro de este bloque errores, una serie de parámetros que aportan niveles de seguridad a la gestión y administración de errores, la gestión y administración de errores en el sistema Sockeletom se hace por parte de una clase de procedimientos estáticos creada para este fin específico, para un mejor entendimiento de estos parámetro y su función se debe consultar la sección de ayuda sobre gestión de errores que existe en este documento, a continuación se exponen los elementos pertenecientes a esta sección:

- gestion_seguridad (boolean) es quien define si se toman o no, en cuenta los parámetros de seguridad para la gestión y administración de errores.

- propietario_entidad (boolean) en true define si al ingresar una entidad errores en el gestor de entidades errores se registra el proceso que hizo el ingreso de la entidad errores, esto se hace con el objetivo de que la opción de eliminar la entidad errores sea controlada y solo pueda eliminarla el proceso que la creó, de lo contrario cualquier proceso puede eliminar la entidad errores. Este elemento es el único que aunque este desactivado el control de seguridad (gestion_seguridad: false) funciona según su valor.

- ambito_seguridad (string) es el ámbito de seguridad por el que se desarrollará el chequeo de la lista de entidades errores, que se admiten o deniegan su ingreso en el gestor de entidades errores. Sus valores posibles son *permisivo* o *restrictivo*, permisivo permite todos y restringe solo los elementos de la lista, restrictivo restringe todos y permite solo los elementos de la lista.

- actualizar_datos_seguridad (boolean) define la posibilidad de que cualquier proceso pueda modificar el elemento datos_seguridad, este elemento con valor true permite que cualquier proceso modifique la lista de procesos permitidos o denegados su gestión por parte de esta clase dedicada a las entidades errores.

- datos_seguridad (array)(asociativo) es donde se listan las entidades errores que se permitirán o denegarán su gestión por parte de la clase Errores, la estructura de este arreglo está dada por las clases y sus namespace, que serán chequeados, la gestión y administración de entidades errores en el sistema Sockeletom se hace por parte de una clase de procedimientos estáticos creada para este fin específico, para un mejor entendimiento de estos parámetro y su función se debe consultar la sección de ayuda sobre gestión de errores que existe en este documento, a continuación un ejemplo de la estructura.

nombre_namespace (string) es el nombre del namespace al que pertenece la entidad errores. (obligatorio)

nombre_clase (string) es el nombre de la clase a la que pertenece la entidad errores. (obligatorio)

```
    seguridad:
                    gestion_seguridad: true
                    propietario_entidad: true
```

```
ambito_seguridad: permisivo o restrictivo
actualizar_datos_seguridad: false
datos_seguridad:
        nombre_namespace1:
                nombre_clase1
                nombre_clase2
                nombre_claseN
        nombre_namespaceN:
                nombre_clase1
                nombre_clase2
                nombre_claseN
```

###existentes_sistema (arreglo), es la declaración de un grupo de entidades errores y datos necesarios para su implementación por parte del sistema, los elementos llave de este arreglo son los namespaces de las entidades errores que el sistema Sockeletom carga desde sus inicios, ejemplo: \Emod\Nucleo\Errores.
la llave namespace representa a su vez un arreglo con las clases y algunas características de estas clases que finalmente son las entidades errores declaradas, un ejemplo de estas clases es: EErrores
la llave clase representa a su vez un arreglo con algunas características necesarias para la correcta gestión de las entidades errores.

Para comprender más a fondo esta sección puede remitirse a la clase Emod\Nucleo\Errores. A continuación un ejemplo###

```
existentes_sistema:
        namespace_errores 1:
                clase_errores1:
                        path_entidad_clase:
                        referencia_path_entidad:
                        path_control:
                        referencia_path_control:
                        tipo_entidad:
                        iniciacion:
                        instancias:
                                id_errores1:
                                        datos:
                                        parametros_iniciacion:
                                id_errores2
                                id_errores3:
                                        parametros_iniciacion:
                                id_errores4:
                                        parametros_iniciacion:
                                        datos:
```

```
namespace_errores2:
        clase_errores 1:
                path_entidad_clase:
                referencia_path_entidad:
                path_control:
                referencia_path_control:
                tipo_entidad:
                iniciacion:
                instancias:
                        id_errores 1:
                                parametros_iniciacion:
                                datos:
                        id_errores2:
                                datos:
                        id_errores3
                        id_errores4
        clase_errores2:
                path_entidad_clase:
                referencia_path_entidad:
                path_control:
                referencia_path_control:
                tipo_entidad:
                iniciacion:
                instancias:
                        id_errores1:
                                parametros_iniciacion:
                                datos:
                        id_errores2:
                                datos:
                        id_errores3:
                                parametros_iniciacion:
                                datos:
                        id_errores4:
                                datos:
```

Los elementos de este arreglo se explican a continuación:

- namespace_errores: (string) es el nombre del namespace al que pertenece la entidad errores.

- clase_errores: (string) es el nombre de la clase a la que pertenece la entidad errores.

- ' path_entidad_clase ': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de la clase Errores, si el fichero no existe no se ingresa la entidad errores al arreglo $laEntidades.

- ' referencia_path_entidad ': (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor de la propiedad path_dir_raiz declarado en esta sección de configuración Errores. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod (). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

- ' path_control ': (string) es el path del fichero control a ejecutar para el ingreso de la entidad errores. (obligatorio)

- ' referencia_path_control ': (string) es la forma que se hace referencia a path_control, tiene tres posibles valores, *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_control relativo al valor de la propiedad path_dir_raiz declarado en esta sección de configuración Errores. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_control relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_control de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

- ' tipo_entidad ': (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

- ' iniciación ': (string) es la forma como se iniciará cada id_errores, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades errores al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades errores le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades errores le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad errores sea de tipo clase ($Ls_tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad errores del arreglo instancias de la clase base que se gestione, ya que con este está iniciada

la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento [' instancias '] [id_errores]['parametros_iniciacion'] de la clase base, por lo que cada entidad errores puede tener sus valores propios, excepto en caso de que la entidad errores sea de tipo clase (tipo_entidad = 'clase') que con el primer id_errores se inicia la clase base. Los elementos ['instancias'][id_errores] ['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

- ' instancias ' (array) es un arreglo con los identificadores de entidades errores a reservar desde el procedimiento correspondiente, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_errores. (obligatorio)

id_errores (string) es un nombre o identificador de cada entidad errores instancia de la clase correspondiente. y a su vez este id_errores es un arreglo asociativo al que se puede insertar una llave de nombre ' parametros_iniciacion ' y otra de nombre ' datos ', posteriormente este arreglo id_errores se le incorporan otros elementos con llave asociativa para el trabajo de la clase Errores. Pueden existir tantos id_errores como sea necesario ya que la entidad errores puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad errores (id_errores), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento correspondiente pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad errores a gestionar sea de tipo clase (tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_errores declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad errores es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad errores en la clase Errores; estos datos se crean en el momento del ingreso de la entidad errores sin que permita cambios posteriores. (opcional)

entidad_errores_emod: (array) es la estructura de datos para gestionar una entidad de tipo objeto que será la encargada de la gestión de errores en el sistema Sockeletom, es una propuesta de gestor de errores ya que es opcional su utilización, aclaramos que el Sockeletom tiene un gestor de entidades errores que no es lo mismo que esta entidad_errores_emod ya que el gestor de entidades errores gestiona y contiene entidades para la gestión de errores y esta entidad_errores_emod es quien gestiona los errores específicamente es decir la estructura de mensajes de error con su tipo y la fuente donde se encuentra esta estructura, como también se

encarga del destino y estructura del log de errores, es la entidad que se implantará desde set_error_handler y se encargará de lanzar los trigger_error con sus parámetros ya elaborados.

Este array tiene la misma estructura que la de una entidad a ingresar en la clase Errores, ya que internamente el sistema la incluye dentro de esta clase.

A continuación un ejemplo de la estructura y la explicación de cada elemento de esta.

```
entidad_errores_emod:
        namespace_eemod:
                clase_eemod:
                        path_entidad_clase:
                        referencia_path_entidad:
                        path_control:
                        referencia_path_control:
                        tipo_entidad:
                        iniciacion:
                        instancias:
                                id_eemod:
                                        parametros_iniciacion:
                                            - localis_['errores']['entidad_gedege_emod']
                                            - localis_['errores']['funcion_gestor_errores']
                                            - localis_['errores']['elementos_error']
                                            - localis_['errores']['formato_mensaje']
                                            - localis_['errores']['fuente_datos_error']
                                            - localis_['errores']['formato_filtrado']
                                            - localis_['errores']['tipos_error']
                                        datos:
```

- namespace_eemod: (string) es el nombre del namespace al que pertenece la entidad errores.

- clase_eemod: (string) es el nombre de la clase a la que pertenece la entidad errores.

- ' path_entidad_clase ': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de la clase Errores, si el fichero no existe no se ingresa la entidad errores al arreglo $laEntidades.

- ' referencia_path_entidad ': (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor de la propiedad path_dir_raiz declarado en esta sección de configuración Errores. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo-

>pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

- ' path_control ': (string) es el path del fichero control a ejecutar para el ingreso de la entidad errores. (obligatorio)

- ' referencia_path_control ': (string) es la forma que se hace referencia a path_control, tiene tres posibles valores, *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_control relativo al valor de la propiedad path_dir_raiz declarado en esta sección de configuración Errores. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_control relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_control de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

- ' tipo_entidad ': (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

- ' iniciación ': (string) es la forma como se iniciará cada id_errores, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades errores al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades errores le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades errores le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad errores sea de tipo clase ($Ls_tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad errores del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento [' instancias '] [id_errores]['parametros_iniciacion'] de la clase base, por lo que cada entidad errores puede tener sus valores propios, excepto en caso de que la entidad errores sea de tipo clase (tipo_entidad = 'clase') que con el primer id_errores se inicia la clase base. Los elementos ['instancias'][id_errores] ['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

- ' instancias ' (array) es un arreglo con los identificadores de entidades errores a reservar desde el procedimiento correspondiente, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_errores. (obligatorio)

id_eemod (string) es un nombre o identificador de cada entidad errores instancia de la clase correspondiente. y a su vez este id_errores es un arreglo asociativo al que se puede insertar una llave de nombre ' parametros_iniciacion ' y otra de nombre ' datos ', posteriormente este arreglo id_errores se le incorporan otros elementos con llave asociativa para el trabajo de la clase Errores. Pueden existir tantos id_errores como sea necesario ya que la entidad errores puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad errores (id_errores), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento correspondiente pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad errores a gestionar sea de tipo clase (tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_errores declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad errores es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional). En esta estructura de parámetros de iniciación vemos algo diferente, se debe a la utilización de
- localis_['errores']['entidad_gedege_emod'],
- localis_['errores']['funcion_gestor_errores'],
- localis_['errores']['elementos_error'],
- localis_['errores']['formato_mensaje'],
- localis_['errores']['fuente_datos_error']
- localis_['errores']['formato_filtrado'],
- localis_['errores']['tipos_error']

El uso de localis_***está descrito en la sección identificador_referencia_local de esta configuración, es la primera sección de esta configuración, los elementos que nos interesan son ['entidad_gedege_emod'], ['funcion_gestor_errores'], ['elementos_error'], ['formato_mensaje'], ['fuente_datos_error'], ['formato_filtrado'] y ['tipos_error'], estos elementos son los parámetros con los que se iniciará o construirá la instancia de esta entidad id_eemod, pero por ser estos elementos a su vez arreglos con una importancia relevante, los pasamos a describir en otra ubicación del arreglo errores, con el fin de hacerlos mas legibles.

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad errores en la clase Errores; estos datos se crean en el momento del ingreso de la entidad errores sin que permita cambios posteriores. (opcional)

funcion_gestor_errores: (array), contiene los valores referentes a la función que se cargara con set_error_handler como gestor de errores por defecto del sistema Sockeletom. Este es uno de los

elementos a los que se hace referencia con localis_['errores'] en 'parametros_iniciacion' del id_eemod de entidad_errores_emod. Su estructura es la siguiente:

```
funcion_gestor_errores:
                identificador:
                nombre_funcion:
```

Consta de dos elementos que explicamos a continuación:

identificador: string, valor identificador de la función gestor de errores que se cargara con set_error_handler como gestor de errores por defecto del sistema Sockeletom, este identificador es obligatorio su existencia.
Esta propiedad es específicas de cada instancia, por eso se deben introducir en la iniciación de la instancia.

nombre_funcion: string, valor nombre de la función gestor de errores que se cargara con set_error_handler como gestor de errores por defecto del sistema Sockeletom, este nombre es obligatorio su existencia. Es dentro del objeto instancia de la entidad_errores_emod el procedimiento escogido para implantarse como gestor de errores, con la función del lenguaje set_error_handler.
Esta propiedad es específicas de cada instancia, por eso se deben introducir en la iniciación de la instancia.

###entidad_gedege_emod: (array) es la estructura de datos para gestionar una entidad de tipo objeto que será la encargada de gestionar estructuras de datos que se utilizarán en la gestión de errores en el sistema Sockeletom, es una propuesta de, gestor de estructuras de datos para entidades gestoras de errores(GEDEGE), ya que es opcional su utilización, aclaramos que el GEDEGE es la interfaz entre entidad_errores_emod y las fuentes de datos donde se encuentran los datos correspondientes a errores y log de errores, es el GEDEGE quien sabe cómo gestionar(leer, escribir, eliminar) estos datos en la fuente. Independizando la gestión de la entidad_errores_emod de la estructura de datos error-errorlog y la fuente donde se encuentran estos, puede existir una entidad_errores_emod que pueda interactuar con estructuras de datos diferentes(txt, bd, xml, yaml, etc.)Porque para cada una de estas estructuras o fuentes de datos existe un GEDEGE.

Este array tiene la misma estructura que la de una entidad a ingresar en la clase Errores, ya que internamente el sistema la incluye dentro de esta clase.

Este es uno de los elementos a los que se hace referencia con localis_['errores'] en 'parametros_iniciacion' del id_eemod de entidad_errores_emod. A continuación un ejemplo de la estructura y la explicación de cada elemento de esta.

```
entidad_gedege_emod:
        namespace_gedege:
```

```
                    clase_gedege:
                        path_entidad_clase:
                        referencia_path_entidad:
                        path_control:
                        referencia_path_control:
                        tipo_entidad:
                        iniciacion:
                        instancias:
                            id_gedege:
                                parametros_iniciacion:
                                datos:
```

- namespace_gedege: (string) es el nombre del namespace al que pertenece la entidad errores.

- clase_gedege: (string) es el nombre de la clase a la que pertenece la entidad errores.

- ' path_entidad_clase ': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de la clase Errores, si el fichero no existe no se ingresa la entidad errores al arreglo $laEntidades.

- ' referencia_path_entidad ': (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor de la propiedad path_dir_raiz declarado en esta sección de configuración Errores. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

- ' path_control ': (string) es el path del fichero control a ejecutar para el ingreso de la entidad errores. (obligatorio)

- ' referencia_path_control ': (string) es la forma que se hace referencia a path_control, tiene tres posibles valores, *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_control relativo al valor de la propiedad path_dir_raiz declarado en esta sección de configuración Errores. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_control relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_control de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

- ' tipo_entidad ': (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

- ' iniciación ': (string) es la forma como se iniciará cada id_errores, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades errores al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades errores le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades errores le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad errores sea de tipo clase ($Ls_tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad errores del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento [' instancias '] [id_errores]['parametros_iniciacion'] de la clase base, por lo que cada entidad errores puede tener sus valores propios, excepto en caso de que la entidad errores sea de tipo clase (tipo_entidad = 'clase') que con el primer id_errores se inicia la clase base. Los elementos ['instancias'][id_errores] ['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

- ' instancias ' (array) es un arreglo con los identificadores de entidades errores a reservar desde el procedimiento correspondiente, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_errores. (obligatorio)

id_gedege (string) es un nombre o identificador de cada entidad errores instancia de la clase correspondiente. y a su vez este id_errores es un arreglo asociativo al que se puede insertar una llave de nombre ' parametros_iniciacion ' y otra de nombre ' datos ', posteriormente este arreglo id_errores se le incorporan otros elementos con llave asociativa para el trabajo de la clase Errores. Pueden existir tantos id_errores como sea necesario ya que la entidad errores puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad errores (id_errores), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento correspondiente pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad errores a gestionar sea de tipo clase (tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_errores declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad errores es que se

buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional).

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad errores en la clase Errores; estos datos se crean en el momento del ingreso de la entidad errores sin que permita cambios posteriores. (opcional)

elementos_error: (array), donde se definen los elementos que serán manejados por la entidad_errores_emod para el tratamiento de errores, un ejemplo de elementos puede ser identificador del error, tipo del error, mensaje a mostrar etc. Estos elementos servirán como mascara, patrón para comparar y filtrar, o asociar valores de datos gestionados en los procedimientos de la entidad_errores_emod, funcionando como campos o variables con que regir datos que se gestionen en la entidad_errores_emod. Este arreglo consta de dos llaves asociativas que explicamos a continuación:

'miembros_error' define los campos o elementos que se gestionaran para un mensaje u otro destino de errores un ejemplo puede ser array('id' , 'tipo' , 'tratamiento' , 'alcance' , 'mensaje').

'miembros_errorlog' define los campos o elementos que se gestionaran para un contenedor de registros de errores, a esta propiedad siempre se le adicionan los elementos de la propiedad 'miembros_error', un ejemplo puede ser array('date' , 'aplicacion' , 'modulo' , 'opcion')+ ('id' , 'tipo' , 'tratamiento' , 'alcance' , 'mensaje').

Esta propiedad es específicas de cada instancia, por eso se deben introducir en la iniciación de la instancia.

Este es uno de los elementos a los que se hace referencia con localis_['errores'] en 'parametros_iniciacion' del id_eemod de entidad_errores_emod.

A continuación un ejemplo de esta estructura.

```
elementos_error:
        miembros_errorlog:
                - tiempo
                - fichero
                - linea
                - proceso
                - gedee_proceso
        miembros_error:
                - id
                - tipo
                - mensaje
```

fuente_datos_error(array), donde se definen los datos para el acceso a la fuente de datos de los errores, estos datos hacen referencia a la fuente donde se gestionaran los datos error y los datos errorlog, la estructura interna de este arreglo depende de la forma que los deba leer el gedege asignado a la instancia de la entidad_errores_emod definida, ya que fuente_datos_error será pasado como argumento a los procedimientos del gedege en su gestión.

Este es uno de los elementos a los que se hace referencia con localis_['errores'] en 'parametros_iniciacion' del id_eemod de entidad_errores_emod.

un ejemplo de esta estructura puede ser:

fuente_datos_error ['path_fich_errorlog'] (esto es en caso de ser un fichero)
fuente_datos_error [referencia_path_errorlog ']
fuente_datos_error ['path_fich_error'](string)
fuente_datos_error [referencia_path_error ']

fuente_datos_error ['servidor_bd'](string)(esto es en caso de ser una Bd)
fuente_datos_error ['usuario_bd'](string)
fuente_datos_error ['password_bd'](string)

Esta propiedad es específicas de cada instancia, por eso se deben introducir en la iniciación de la instancia.

formato_filtrado: (array), donde se definen los datos para el filtrado de paquetes de datos en términos de el funcionamiento del filtrado y la estructura de los paquetes de datos de errores, estos datos hacen referencia a elementos necesarios en el filtrado y/o formato de los datos en la fuente donde estos se encuentren, éstos datos hacen referencia a los datos error y los datos errorlog, la estructura interna de este arreglo depende de la forma que los deba leer el gedege asignado a la entidad_errores_emod definida, ya que formato_filtrado será pasado como argumento a los procedimientos del gedege en su gestión; un ejemplo puede ser:

formato_filtrado ['formato']['separador'](string) es el separador de datos en una línea de texto en un txt
formato_filtrado ['filtrado']['marca_ausente'](boolean) si un campo de datos tiene valor vacío, marca el campo con el propio nombre del campo entre paréntesis como valor del campo.
formato_filtrado ['filtrado']['llave_unica'](string) define una llave como condición para que cuando esta llave y su valor sean encontrados en la línea del txt, se detenga la búsqueda por las líneas restantes del fichero, es decir se detenga el proceso de filtrado y se pase a al termino de la gestión de un procedimiento determinado.

Esta propiedad es específicas de cada instancia, por eso se deben introducir en la iniciación de la instancia.

formato_mensaje: (array), es donde se define la estructura del formato del mensaje error a emitir o el mensaje errorlog a guardar, este arreglo consta de dos llaves o índices asociativos que se explican continuación:
formato_mensaje_error: array, donde se definen los elementos que conformarán el mensaje de error, es el grupo de elementos que se concatenaran para formar el mensaje de error emitido por el procedimiento correspondiente, estos elementos conformarán el mensaje en el mismo orden en que aparecen los elementos, cada elemento de este arreglo será tenido en cuenta como llave del arreglo asociativo fuente para los datos que conformarán el mensaje de error. Un ejemplo puede ser array('id' , 'tipo' , 'mensaje')

formato_mensaje_errorlog: array, donde se definen los elementos que conformarán el contenido del registro de errorlog, es el grupo de elementos que se concatenaran para formar el contenido del registro de errorlog emitido por el procedimiento correspondiente, estos elementos conformarán el registro en el mismo orden en que aparecen los elementos, cada elemento de este arreglo será tenido en cuenta como llave del arreglo asociativo fuente para los datos que conformarán el registro de errorlog. Un ejemplo puede ser array('id' , 'tipo' , 'mensaje' , 'tipo' , 'tiempo' , 'fichero' , 'linea', 'proceso').

Aclaramos que esta propiedad es independiente de formato_mensaje_error.

Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.
Un ejemplo de esta estructura puede ser la siguiente:

```
formato_mensaje:
        formato_mensaje_error:
                - id
                - tipo
                - mensaje
        formato_mensaje_errorlog:
                - id
                - tipo
                - mensaje
                - tiempo
                - fichero
                - linea
                - proceso
                - gedee_proceso
```

tipos_error: (string), valor referente a los tipos de errores E_USER_ERROR , E_USER_WARNING , y E_USER_NOTICE establecidos en los niveles de errores de PHP, este valor se pasa como parámetro de la función set_error_handler() para la instauración de un

manejador de errores, aportado por un script usuario del manejo de errores del Sockeletom. Para un mejor entendimiento ver la ayuda de la función set_error_handler() del lenguaje php.

Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

Un ejemplo de esta estructura puede ser:

tipos_error: E_ALL

```
#######################################
#  Valores referentes a Procesos                #
#######################################
# Bloques de procesos a cargar por el sistema
 procesos:
```

Raíz del directorio de procesos, es el path donde radicarán todos los procesos a ejecutarse en el sistema Sockeletom, este valor se anexa al path raíz_sistema_Sockeletom/esquelemod/e_modulos, quedando como resultado el path raíz_sistema_Sockeletom/esquelemod/e_modulos/procesos en el caso del valor ¨procesos¨ que por defecto trae el Sockeletom, estos datos son de obligatoria definición.###

```
                        path_raiz_procesos: procesos
```

###Orden y permiso de ejecución de bloques de procesos, mas adelante se explicará que los procesos a ejecutar se definen en bloques o listas de estos, y es en este elemento donde se definen el orden en que se ejecutarán estos bloques o listas de procesos, además de que si se omite o comenta el identificador de un bloque este no se ejecutará, estos datos son de obligatoria definición###

```
                        orden_permiso_ejecucion_bloques:
                                                -bloque_defecto
                                                #-bloque_fundator
```

###arbol_procesos es donde se definen elementos relacionado con el árbol de procesos, A continuación un ejemplo:

```
arbol_procesos:
                            limite_idorejec_global: -1
                            limite_idorejec_bloqueprocesos: -1
                            limite_idorejec_subproceso: -1
                            limite_idorejec_recursivoproceso: -1
                            limite_idorejec_recursivoproimpejec: -1
```

Esta estructura (arreglo) se constituye de los elementos que se explican a continuación:

limite_idorejec_global: (integer) límite para el número entero identificativo de orden de ejecución a escala global, es decir un número límite para el orden consecutivo que cada proceso ocupa en la ejecución de procesos durante la ejecución del sistema Sockeletom de forma global. El valor -1 significa que el límite es infinito, los restantes posibles valores son desde 0 hasta infinito positivo.

limite_idorejec_bloqueprocesos: (integer) límite para el número entero identificativo de orden de ejecución en el bloque de procesos, es decir un número límite para el orden consecutivo que cada proceso ocupa en el bloque declarado en la configuración del sistema. El valor -1 significa que el límite es infinito, los restantes posibles valores son desde 0 hasta infinito positivo.

limite_idorejec_subproceso: (integer) límite para el número entero identificativo de orden de ejecución de un proceso como subproceso de otro, es decir un número límite para el orden consecutivo que cada proceso ocupa en la ejecución de procesos como subproceso del proceso padre. El valor -1 significa que el límite es infinito, los restantes posibles valores son desde 0 hasta infinito positivo.

limite_idorejec_recursivoproceso: (integer) límite para el número entero identificativo de orden de ejecución de un proceso de forma recursiva, es decir un número límite para el orden consecutivo que cada proceso ocupa cuando es ejecutado por él mismo, de forma recursiva. El valor -1 significa que el límite es infinito, los restantes posibles valores son desde 0 hasta infinito positivo.

limite_idorejec_recursivoproimpejec: (integer) límite para el número entero identificativo de orden de ejecución de procesos cuando el procedimiento que implementa la ejecución de los procesos es llamado recursivamente, es decir un número límite de veces que es llamado recursivamente el procedimiento de ejecución, no el proceso, es cuando un proceso llama dentro de sí el procedimiento de ejecución para ejecutarse a sí mismo o a otro proceso, se refiere a la llamada del procedimiento que implementa la ejecución de los procesos no desde el núcleo del sistema sino desde un proceso ya ejecutándose. El valor -1 significa que el límite es infinito, los restantes posibles valores son desde 0 hasta infinito positivo.

###seguridad_procesos es donde se definen los permisos de ejecución de los procesos, permisos para procesos que se listan en los bloques de procesos de esta configuración y también de los procesos que se ejecutan desde otro proceso, todos están sujetos a esta estructura de datos de seguridad.
A continuación un ejemplo:

```
seguridad_procesos:
                        permision_nucleo:
                                ambito_seguridad: permisivo
                                procesos:
                                        - proceso1
                                        - procesoN
```

```
permision_clientes:
        ambito_seguridad: permisivo
        procesos:
            \namespace1:
                clase1:
                    - proceso1
                    - procesoN
                clase2:
                    - proceso1
                    - procesoN
            \namespace2:
                clase1:
                    - proceso1
                    - procesoN
                clase2:
                    - proceso1
                    - procesoN
```

Esta estructura (arreglo) se constituye de dos índices o llaves asociativas que se explican a continuación:

permision_nucleo: es donde se definen los procesos de tipo núcleo que pueden ejecutarse, esta llave tiene como estructuras las llaves o índices asociativos ‘ambito_seguridad‘ y ‘procesos‘con las siguientes estructuras:

ambito_seguridad (string) es el ámbito de seguridad por el que se desarrollará el chequeo de la lista de procesos, que se permite su ejecución por parte del sistema. Sus valores posibles son *permisivo* o *restrictivo*, permisivo permite todos y restringe solo los elementos de la lista, restrictivo restringe todos y permite solo los elementos de la lista.

procesos (array) no asociativo donde cada elemento debe ser el identificador del proceso (id_proceso) a controlar su ejecución por seguridad, en este caso no se declaran el namespace y clase del GEDEE del proceso a controlar por seguridad, porque el sistema Sockeletom considera que los procesos que pueden ejecutarse como núcleo tienen que tener el mismo namespace y clase de GEDEE que el proceso que arranca el sistema como núcleo.

permision_clientes: es donde se definen los procesos de tipo núcleo que pueden ejecutarse, esta llave tiene como estructuras las llaves o índices asociativos ‘ambito_seguridad‘ y ‘procesos‘con las siguientes estructuras:

ambito_seguridad (string) es el ámbito de seguridad por el que se desarrollará el chequeo de la lista de procesos, que se permite su ejecución por parte del sistema. Sus valores posibles son

permisivo o *restrictivo*, permisivo permite todos y restringe solo los elementos de la lista, restrictivo restringe todos y permite solo los elementos de la lista.

procesos (array) asociativo donde cada elemento debe ser el identificador del namespace del GEDEE del proceso (id_proceso) a ejecutar, y este identificador del namespace es un arreglo asociativo donde cada elemento debe ser el identificador de la clase del GEDEE proceso (id_proceso) a ejecutar, esta clase es un arreglo no asociativo donde cada elemento debe ser el identificador del proceso (id_proceso) a controlar su ejecución por seguridad.

bloques_procesos es un arreglo asociativo donde sus llaves son los identificadores de los bloques de procesos a cargar por el sistema, aquí se declaran bloques de procesos a ejecutar por el sistema:

bloques_procesos

estos bloques son estructuras de arreglos asociativos que están compuestos por el id de proceso a ejecutarse en este bloque, y cada proceso tiene los siguientes parámetros

gedee_proceso (array asociativo) con parámetros referentes al GEDEE del proceso,entre ellos están:

- namespace del GEDEE que utiliza este proceso.
- clase del GEDEE que utiliza este proceso.
- Id_entidad del GEDEE que utiliza este proceso.

propiedades_implementacion_proceso (array asociativo) que a su vez está compuesto por tres parámetros obligatorios y uno opcional que son:

- Primero está el parámetro path_raiz que define el path del directorio raíz para los recursos del proceso, este path se toma a partir del elemento path_raiz_procesos definido en esta sección bloques_procesos, quedando de la siguiente forma

  raíz_sistema_Sockeletom/esquelemod/e_modulos/path_raiz_procesos/path_raiz

- segundo está el parámetro path_arranque que define el fichero que ejecutará el sistema como arranque o inicio de este proceso y que su valor se anexa al path raíz declarado en el parámetro anterior.
- tercero está el parámetro obligatoriedad que define que tipo de inclusión se hará con el fichero de arranque, los posibles valores para la obligatoriedad son 1- require_once 2- require 3-include_once 4- include y 5-eval, solo se pone el numero en representación del tipo e inclusión, estos datos son de obligatoria definición.

- cuarto parámetro (opcional) es condicion_ejecucion que define una expresión que al ser evaluada dentro de un if() si resulta true se ejecutará el proceso, si la evaluación de la expresión resulta false no se ejecutará el proceso.###

A continuación un ejemplo de la estructura bloques de procesos

```
bloques_procesos:
        bloque_defecto:
                pprueba:
                    gedee_proceso:
                                namespace: \Emod\Nucleo\Gedees
                                clase: GedeeEComun
                                id_entidad: GedeeEComun
                    propiedades_implementacion_proceso:
                                    path_raiz: proceso_prueba
                                    path_arranque: control_prueba.php
                                    obligatoriedad: 1
                                    condicion_ejecucion: $_GET['a'] ==2
                pprueba2:
                    gedee_proceso:
                                namespace: \Emod\\Nucleo\Gedees
                                clase: GedeeEComun
                                id_entidad: GedeeEComun
                    propiedades_implementacion_proceso:
                                    path_raiz: proceso_prueba_2
                                    path_arranque: control_prueba.php
                                    obligatoriedad: 1
        mi_bloque:...
            *
            *
            *
```

e_nucleo_control.php

Este fichero se divide en varias secciones por razones organizativas, a continuación explicamos cada una de estas secciones.

- sección inclusión bibliotecas de clases y funciones del núcleo, en esta sección se incluyen los ficheros que conforman la estructura básica del motor del núcleo del sistema Sockeletom.

- sección instanciación de clases del núcleo, en esta sección se crean las instancias de los objetos que conforman la estructura básica del motor del núcleo del sistema Sockeletom.

- sección actualización del contenedor de referencias permanentes a objetos del núcleo Sockeletom, este contenedor existe para el acceso desde cualquier parte del sistema y sus códigos clientes, a los objetos del núcleo Sockeletom.

- sección inicialización de los objetos del núcleo, es la inicialización de los objetos del núcleo que precisan inicialización.

- sección gestión de dependencias entre objetos del núcleo, esta sección se encarga de inyectar las dependencias (referencias a objetos) que cada objeto del núcleo necesita para su correcto desempeño en la gestión del sistema Sockeletom, la palabra entre es por hecho que estas dependencias son entre objetos pertenecientes todos al núcleo.

- sección carga de configuración del sistema Sockeletom, es la sección donde se ejecuta la acción de cargar los valores de configuración del sistema Sockeletom en el espacio para la configuración que para ello tiene el objeto EEoNucleo. Este espacio es brindado por el objeto EEoConfiguracion.

- sección inicialización de los dato de seguridad del proceso núcleo del sistema Sockeletom, esta sección ejecuta la toma de los datos de seguridad del proceso núcleo, declarados en la configuración del sistema, y cargados estos valores en el espacio para la seguridad que para ello tiene el objeto EEoNucleo. Este espacio es brindado por el objeto EEoSeguridad.

- sección ejecución del bloque de procesos configurado para el sistema Sockeletom, esta sección se encarga de desatar la ejecución de la lista de procesos que se encuentran declarados en la configuración del sistema. Este espacio depende de la interacción entre el objeto EEoNucleo y el objeto EEoImplementacionProcesos.

class_epatron_multiton.php

Este fichero contiene la programación del patrón singlenton pero de forma tal que cualquier clase que herede de este patrón, lo pueda implementar sin hacer cambios en el código, sin adaptarlo a la clase específica que hereda , esto se debe a que el procedimiento detecta mediante get_called_class() la clase que lo ejecuta y crea la instancia de esa clase, convirtiéndose en un código de uso múltiple sin corrección de código al heredar, de ahí el nombre de multiton.

Namespace: Emod\Nucleo\Herramientas
Clase Abstracta
Nombre de la clase: Multiton
Herencia:

Propiedades:
protected static $laInstancias = array();
Es la propiedad que albergará la referencia al objeto que se cree, y su valor por defecto es array();.

Procedimientos:

final public static function instanciar()
Es el procedimiento que instanciará la clase que lo llama

private function __construct(){}
Es el procedimiento constructor que se privatiza para no permitir instanciar por esta vía la clase

final private function __clone(){}
Es el procedimiento para clonar el objeto que se privatiza para no permitir clonar una instancia de la clase

trait_dependencias_entidades_emod.php

Este fichero es un trait para la constitución de clases que tienen dependencias de entidades del núcleo Sockeletom, a continuación explicamos su estructura:

Namespace: Emod\Nucleo
Nombre del trait: DependenciasEntidadesEmod
Herencia:

Propiedades:
protected $EEoNucleo = null ;
Es la propiedad que albergará una referencia al objeto EEoNucleo y su valor por defecto es null.

protected $EEoInterfazDatos = null ;
Es la propiedad que albergará una referencia al objeto EEoInterfazDatos y su valor por defecto es null.

protected $EEoConfiguracion = null ;
Es la propiedad que albergará una referencia al objeto EEoConfiguracion y su valor por defecto es null.

protected $EEoSeguridad = null ;
Es la propiedad que albergará una referencia al objeto EEoSeguridad y su valor por defecto es null.

protected $EEoDatos = null ;
Es la propiedad que albergará una referencia al objeto EEoDatos y su valor por defecto es null.

protected $EEoImplementacionProcesos = null ;
Es la propiedad que albergará una referencia al objeto EEoImplementacionProcesos y su valor por defecto es null.

Procedimientos:

final public function cargarObjetosDependencia ($id_objeto , $class , $namespace)
procedimiento para cargar las referencias a objetos que conforman la base o núcleo del sistema Sockeletom, este procedimiento es quien inicializa las propiedades de esta clase.

$id_objeto es un identificador o nombre del objeto, que deberá coincidir con los id predefinidos en este procedimiento para la asignación del objeto a la propiedad correspondiente en esta clase. Si su valor es vacío el procedimiento retorna el valor null. Sus posibles valores son ' EEoNucleo ', 'EEoInterfazDatos',' EEoSeguridad ',' EEoConfiguracion ', EEoDatos ', 'EEoImplementacionProcesos'.

$class es la clase a la que pertenece el objeto por asignar.

$namespace es el namespace al que pertenece el objeto por asignar.

Este procedimiento retorna true si la operación es satisfactoria y null si es insatisfactoria.

class_enucleo_entidad_base.php

Este fichero es una clase intermedia para la constitución de clases que conforman la base o núcleo del sistema Sockeletom, a continuación explicamos su estructura:

Namespace: Emod\Nucleo
Nombre de la clase: NucleoEntidadBase
Herencia: extends \Emod\Nucleo\Herramientas\Multiton
use \Emod\Nucleo\DependenciasEntidadesEmod

Procedimientos:

final protected function implementarProcedimientosEE (&$estructura_datos , $nombre_funcion , &$La_argumentos = null , $chequeo_procesoygedee = false)
este procedimiento fabrica, implementa código para que los objetos que heredan de esta clase puedan gestionar sus estructuras de datos en dependencia del dominio tipo proceso que haga la petición de gestión, y el dominio tipo proceso que sea gestionado,

$estructura_datos es la estructura de datos sobre la que operara la función implementada. Es el primer parámetro que se le pasará a la función que se implementará.

$nombre_funcion es el nombre de la función que se implementará.

$La_argumentos es un arreglo con los parámetros que se le pasaran a la función que se implementará, cuando se fabrica la función se le ubican los parámetros en el orden que tienen los argumentos de este arreglo y a partir del segundo parámetro, ya que el primero lo constituye $estructura_datos explicado anteriormente.

$chequeo_procesoygedee es para definir si la función que se implementará debe chequear el GEDEE del proceso al que se quiere acceder o gestionar sus datos. Este parámetro existe porque existen funciones a implementar que no requieren de un chequeo del GEDEE a gestionar porque siempre trabajan sobre su propio GEDEE .

Este procedimiento concluye con exit si su gestión no es satisfactoria.

class_enucleo_control.php

Este fichero es la clase que conforma el objeto EEoNucleo, y que junto con el fichero **e_nucleo_control.php** son los controladores del proceso núcleo del sistema Sockeletom, a continuación explicamos su estructura:

Namespace: Emod\Nucleo
Nombre de la clase: NucleoControl
Herencia: extends \Emod\Nucleo\NucleoEntidadBase

Propiedades:

protected $EEoInterfazDatos = null ;
puntero al objeto EEoInterfazDatos ;

protected $EEoConfiguracion = null ;
puntero al objeto EEoConfiguracion ;

protected $EEoSeguridad = null ;
puntero al objeto EEoSeguridad ;

protected $EEoImplementacionProcesos = null ;
puntero al objeto EEoImplementacionProcesos ;

private $lsPathDirEsquelemod = null ;
valor del path del directorio esquelemod, esta propiedad debe tener valor vacío al instanciarse el objeto núcleo ya que de esto depende todo el proceso de inicialización del objeto. Su valor se asigna a través de un procedimiento de esta clase.

private $lsPathDirModulos = null ;
valor del path del directorio módulos, se fabrica a partir del valor del directorio esquelemod, su valor contiene la misma raíz que el directorio esquelemod seguido del directorio módulos. Su valor se asigna a través de un procedimiento de esta clase.

private $lsIdProcesoNucleo = ' NucleoEsquelemod ' ;
es el identificador del proceso núcleo para todas las operaciones que necesitan este indicador, como son en las estructuras de datos de los objetos del núcleo ' EEoNucleo ', 'EEoInterfazDatos',' EEoSeguridad ',' EEoConfiguracion ', EEoDatos ', 'EEoImplementacionProcesos', entre otros. Su valor se define por defecto en esta misma clase cambiándose por el valor que se asigne posteriormente en el fichero de configuración del sistema.

private $lsNamespaceGedeeProcesoNucleo = ' \\Emod\\Nucleo\\Gedees' ;
es el namespace del GEDEE al que pertenece el proceso enucleo, su valor se define por defecto en esta misma clase cambiándose por el valor que se asigne posteriormente en el fichero de configuración del sistema.

private $lsClaseGedeeProcesoNucleo = ' GedeeENucleo' ;
es la clase del GEDEE al que pertenece el proceso enucleo, su valor se define por defecto en esta misma clase cambiándose por el valor que se asigne posteriormente en el fichero de configuración del sistema.

private $lsIdGedeeProcesoNucleo = ' GedeeENucleo ' ;
es el GEDEE al que pertenece el proceso enucleo, su valor se define por defecto en esta misma clase cambiándose por el valor que se asigne posteriormente en el fichero de configuración del sistema.

private $lbIniciacionGedee = null ;
es el identificador de la iniciación de la clase GEDEEs, si fue ejecutada con éxito su valor cambia a true. Su valor se asigna a través de un procedimiento de esta clase.

private $lbEConfiguracionEjecutada = false ;
identifica si la configuración del sistema Sockeletom fue ejecutada con éxito, su valor se asigna a través de un procedimiento de esta clase.

private $lbESeguridadIniciada = false ;
identifica si la sección de seguridad del sistema Sockeletom fue inicializada con éxito, su valor se asigna a través de un procedimiento de esta clase.

private $lbIniciacionErrores = null;
identifica si la sección de Errores del sistema Sockeletom fue inicializada con éxito, su valor se asigna a través de un procedimiento de esta clase.

private $lbIniciacionHeramientas = null;
identifica si la sección de herramientas del sistema Sockeletom fue inicializada con éxito, su valor se asigna a través de un procedimiento de esta clase.

private $lbIniciacionUtiles = null;
identifica si la sección de útiles del sistema Sockeletom fue inicializada con éxito, su valor se asigna a través de un procedimiento de esta clase.

private $lsPathDirRaizProcesoEjecucion= ' ' ;
valor del directorio raíz del proceso en ejecución, es el proceso que se ejecuta en el momento que esta propiedad es consultada, su valor contendrá el nombre o identificador de este proceso, su valor se asigna a través de un procedimiento de esta clase.

private $lsIdProcesoEjecucion = ' ' ;
valor del proceso en ejecución, es el proceso que se ejecuta en el momento que esta propiedad es consultada, su valor contendrá el path del directorio raíz y este a su vez se encuentra en el directorio procesos del sistema de archivos del sistema Sockeletom, su valor se asigna a través de un procedimiento de esta clase.

private lsNamespaceGedeeProcesoEjecucion = '' ;
valor del namespace del GEDEE del proceso en ejecución, es el namespace del GEDEE del proceso que se ejecuta en el momento que esta propiedad es consultada, su valor contendrá el nombre o identificador de este namespace de GEDEE, su valor se asigna a través de un procedimiento de esta clase.

private lsClaseGedeeProcesoEjecucion = '' ;
valor de la clase del GEDEE del proceso en ejecución, es la clase del GEDEE del proceso que se ejecuta en el momento que esta propiedad es consultada, su valor contendrá el nombre o identificador de esta clase de GEDEE, su valor se asigna a través de un procedimiento de esta clase.

private lsIdGedeeProcesoEjecucion = '' ;
valor del id del GEDEE del proceso en ejecución, es el id del GEDEE del proceso que se ejecuta en el momento que esta propiedad es consultada, su valor contendrá el nombre o identificador del GEDEE, su valor se asigna a través de un procedimiento de esta clase.

private $laArbolProcesos = null ;
es un historial de todos los procesos que se ejecutan, cada proceso se guarda en este árbol según el orden en que se ejecutó y con los siguientes, parámetros: 'id_proceso', 'gedee', 'apuntador_procesoPadre', 'idorejec_subproceso', 'id_clave_ejecucion', 'procesos'. Para que un proceso quede registrado en este historial debe ser ejecutado utilizando el procedimiento cargaControlProcesos() del objeto 'EEoImplementacionProcesos'. Su valor se asigna a través de un procedimiento de esta clase.

private $idorejecGlobal = 0 ;
número entero identificativo de orden de ejecución a escala global, es decir el orden consecutivo que cada proceso ocupa en la ejecución de procesos durante la ejecución del sistema Sockeletom de forma global. Su valor se asigna a través de un procedimiento de esta clase.

private $limiteIdorejecGlobal = -1 ;
límite para el número entero identificativo de orden de ejecución a escala global, es decir un número límite para el orden consecutivo que cada proceso ocupa en la ejecución de procesos durante la ejecución del sistema Sockeletom de forma global. Su valor se asigna a través de un procedimiento de esta clase. El valor -1 significa que el límite es infinito, los restantes posibles valores son desde 0 hasta infinito positivo.

private $idorejecBloqueProcesos = 0 ;
número entero identificativo de orden de ejecución en el bloque de procesos, es decir el orden consecutivo que cada proceso ocupa en el bloque declarado en la configuración del sistema. Su valor se asigna a través de un procedimiento de esta clase.

private $limiteIdorejecBloqueProcesos = -1 ;
límite para el número entero identificativo de orden de ejecución en el bloque de procesos, es decir un número límite para el orden consecutivo que cada proceso ocupa en el bloque declarado en la configuración del sistema. Su valor se asigna a través de un procedimiento de esta clase.El valor -1 significa que el límite es infinito, los restantes posibles valores son desde 0 hasta infinito positivo.

private $idorejecSubProceso = 0 ;
número entero identificativo de orden de ejecución de un proceso como subproceso de otro, es decir el orden consecutivo que cada proceso ocupa en la ejecución de procesos como subproceso del proceso padre. Su valor se asigna a través de un procedimiento de esta clase.

private $limiteIdorejecSubProceso = -1 ;
límite para el número entero identificativo de orden de ejecución de un proceso como subproceso de otro, es decir un número límite para el orden consecutivo que cada proceso ocupa en la ejecución de procesos como subproceso del proceso padre. Su valor se asigna a través de un procedimiento de esta clase. El valor -1 significa que el límite es infinito, los restantes posibles valores son desde 0 hasta infinito positivo.

private $idorejecRecursivoProceso = 0 ;
número entero identificativo de orden de ejecución de un proceso de forma recursiva, es decir el orden consecutivo que cada proceso ocupa cuando es ejecutado por él mismo, de forma recursiva. Su valor se asigna a través de un procedimiento de esta clase.

private $limiteIdorejecRecursivoProceso = -1 ;
límite para el número entero identificativo de orden de ejecución de un proceso de forma recursiva, es decir un número límite para el orden consecutivo que cada proceso ocupa cuando es ejecutado

por él mismo, de forma recursiva. Su valor se asigna a través de un procedimiento de esta clase. El valor -1 significa que el límite es infinito, los restantes posibles valores son desde 0 hasta infinito positivo.

private $idorejecRecursivoProImpEjec = 0 ;
número entero identificativo de orden de ejecución de procesos cuando el procedimiento que implementa la ejecución los procesos es llamado recursivamente, es decir el que es llamado recursivamente es el procedimiento de ejecución, no el proceso, es cuando un proceso llama dentro de sí el procedimiento de ejecución para ejecutarse a sí mismo o a otro proceso, se refiere a la llamada del procedimiento que implementa la ejecución de los procesos no desde el núcleo del sistema sino desde un proceso ya ejecutándose. Su valor se asigna a través de un procedimiento de esta clase.

private $limiteIdorejecRecursivoProImpEjec = -1 ;
límite para el número entero identificativo de orden de ejecución de procesos cuando el procedimiento que implementa la ejecución de los procesos es llamado recursivamente, es decir un número límite de veces que es llamado recursivamente el procedimiento de ejecución, no el proceso, es cuando un proceso llama dentro de sí el procedimiento de ejecución para ejecutarse a sí mismo o a otro proceso, se refiere a la llamada del procedimiento que implementa la ejecución de los procesos no desde el núcleo del sistema sino desde un proceso ya ejecutándose. Su valor se asigna a través de un procedimiento de esta clase. El valor -1 significa que el límite es infinito, los restantes posibles valores son desde 0 hasta infinito positivo.

private $apuntadorProcesoActual = null ;
es un puntero al arreglo estructura del proceso actual(ejecutándose) y situada en el arreglo $ laArbolProcesos de esta misma clase. Su valor se asigna a través de un procedimiento de esta clase.

private $statuEjecucionProceso = null ;
estado de la ejecución del proceso desde el bloque de procesos, cada proceso que se ejecuta desde el bloque de procesos declarado en la configuración del Sockeletom, es chequeado por el núcleo el hecho de que este proceso haya finalizado, en cada proceso que se comienza a ejecutar el núcleo del sistema pone esta propiedad con valor inicializado y hasta que esta propiedad no cambie este valor a null el sistema no ejecuta otro proceso. Los cambios de valor de esta propiedad solo los hace el núcleo. Su valor se asigna a través de un procedimiento de esta clase.

private $lsEjecucionBloquesProcesos = false ;
marca para definir si ya se ejecutaron o ejecuto algún bloque de procesos. Su valor se asigna a través de un procedimiento de esta clase.

private $lsIdBloqueEjecucionProcesos = null ;
identificador nombre del bloque de procesos actual, en ejecución. Su valor se asigna a través de un procedimiento de esta clase.

private $lsUbicacionPedidoActualizacionComienzoProceso = null ;
esta propiedad es para guardar y luego comparar el lugar desde el que se hace el pedido de actualización de parámetros de procesos que comienzan su ejecución, los parámetros que se actualizan son lo que tienen que ver con el seguimiento que hace el núcleo a los procesos que se ejecutan utilizando el objeto EEoImplementacionProcesos, esta propiedad existe con el objetivo que los procesos no puedan tergiversar los datos de su ejecución en el núcleo y está estrechamente vinculada con el árbol de ejecución procesos. Su valor se asigna a través de un procedimiento de esta clase.

private $lsUbicacionPedidoActualizacionFinalizacionProceso = null ;
esta propiedad es para guardar y luego comparar el lugar desde el que se hace el pedido de actualización de parámetros de procesos que terminan su ejecución, los parámetros que se actualizan son lo que tienen que ver con el seguimiento que hace el núcleo a los procesos que se ejecutan utilizando el objeto EEoImplementacionProcesos, esta propiedad existe con el objetivo que los procesos no puedan tergiversar los datos de su ejecución en el núcleo y está estrechamente vinculada con el árbol de ejecución procesos. Su valor se asigna a través de un procedimiento de esta clase.

Procedimientos:

final private function iniciacionClassGedees ($La_configuracion_nucleo_gedees = null)
este procedimiento inicializa la clase de métodos abstractos GEDEEs para el trabajo con los GEDEEs de procesos que se gestionaran en el sistema.

$La_configuracion_nucleo_gedees es la sección de configuración ¨Valores referentes a los GEDEE¨ que se encuentra en la configuración del sistema Sockeletom.

final public function iniciacionImplementacionNucleo ($cantidad_niveles = 3)
este procedimiento define el path global del directorio esquelemod y el directorio módulos, necesarios en la implementación de otros códigos del sistema.

$cantidad_niveles es la cantidad de niveles de directorio que hay que ir hacia atrás, en el árbol de directorio, para llegar al directorio esquelemod desde el lugar donde se encuentra el fichero de esta clase, la cantidad se comienza a contar a partir de 1, no de cero, y por defecto tiene el valor 3.

public function pathDirEsquelemod()
este procedimiento retorna el valor absoluto del path del directorio esquelemod, valor definido en la propiedad $this-> lsPathDirEsquelemod de esta clase.

public function pathDirModulos()
este procedimiento retorna el valor absoluto del path del directorio módulos, valor definido en la propiedad $this->lsPathDirModulos de esta clase.

public function idProcesoNucleo()
este procedimiento retorna el valor del id del proceso núcleo, valor definido en la propiedad $this->lsIdProcesoNucleo de esta clase.

public function gedeeProcesoNucleo ()
este procedimiento retorna el valor del id o nombre del namespace del GEDEE concatenado con el valor del id o nombre de la clase del gedee concatenado con el valor del id o nombre del gedee del proceso núcleo, valores definidos en las propiedades $this-> lsNamespaceGedeeProcesoNucleo, $this->lsClaseGedeeProcesoNucleo y $this->lsIdGedeeProcesoNucleo de esta clase. Quedando de la siguiente forma:
$this->lsNamespaceGedeeProcesoNucleo.'\\'.$this->lsClaseGedeeProcesoNucleo .'\\'.$this->lsIdGedeeProcesoNucleo

public function idGedeeProcesoNucleo ()
este procedimiento retorna el valor del id o nombre del GEDEE del proceso núcleo, valor definido en la propiedad $this->lsIdGedeeProcesoNucleo.

public function namespaceGedeeProcesoNucleo ()
este procedimiento retorna el valor del id o nombre del namespace del GEDEE del proceso núcleo, valor definido en la propiedad $this->lsNamespaceGedeeProcesoNucleo.

public function claseGedeeProcesoNucleo ()
este procedimiento retorna el valor del id o nombre de la clase del GEDEE del proceso núcleo, valor definido en la propiedad $this->lsClaseGedeeProcesoNucleo.

public function gestionIniciacionConfiguracionEsquelemod($La_fich_interfaz_config , $La_fich_datos_config = null)

este procedimiento gestiona los datos de configuración del sistema Sockeletom y los inicia en el objeto EEoConfiguracion para su posterior utilización, los parámetros de este procedimiento, que explicamos a continuación, responden directamente a la utilización dentro de este procedimiento del objeto 'EEoInterfazDatos' para la gestión de los datos de configuración. Este procedimiento retorna true si su gestión fue exitosa, de lo contrario retorna null.

$La_fich_interfaz_config['Ls_pathfich_interfaz'] (string) es el path del fichero interfaz, En el caso que se quiera utilizar el path que por defecto tiene asignado el objeto instancia de esta clase, entonces se puede poner el valor 'hereda' o el path explícitamente, de ser así no se tienen en cuenta los demás elementos de este arreglo $La_fich_interfaz_config. Este elemento debe tener valor diferente de vacío, de lo contrario el procedimiento retorna null

- $La_fich_interfaz_config['Ls_path_base'] (string) (opcional) es un fragmento de path que se inserta al principio del path $La_fich_interfaz_config['Ls_pathfich_interfaz'] quedando de la siguiente forma

$La_fich_interfaz_config['Ls_path_base'].'/'. $La_fich_interfaz_config['Ls_pathfich_interfaz']

- $La_fich_interfaz_config['Ls_path_operativo'] (string) (opcional) es un fragmento de path que se inserta al principio del path $La_fich_interfaz_config['Ls_pathfich_interfaz'] y al final del path $La_fich_interfaz_config['Ls_path_base'] quedando de la siguiente forma

$La_fich_interfaz_config['Ls_path_base'] .'/'. $La_fich_interfaz_config['Ls_path_operativo'] .'/'. $La_fich_interfaz_config['Ls_pathfich_interfaz']

este elemento, fragmento de path, se recorrerá desde la rama hasta la raíz en busca del fichero interfaz declarado en $La_fich_interfaz_config['Ls_pathfich_interfaz'], es decir, se puede poner el fichero interfaz en cualquiera de los directorios de este elemento y el sistema lo encontrará y formará el path real del fichero interfaz, para su tratamiento por parte del procedimiento actual. Esta característica dota al procedimiento de la posibilidad de tener un fichero interfaz para varios subdirectorios dentro de los que se puede considerar la posibilidad de tener o no otro fichero interfaz.

$La_fich_datos_config (array) (opcional) contendrá los siguientes elementos:

- $La_fich_datos_config['Ls_pathfich_datos'] (string) (opcional) es el path del fichero contenedor de datos que será procesado, parseado, interpretado por el fichero interfaz.
- $La_fich_datos_config['Ls_path_base'] (string) (opcional) es un fragmento de path que se inserta al principio del path $La_fich_datos_config['Ls_pathfich_datos'] quedando de la siguiente forma

$La_fich_datos_config['Ls_path_base'].'/'. $La_fich_datos_config['Ls_pathfich_datos']

- $La_fich_datos_config['Ls_path_operativo'] (string) (opcional) es un fragmento de path que se inserta al principio del path $La_fich_datos_config['Ls_pathfich_datos'] y al final del path $La_fich_datos_config['Ls_path_base'] quedando de la siguiente forma

$La_fich_datos_config['Ls_path_base'] .'/'. $La_fich_datos_config['Ls_path_operativo'] .'/'. $La_fich_datos_config['Ls_pathfich_datos']

este elemento, fragmento de path, se recorrerá desde la rama hasta la raíz en busca del fichero datos declarado en $La_fich_datos_config['Ls_pathfich_datos'], es decir, se puede poner el fichero datos en cualquiera de los directorios de este elemento y el sistema lo encontrará y formará el path real del fichero datos, para su tratamiento por parte del procedimiento actual. Esta característica dota al procedimiento de la posibilidad de tener un fichero datos para varios subdirectorios dentro de los que se puede considerar la posibilidad de tener o no otro fichero datos.

public function iniciacionDatosSeguridadEsquelemod()

este procedimiento inicia los datos de seguridad del proceso núcleo en el objeto EEoSeguridad para su posterior utilización. Estos datos de seguridad están estructurados según el dominio tipo de proceso al que pertenece el núcleo, en este caso es ¨ecomun¨. Los datos de seguridad los toma este procedimiento de los datos de configuración del sistema Sockeletom que son los mismos que los datos de configuración que el proceso núcleo inició en el objeto EEoConfiguracion. Este procedimiento retorna true si su gestión fue exitosa, de lo contrario retorna null.

public function accesoPropiedadesEsquelemod()
este procedimiento es para acceder a las propiedades declaradas en la sección ['propiedades_proceso'] del fichero de configuración del sistema. Los datos los toma este procedimiento de los datos de configuración del sistema Sockeletom que son los mismos que los datos de configuración que el proceso núcleo inició en el objeto EEoConfiguracion. Este procedimiento retorna un arreglo con la estructura correspondiente a los datos plasmados en la sección [sistema] subsección ['propiedades_proceso'] del fichero de configuración del sistema si su gestión fue exitosa, de lo contrario retorna null.

private function iniciacionErrores($La_configuracion_nucleo_errores = null)
este procedimiento es para iniciar la clase Errores, encargada de contener y facilitar las entidades (objetos o clase) que se comportan como gestores de errores en el sistema Sockeletom. Este procedimiento solo se ejecuta si la variable ejecución, contenida en la sección errores de la configuración del Sockeletom, tiene valor true .Este procedimiento retorna true si la gestión tuvo éxito de lo contrario termina la ejecución del intérprete de php (exit) con un mensaje de error. A continuación la descripción de su parámetro:

$La_configuracion_nucleo_errores es un arreglo con los datos contenidos en la sección ¨errores¨ de la configuración del sistema por esa razón la explicación de los elementos de este arreglo se sitúan en la sección de ayuda de la configuración del sistema Sockeletom en este mismo documento.

private function iniciacionClassHerramientas($La_configuracion_nucleo_herramientas = null)
este procedimiento es para iniciar la clase Herramientas, encargada de contener y facilitar las entidades (objetos o clase) que se comportan como herramientas en el sistema Sockeletom. Este procedimiento retorna true si la gestión tuvo éxito de lo contrario termina la ejecución del intérprete de php (exit) con un mensaje de error. A continuación la descripción de su parámetro:

$La_configuracion_nucleo_herramientas es un arreglo con los datos contenidos en la sección ¨herramientas¨ de la configuración del sistema por esa razón la explicación de los elementos de este arreglo se sitúan en la sección de ayuda de la configuración del sistema Sockeletom en este mismo documento.

private function iniciacionClassUtiles($La_configuracion_nucleo_utiles = null)

este procedimiento es para iniciar la clase Utiles, encargada de contener y facilitar las entidades (objetos o clase) que se comportan como útiles en el sistema Sockeletom. Este procedimiento retorna true si la gestión tuvo éxito de lo contrario termina la ejecución del intérprete de php (exit) con un mensaje de error. A continuación la descripción de su parámetro:

$La_configuracion_nucleo_utiles es un arreglo con los datos contenidos en la sección ¨utiles¨ de la configuración del sistema por esa razón la explicación de los elementos de este arreglo se sitúan en la sección de ayuda de la configuración del sistema Sockeletom en este mismo documento.

private function iniciacionProcesos ($La_configuracion_nucleo_procesos = null)
este procedimiento es para chequear e/o iniciar cualquier elemento o propiedad de esta clase que dependa de la sección procesos de la configuración del sistema, un ejemplo es la estructura

```
procesos:
        arbol_procesos:
                limite_idorejec_global: -1
                limite_idorejec_bloqueprocesos: -1
                limite_idorejec_subproceso: -1
                limite_idorejec_recursivoproceso: -1
                limite_idorejec_recursivoproimpejec: -1
```

A continuación la descripción de su parámetro:

$La_configuracion_nucleo_procesos es un array con la sección procesos de la configuración del sistema.

public function permisionProcesoNucleo($Ls_id_proceso)
este procedimiento es para chequear si un proceso de namespace\clase núcleo (tipo núcleo) tiene permiso de ejecución, los procesos con permiso de ejecución deben ponerse con anterioridad en una sección que la configuración del sistema tiene para esta prestación, el sistema para ejecutar un proceso núcleo se rige por este procedimiento. A continuación la descripción de su parámetro:

$Ls_id_proceso es un string con el id del proceso núcleo a chequear permiso de ejecución.

public function permisionProcesoCliente($Ls_id_proceso , $Ls_clase_gedee_proceso , $Ls_namespace_gedee_proceso)
este procedimiento es para chequear si un proceso cualquiera (tipo clientes, no de tipo núcleo) tiene permiso de ejecución, los procesos con permiso de ejecución deben ponerse con anterioridad en una sección que la configuración del sistema tiene para esta prestación, el sistema para ejecutar un proceso cliente se rige por este procedimiento. A continuación la descripción de sus parámetro:

$Ls_id_proceso es un string con el id del proceso a chequear permiso de ejecución.

$Ls_clase_gedee_proceso es un string con la clase del GEDEE a utilizar por el proceso a chequear permiso de ejecución.

$Ls_namespace_gedee_proceso es un string con el namespace del GEDEE a utilizar por el proceso a chequear permiso de ejecución.

public function bloqueEjecucionProcesos()
este procedimiento retorna el identificador nombre del bloque de procesos actual, en ejecución, el valor de retorno lo toma de la propiedad $lsIdBloqueEjecucionProcesos de esta clase.

private function ejecutarBloqueProcesos($id_bloque)
este procedimiento ejecuta en orden consecutivo la lista de procesos que contiene el bloque identificado con el id o nombre $id_bloque, en la configuración del sistema Sockeletom.

$id_bloque es el id o nombre de una lista de procesos y sus parámetros que se declara en la configuración del sistema Sockeletom en la sección ¨Definición de bloques, Valores referentes a Procesos a ejecutar en cada bloque¨.

Este procedimiento retorna true si su gestión fue exitosa, de lo contrario retorna null.

public function ejecutarBloquesProcesos()
este procedimiento ejecuta los bloque declarados para ejecución en el fichero de configuración del sistema, la sección Bloques de procesos a cargar por el sistema de la configuración del Sockeletom. De no poder comenzar a ejecutar los bloques retornará null.

public function pathDirRaizProcesos()
este procedimiento retorna el valor del path raíz del directorio de procesos, declarado en la configuración del sistema, teniendo en cuenta que el path que aquí se especifique se incorporará al path raíz_emod/esquelemod/e_modulos/ quedando de la siguiente forma ¨raíz_emod/esquelemod/e_modulos/pathRaizProcesos()¨. Los datos los toma este procedimiento de los datos de configuración del sistema Sockeletom que son los mismos que los datos de configuración que el proceso núcleo inició en el objeto EEoConfiguracion. Este procedimiento retorna un valor vacío si su gestión no fue exitosa.

public function pathDirRaizProcesoEjecucion()
este procedimiento retorna el valor del path raíz del directorio del proceso en ejecución, declarado en la configuración del sistema para el proceso en ejecución, teniendo en cuenta que el path que aquí se especifique se incorporará al path raíz_emod/ pathDirEesquelemod()/e_modulos/pathRaizProcesos() quedando de la siguiente forma ¨ path raíz_emod/ pathDirEesquelemod()/e_modulos/pathRaizProcesos()/pathDirRaizProcesoEjecucion() ¨. Los datos los toma este procedimiento de los datos de configuración del sistema Sockeletom que

son los mismos que los datos de configuración que el proceso núcleo inició en el objeto EEoConfiguracion. Este procedimiento retorna un valor vacío si su gestión no fue exitosa.

public function pathAbsolutoDirRaizProcesoEjecucion()
este procedimiento retorna el valor absoluto del path raíz del directorio del proceso en ejecución, es el path compuesto por :
pathraíz_emod/pathDirEesquelemod()/e_modulos/pathRaizProcesos()/pathDirRaizProcesoEjecucion()
Los datos los toma este procedimiento de los datos de configuración del sistema Sockeletom que son los mismos que los datos de configuración que el proceso núcleo inició en el objeto EEoConfiguracion. Este procedimiento retorna un valor vacío si su gestión no fue exitosa.

public function idProcesoEjecucion ()
este procedimiento retorna el valor del id del proceso en ejecución en el momento en que se ejecuta este procedimiento. Retorna el valor de la propiedad $lsIdProcesoEjecucion de esta clase.

public function gedeeProcesoEjecucion ()
este procedimiento retorna el valor del id o nombre del namespace del GEDEE concatenado con el valor del id o nombre de la clase del GEDEE concatenado con el valor del id o nombre del GEDEE del proceso en ejecución en el momento en que se ejecuta este procedimiento, valores definidos en las propiedades $this-> lsNamespaceGedeeProcesoEjecucion, $this->lsClaseGedeeProcesoEjecucion y $this->lsIdGedeeProcesoEjecucion de esta clase. Quedando de la siguiente forma:
$this->lsNamespaceGedeeProcesoEjecucion.'\\'.$this->lsClaseGedeeProcesoEjecucion.'\\'.$this->lsIdGedeeProcesoEjecucion

public function idGedeeProcesoEjecucion ()
este procedimiento retorna el valor del id o nombre del GEDEE del proceso en ejecución en el momento en que se ejecuta este procedimiento, valor definido en la propiedad $this->lsIdGedeeProcesoEjecucion.

public function namespaceGedeeProcesoEjecucion ()
este procedimiento retorna el valor del id o nombre del namespace del GEDEE del proceso en ejecución en el momento en que se ejecuta este procedimiento, valor definido en la propiedad $this->lsNamespaceGedeeProcesoEjecucion.

public function claseGedeeProcesoEjecucion ()
este procedimiento retorna el valor del id o nombre de la clase del GEDEE del proceso en ejecución en el momento en que se ejecuta este procedimiento, valor definido en la propiedad $this->lsClaseGedeeProcesoEjecucion.

private function procesoHijo($id_proceso , $Ls_path_dir_raiz_proceso)

procedimiento para la actualización del árbol de ejecución de proceso y sus datos, y los punteros que se mueven por su estructura, en este caso es la creación de un nuevo nodo como proceso hijo, y la actualización de apuntadores hacia este próximo nuevo nodo del árbol de ejecución de procesos

$id_proceso es el id del nuevo proceso a actualizar, si su valor es vacío el procedimiento no realiza la gestión y retorna null

$Ls_path_dir_raiz_proceso es el path del directorio raíz del proceso en ejecución, este directorio a su vez tiene como raíz el fichero procesos del sistema de archivos del sistema Sockeletom.

el procedimiento retorna una id-clave-ejecución de este proceso para identificar esta ejecución en particular, y null si fue insatisfactoria la gestión, el id-clave-ejecución de este proceso está compuesto de la siguiente manera "$this->idorejecGlobal::$this->idorejecRecursivoProImpEjec:: $this->idorejecRecursivoProceso".

private function procesoPadre()
procedimiento para la actualización del árbol de ejecución de proceso y sus datos, y los punteros que se mueven por su estructura, en este caso en un movimiento hacia el proceso padre o nodo anterior del árbol de ejecución de procesos, el procedimiento retorna true si su gestión fue satisfactoria y null si fue insatisfactoria.

public function actualizacionComienzoProceso($id_proceso , $Ls_path_dir_raiz_proceso , $Ls_namespace_gedee_proceso , $Ls_clase_gedee_proceso , $ubicacion_pedido_actualizacion) este procedimiento actualiza al objeto implementación de esta clase(objeto e_nucleo)con los datos correspondientes a la ejecución de un nuevo proceso, actualiza también el árbol de ejecución de procesos que es propiedad de esta clase.

$id_proceso es el id del nuevo proceso a actualizar, si su valor es vacío el procedimiento no realiza la gestión y retorna null.

$Ls_path_dir_raiz_proceso es el path del directorio raíz del proceso en ejecución, este directorio a su vez tiene como raíz el fichero procesos del sistema de archivos del sistema Sockeletom.

$Ls_namespace_gedee_proceso es el namespace del gedee del nuevo proceso a actualizar, si su valor es vacío el procedimiento no realiza la gestión y retorna null.

$Ls_clase_gedee_proceso es la clase del GEDEE del nuevo proceso a actualizar, si su valor es vacío el procedimiento no realiza la gestión y retorna null.

$ubicacion_pedido_actualizacion es la combinación de clase-procedimiento y línea desde donde se hace el pedido de ejecución de este procedimiento, luego de la primera ejecución de este procedimiento se guarda este dato en la propiedad

$lsUbicacionPedidoActualizacionComienzoProceso de esta clase para tomarlo como comparación en las demás llamadas a este procedimiento, si no coincidiera en una posterior llamada el procedimiento no realizará su función y retornará null, este dato se mantiene en memoria lógica no física, cuando el objeto muere, muere el dato con él, si su valor es vacío el procedimiento no realiza la gestión y retorna null, el primer proceso en actualizar el árbol de procesos es el núcleo pero este lo hace internamente desde su instancia por lo que no actualiza $lsUbicacionPedidoActualizacionComienzoProceso, este parámetro lo actualiza el segundo proceso que actualiza el árbol de procesos.

el procedimiento retorna una id-clave-ejecucion de este proceso para identificar esta ejecución en particular, y null si fue insatisfactoria la gestión, el id-clave-ejecucion de este proceso está compuesto de la siguiente manera "$this->idorejecGlobal::$this->idorejecRecursivoProImpEjec:: $this->idorejecRecursivoProceso".

public function actualizacionFinalizacionProceso($id_clave_ejecucion , $ubicacion_pedido_actualizacion)
este procedimiento actualiza al objeto implementación de esta clase(objeto e_nucleo)con los datos correspondientes a la finalización de la ejecución de un proceso, actualiza también el árbol de ejecución de procesos que es propiedad de esta clase.

$id_clave_ejecucion es el id clave de ejecución que se le dio al proceso que finaliza, en el comienzo de su ejecución; si su valor es vacio el procedimiento no realiza la gestión y retorna null.

$ubicacion_pedido_actualizacion es la combinación de clase-procedimiento y línea desde donde se hace el pedido de ejecución de este procedimiento, luego de la primera ejecución de este procedimiento se guarda este dato en la propiedad $lsUbicacionPedidoActualizacionFinalizacionProceso de esta clase para tomarlo como comparación en las demás llamadas a este procedimiento, si no coincidiera en una posterior llamada el procedimiento no realizará su función y retornará null, este dato se mantiene en memoria lógica no física, cuando el objeto muere, muere el dato con él, si su valor es vacío el procedimiento no realiza la gestión y retorna null, el primer proceso en actualizar el árbol de procesos es el núcleo pero este lo hace internamente desde su instancia por lo que no actualiza $lsUbicacionPedidoActualizacionFinalizacionProceso, este parámetro lo actualiza el segundo proceso que actualiza el árbol de procesos.

el procedimiento retorna true si su gestión fue satisfactoria, y termina con exit la ejecución del sistema y un mensaje, si fue insatisfactoria.

private function filtrarHistorialArbolProcesos(&$La_historial_procesos , $Li_nivel = 0)
este procedimiento hace una copia de la propiedad $laArbolProcesos de esta clase y filtra esta copia eliminando alguno de sus elementos como son ['apuntador_proceso_padre'] e ["idorejec_subproceso "].

$La_historial_procesos es el arreglo laArbolProcesos al que se va a filtrar, esta variable se pasa por referencia.

$Li_nivel es para el funcionamiento interno del procedimiento, específicamente en la recursión, no es necesario se utilice por el usuario.

public function accederHistorialArbolProcesos()
este procedimiento es para acceder al arreglo historial_arbol_procesos, basado en el arreglo laArbolProcesos de esta clase, arreglo que no modifica ya que hace el filtrado sobre una copia de la variable, filtra esta copia eliminando alguno de sus elementos como son ['apuntador_proceso_padre'] e ["idorejec_subproceso '].

el arreglo tendrá la misma estructura que el arreglo laArbolProcesos de esta clase pero sin los elementos 'apuntador_proceso_padre' e 'idorejec_subproceso'.

public function accederPropiedadesProcesoPadre()
este procedimiento es para acceder a propiedades del proceso padre del proceso actual(ejecutándose).

el procedimiento retorna un arreglo asociativo con los id_clave: 'id_proceso', 'tipo_proceso' y sus valores correspondientes, del proceso padre del proceso actual (ejecutándose).

en caso de no ser exitosa la gestión retornara null.

class_econfiguracion_procesos.php

Este fichero es la clase que conforma el objeto EEoConfiguracion, y que junto a EEoSeguridad, y EEoDatos son los contenedores de datos correspondientes con sus nombres y los procedimientos para gestionar estos, es decir es el contenedor de datos de configuración de los procesos, y las herramientas para gestionarlos, la estructura de los datos depende del GEDEE del proceso, esta estructura hay que conocerla pero su gestión es transparente para el programador del proceso, a continuación explicamos la estructura y contenido de esta clase:

Namespace: Emod\Nucleo
Nombre de la clase: ConfiguracionProcesos
Herencia: extends \Emod\Nucleo\NucleoEntidadBase

Propiedades:

protected $EEoNucleo = null ;
puntero al objeto EEoNucleo ;

protected $EEoInterfazDatos = null ;
puntero al objeto EEoInterfazDatos ;

protected $EEoSeguridad = null ;
puntero al objeto EEoSeguridad ;

private $datosConfiguracionProcesos = array() ;
valores de configuración que cada proceso guarda en la instancia de esta clase. Su valor primero y gestión de cambios se realizan a través de procedimiento de esta clase, la estructura de los datos de la configuración no tiene que ser uniforme ya que pueden coexistir en esta propiedad diferentes estructuras de datos de configuración, tantos como procesos determinen poner sus datos de configuración aquí, lo que si se mantiene uniforme es que cada proceso tendrá sus datos en el elemento id_ proceso que a su vez se encontrará en el elemento [namespace][clase] del GEDEE al que pertenece el proceso, a continuación un ejemplo para un proceso de id o nombre mi_proceso y perteneciente al GEDEE de namespace \Emod\Nucleo\Gedees y clase GedeeEComun:

$datos_configuracion_procesos[‘\Emod\Nucleo\Gedees’] GedeeEComun’] [‘ mi_proceso’] = datos

Procedimientos:

Antes de entrar en la explicación de cada procedimiento debemos aclarar que cada procedimiento de esta clase tiene un homólogo en las entidades correspondientes a los GEDEEs, y son estos procedimientos de entidades GEDEEs los que realizan la gestión de datos sobre el contenedor de datos $datos_configuracion_procesos de esta clase, por lo que se necesita tener conocimiento de esta función del GEDEE para pasar los parámetros correctamente y conocer qué gestión se hace realmente sobre el contenedor de datos $datos_configuracion_procesos de esta clase.

public function iniciarDatosConfiguracionProceso()
este procedimiento inicia los datos de configuración de un proceso, este procedimiento no tiene parámetros visible en la definición del procedimiento porque utiliza listas variables de parámetros, por lo que se debe tener en cuenta el orden de estos para un correcto desempeño del procedimiento, el orden de los parámetros se corresponde con la función que tiene el mismo nombre que esta, en la entidad del GEDEE al que pertenece la gestión en curso.

public function existenciaIdProceso()
este procedimiento chequea la existencia del proceso y sus datos en el contenedor $datos_configuracion_procesos de esta clase, este procedimiento no tiene parámetros visible en la definición del procedimiento porque utiliza listas variables de parámetros, por lo que se debe tener en cuenta el orden de estos para un correcto desempeño del procedimiento, el orden de los parámetros se corresponde con la función que tiene el mismo nombre que esta, en la entidad del GEDEE al que pertenece la gestión en curso.

public function accederDatosConfiguracionProceso()
este procedimiento es la opción para acceder a los datos de configuración del proceso en el contenedor $datos_configuracion_procesos de esta clase, este procedimiento no tiene parámetros visible en la definición del procedimiento porque utiliza listas variables de parámetros, por lo que se debe tener en cuenta el orden de estos para un correcto desempeño del procedimiento, el orden de los parámetros se corresponde con la función que tiene el mismo nombre que esta, en la entidad del GEDEE al que pertenece la gestión en curso.

class_eseguridad_procesos.php

Este fichero es la clase que conforma el objeto EEoSeguridad, y que junto a EEoConfiguracion, y EEoDatos son los contenedores de datos correspondientes con sus nombres y los procedimientos para gestionar estos, es decir es el contenedor de datos de seguridad de los procesos, y las herramientas para gestionarlos, la estructura de los datos depende del GEDEE, esta estructura hay que conocerla pero su gestión es transparente para el programador del proceso, a continuación explicamos la estructura y contenido de esta clase:

Namespace: Emod\Nucleo
Nombre de la clase: SeguridadProcesos
Herencia: extends \Emod\Nucleo\NucleoEntidadBase

Propiedades:

protected $EEoNucleo = null ;
puntero al objeto EEoNucleo ;

private $datosSeguridadProcesos = array() ;
valores de seguridad que cada proceso guarda en la instancia de esta clase. Su valor primero y gestión de cambios se realizan a través de procedimiento de esta clase, la estructura de estos datos de seguridad no tiene que ser uniforme ya que pueden coexistir en esta propiedad diferentes estructuras de datos de seguridad, tantos como procesos determinen poner sus datos de seguridad aquí, lo que si se mantiene uniforme es que cada proceso tendrá sus datos en el elemento id_ proceso que a su vez se encontrará en el elemento [namespace][clase] del GEDEE al que pertenece el proceso, a continuación un ejemplo para un proceso de id o nombre mi_proceso y perteneciente al GEDEE de namespace \Emod\Nucleo\Gedees y clase GedeeEComun:

$datosSeguridadProcesos [‘\Emod\Nucleo\Gedees’] GedeeEComun’] [‘ mi_proceso’] = datos

Procedimientos:

Antes de entrar en la explicación de cada procedimiento debemos aclarar que cada procedimiento de esta clase tiene un homólogo en las entidades correspondientes a los GEDEEs, y son estos

procedimientos de entidades GEDEEs los que realizan la gestión de datos sobre el contenedor de datos $datosSeguridadProcesos de esta clase, por lo que se necesita tener conocimiento de esta función del GEDEE para pasar los parámetros correctamente y conocer qué gestión se hace realmente sobre el contenedor de datos $datosSeguridadProcesos de esta clase.

public function iniciarDatosSeguridadProceso()
este procedimiento inicia los datos de seguridad de un proceso, este procedimiento no tiene parámetros visible en la definición del procedimiento porque utiliza listas variables de parámetros, por lo que se debe tener en cuenta el orden de estos para un correcto desempeño del procedimiento, el orden de los parámetros se corresponde con la función que tiene el mismo nombre que esta, en la entidad del GEDEE al que pertenece la gestión en curso.

public function existenciaIdProceso()
este procedimiento chequea la existencia del proceso y sus datos en el contenedor $datosSeguridadProcesos de esta clase, este procedimiento no tiene parámetros visible en la definición del procedimiento porque utiliza listas variables de parámetros, por lo que se debe tener en cuenta el orden de estos para un correcto desempeño del procedimiento, el orden de los parámetros se corresponde con la función que tiene el mismo nombre que esta, en la entidad del GEDEE al que pertenece la gestión en curso.

public function accederDatosSeguridadProceso()
este procedimiento es la opción para acceder a los datos de seguridad del proceso en el contenedor $datosSeguridadProcesos de esta clase, este procedimiento no tiene parámetros visible en la definición del procedimiento porque utiliza listas variables de parámetros, por lo que se debe tener en cuenta el orden de estos para un correcto desempeño del procedimiento, el orden de los parámetros se corresponde con la función que tiene el mismo nombre que esta, en la entidad del GEDEE al que pertenece la gestión en curso.

public function clienteEjecucionProceso()
este procedimiento es la opción para chequear o saber si un proceso tiene permisos para ejecutar otro proceso, el sistema chequea los datos de seguridad en el contenedor $datosSeguridadProcesos de esta clase y retorna un resultado, este procedimiento no tiene parámetros visible en la definición del procedimiento porque utiliza listas variables de parámetros, por lo que se debe tener en cuenta el orden de estos para un correcto desempeño del procedimiento, el orden de los parámetros se corresponde con la función que tiene el mismo nombre que esta, en la entidad del GEDEE al que pertenece la gestión en curso.

public function clienteAccesoDatosProceso()
este procedimiento es la opción para chequear o saber si un proceso tiene permisos para acceder a datos contenidos en el objeto EEoDatos de otro proceso, el sistema chequea los datos de seguridad en el contenedor $datosSeguridadProcesos de esta clase y retorna un resultado, este procedimiento no tiene parámetros visible en la definición del procedimiento porque utiliza listas variables de

parámetros, por lo que se debe tener en cuenta el orden de estos para un correcto desempeño del procedimiento, el orden de los parámetros se corresponde con la función que tiene el mismo nombre que esta, en la entidad del GEDEE al que pertenece la gestión en curso.

public function clienteAccesoConfiguracionProceso()
este procedimiento es la opción para chequear o saber si un proceso tiene permisos para acceder a datos contenidos en el objeto EEoConfiguracion de otro proceso, el sistema chequea los datos de seguridad en el contenedor $datosSeguridadProcesos de esta clase y retorna un resultado, este procedimiento no tiene parámetros visible en la definición del procedimiento porque utiliza listas variables de parámetros, por lo que se debe tener en cuenta el orden de estos para un correcto desempeño del procedimiento, el orden de los parámetros se corresponde con la función que tiene el mismo nombre que esta, en la entidad del dominio tipo proceso al que pertenece la gestión en curso.

class_edatos_procesos.php

Este fichero es la clase que conforma el objeto EEoDatos, y que junto a EEoConfiguracion, y EEoSeguridad son los contenedores de datos correspondientes con sus nombres y los procedimientos para gestionar estos, es decir es el contenedor de datos de los procesos, y las herramientas para gestionarlos, cualquier dato que el proceso estime poner en este contenedor, ya sea para el trabajo del propio proceso o para dejar estos datos al servicio de otros procesos por ejecutarse aun cuando el proceso propietario de estos datos haya finalizado, la estructura de los datos depende del GEDEE (si es que este regula alguna parte de la estructura) y/o del , proceso propietario, en caso de que el GEDEE controle parte de la estructura esta estructura hay que conocerla pero su gestión es transparente para el programador del proceso, a continuación explicamos la estructura y contenido de esta clase:

Namespace: Emod\Nucleo
Nombre de la clase: Datos_procesos
Herencia: extends \Emod\Nucleo\NucleoEntidadBase

Propiedades:

protected $EEoNucleo = null ;
puntero al objeto EEoNucleo ;

protected $EEoSeguridad = null ;
puntero al objeto EEoSeguridad ;

private $datos_salida_procesos = array () ;
valores de datos que cada proceso guarda en la instancia de esta clase. Su valor primero y gestión de cambios se realizan a través de procedimiento de esta clase, la estructura de estos datos de

seguridad no tiene que ser uniforme ya que pueden coexistir en esta propiedad diferentes estructuras de datos de seguridad, tantos como procesos determinen poner sus datos de seguridad aquí, lo que si se mantiene uniforme es que cada proceso tendrá sus datos en el elemento id_ proceso que a su vez se encontrará en el elemento [namespace][clase] del GEDEE al que pertenece el proceso, a continuación un ejemplo para un proceso de id o nombre mi_proceso y perteneciente al GEDEE de namespace \Emod\Nucleo\Gedees y clase GedeeEComun:

$datos_salida_procesos [‘\Emod\Nucleo\Gedees’] GedeeEComun’] [‘ mi_proceso’] = datos

Procedimientos:

Antes de entrar en la explicación de cada procedimiento debemos aclarar que cada procedimiento de esta clase tiene un homólogo en las entidades correspondientes a los GEDEEs, y son estos procedimientos de entidades GEDEEs los que realizan la gestión de datos sobre el contenedor de datos $datos_salida_procesos de esta clase, por lo que se necesita tener conocimiento de esta función del GEDEE para pasar los parámetros correctamente y conocer qué gestión se hace realmente sobre el contenedor de datos $datos_salida_procesos de esta clase.

public function iniciarDatosSalidaProceso()
este procedimiento inicia los datos pertinentes de un proceso, este procedimiento no tiene parámetros visible en la definición del procedimiento porque utiliza listas variables de parámetros, por lo que se debe tener en cuenta el orden de estos para un correcto desempeño del procedimiento, el orden de los parámetros se corresponde con la función que tiene el mismo nombre que esta, en la entidad del GEDEE al que pertenece la gestión en curso.

public function existenciaIdProceso()
este procedimiento chequea la existencia del proceso y sus datos en el contenedor $datos_salida_procesos de esta clase, este procedimiento no tiene parámetros visible en la definición del procedimiento porque utiliza listas variables de parámetros, por lo que se debe tener en cuenta el orden de estos para un correcto desempeño del procedimiento, el orden de los parámetros se corresponde con la función que tiene el mismo nombre que esta, en la entidad del GEDEE al que pertenece la gestión en curso.

public function accederDatosSalidaProceso ()
este procedimiento es la opción para acceder a los datos pertinentes del proceso en el contenedor $datos_salida_procesos de esta clase, este procedimiento no tiene parámetros visible en la definición del procedimiento porque utiliza listas variables de parámetros, por lo que se debe tener en cuenta el orden de estos para un correcto desempeño del procedimiento, el orden de los parámetros se corresponde con la función que tiene el mismo nombre que esta, en la entidad del GEDEE al que pertenece la gestión en curso.

class_eimplementacion_procesos.php

Este fichero es la clase que conforma el objeto EEoImplementacionProcesos, que se encarga de la ejecución de procesos en el sistema Sockeletom, es decir es el implementador de los procesos que se quieren ejecutar en el ambiente Sockeletom, ya sea desde una lista de procesos situada en la configuración del sistema Sockeletom o desde el interior de un proceso ya en ejecución, este implementador de procesos tiene una estrecha relación con el objeto EEoNucleo ya que se rige por ciertos parámetros de seguridad y control de un grupo de variables que el núcleo del Sockeletom obtiene de la configuración del sistema Sockeletom, a continuación explicamos la estructura y contenido de esta clase:

Namespace: Emod\Nucleo
Nombre de la clase: ImplementacionProcesos
Herencia: extends \Emod\Nucleo\NucleoEntidadBase

Propiedades:

protected $EEoNucleo = null ;
puntero al objeto EEoNucleo ;

protected $EEoConfiguracion = null ;
puntero al objeto EEoConfiguracion ;

protected $EEoSeguridad = null ;
puntero al objeto EEoSeguridad ;

protected $EEoDatos = null ;
puntero al objeto EEoDatos ;

protected $EEoInterfazDatos = null ;
puntero al objeto EEoInterfazDatos ;

Procedimientos:

final private function controlEntidadesNucleo($La_entidades = null)
este procedimiento hace un chequeo de las propiedades $EEoNucleo, $EEoConfiguracion, $EEoSeguridad , $EEoDatos y $EEoInterfazDatos de esta clase, el chequeo se hace comparando los valores de estas propiedades con los valores que deben tener realmente y que son los alojados en la clase \Emod\Nucleo\CropNucleo, este chequeo es necesario debido a que los procesos se ejecutan como código de esta clase y pueden, por error o intencionalmente, modificar los valores de las propiedades $EEoNucleo, $EEoConfiguracion, $EEoSeguridad , $EEoDatos y

$EEoInterfazDatos de esta clase, una modificación de estas propiedades traería como consecuencia el caos para el funcionamiento del sistema Sockeletom y sus procedimientos clientes.

$La_entidades tiene como valor por defecto null, puede tener valor null o un arreglo lineal no asociativo con cualquier combinación de elementos entre los que figuren los valores de tipo string: EEoNucleo, EEoConfiguracion, EEoSeguridad , EEoDatos y EEoInterfazDatos. Si el valor es null se chequearan todas las entidades, si el valor es una combinación de los strings antes mencionados entonces se chequearán solo las entidades correspondientes a los strings; en caso de no tener uno de estos valores declarados anteriormente, el procedimiento emite un error fatal.

final private function controlEjecucionProcesos ($id_proceso , $Ls_clase_gedee_proceso , $Ls_namespace_gedee_proceso)

este procedimiento controla si un proceso tiene permisos para ejecutar otro proceso o a si mismo según su configuración, para ello hace uso de un procedimiento del objeto EEoSeguridad.

$id_proceso (string) es el identificador del proceso al que se quiere chequear si es posible ejecutar por el proceso actual (en ejecución) si la gestión es satisfactoria el procedimiento devuelve el valor string 'ejecucion'(valor que corresponde al objeto EEoSeguridad), de no ser satisfactoria la gestión el procedimiento devuelve el valor NULL.

si el $id_proceso no está inicializado en el objeto $EEoSeguridad entonces por defecto se permitirá la ejecución, de lo contrario se rige por los datos de seguridad de este proceso para ser ejecutado o auto ejecutarse.

$Ls_clase_gedee_proceso (string) es la clase a que pertenece el GEDEE del proceso que se quiere ejecutar, este parámetro admite el id o nombre de la clase, o en caso de que la clase del GEDEE del proceso a ejecutar sea el mismo que la clase del GEDEE del proceso en ejecución, se puede poner el valor ´hereda´ a este parámetro.

$Ls_namespace_gedee_proceso (string) es el namespace a que pertenece el GEDEE del proceso que se quiere ejecutar, este parámetro admite el id o nombre del namespace, o en caso de que el namespace del GEDEE del proceso a ejecutar sea el mismo que el namespace del GEDEE del proceso en ejecución, se puede poner el valor ´hereda´ a este parámetro.

final public function cargaControlProcesos ($Ls_id_proceso , $Ls_id_gedee_proceso , $Ls_clase_gedee_proceso , $Ls_namespace_gedee_proceso, $La_entrada_datos_proceso)

este procedimiento gestiona y ejecuta el fichero control de los procesos, es el encargado de gestionar según los parámetros de esta función el fichero que representa el arranque o comienzo de ejecución de un proceso, y luego de su gestión lo ejecuta en dependencia de la obligatoriedad de ejecución, a continuación explicamos los parámetros necesarios para el correcto desempeño de este procedimiento

$Ls_id_proceso es el identificador del proceso que se quiere ejecutar.

$Ls_id_gedee_proceso (string) es el identificador de la entidad clase a que pertenece el GEDEE del proceso que se quiere ejecutar, este parámetro admite el id o nombre de la entidad clase, o en caso de que el id de la entidad clase del GEDEE del proceso a ejecutar sea el mismo que el id de la entidad clase del GEDEE del proceso en ejecución, se puede poner el valor ´hereda´ a este parámetro.

$Ls_clase_gedee_proceso (string) es la clase a que pertenece el GEDEE del proceso que se quiere ejecutar, este parámetro admite el id o nombre de la clase, o en caso de que la clase del GEDEE del proceso a ejecutar sea el mismo que la clase del GEDEE del proceso en ejecución, se puede poner el valor ´hereda´ a este parámetro.

$Ls_namespace_gedee_proceso (string) es el namespace a que pertenece el GEDEE del proceso que se quiere ejecutar, este parámetro admite el id o nombre del namespace, o en caso de que el namespace del GEDEE del proceso a ejecutar sea el mismo que el namespace del GEDEE del proceso en ejecución, se puede poner el valor ´hereda´ a este parámetro.

$La_entrada_datos_proceso es una matriz con todos los datos que el proceso necesita para su correcto desempeño:

$La_entrada_datos_proceso['Ls_path_control_script'] es el path hasta el script control que se quiere incluir, tener muy presente que este path tendrá como raíz el directorio esquelemod/e_modulos/Raiz del directorio de procesos(declarado en pathRaizProcesos en la configuración del sistema Sockeletom) de la estructura de directorios del sistema Sockeletom.

$La_entrada_datos_proceso ['Li_obligatoriedad'] es la obligatoriedad de ejecución del proceso ej 4-include, 3-include_once, 2-require, 1-require_once, 5-eval; cada forma con los riesgos y especificidades que implican estas construcciones del lenguaje, por defecto se ejecutará la 4

Aparte de estos elementos obligatorios del arreglo La_entrada_datos_proceso, se pueden incluir otros para el desempeño interno del propio proceso.

class_gedee_epadre.php

Este fichero contiene la clase abstracta GedeeEPadre, de métodos abstractos, que se encarga de la definición de un grupo de procedimientos de obligatoria existencia en los GEDEEs a implementar en el sistema Sockeletom, a continuación explicamos su estructura y contenido:

Namespace: Emod\Nucleo\Gedees

Nombre de la clase: GedeeEPadre
Herencia:

Propiedades:

Procedimientos:

abstract public function existenciaIdProceso(&$fuente_datos_procesos , $id_proceso = 'hereda') ;
este procedimiento chequea la existencia de un proceso y sus datos en el contenedor &$fuente_datos_procesos. A continuación la explicación de parámetros:

&$fuente_datos_procesos: (array) es la estructura de datos sobre la que este procedimiento hará su gestión, esta estructura se pasa al procedimiento por referencia, por lo que se debe tener conciencia de lo que esto implica. Este parámetro lo inyecta o brinda el motor del sistema en dependencia de la entidad esquelemod sobre la que se gestiona.

$id_proceso: (string) es el identificador o nombre del proceso a gestionar su existencia en &$fuente_datos_procesos. El valor por defecto 'hereda' debe hacer que este procedimiento tome como valor del $id_proceso el proceso en ejecución en ese momento, se puede implementar a través de la entidad esquelemod y su un procedimiento $EEoNucleo->idProcesoEjecucion() .

A continuación de estos parámetros pueden ponerse otros que estime conveniente la gestión del procedimiento que hereda de esta abstracción.

Este procedimiento debe retornar true si existe el proceso, de lo contrario retornar null.

abstract public function iniciarDatosConfiguracionProceso(&$fuente_datos_procesos) ;
este procedimiento inicia los datos de configuración de un proceso, es decir copia por primera ves los datos de configuración de un proceso en la estructura de datos (array) &$fuente_datos_procesos, estos datos de configuración pueden tener una estructura específica para cada GEDEE, ya que esta característica de especificación y especialización es el echo de existir de los GEDEEs. A continuación la explicación de parámetros:

&$fuente_datos_procesos: (array) es la estructura de datos sobre la que este procedimiento hará su gestión, esta estructura se pasa al procedimiento por referencia, por lo que se debe tener conciencia de lo que esto implica. Este parámetro lo inyecta o brinda el motor del sistema tomando la estructura de datos de la entidad esquelemod EEoConfiguracion.

A continuación de este parámetro pueden ponerse otros que estime conveniente la gestión del procedimiento que hereda de esta abstracción.

Este procedimiento debe retornar true si su gestión fue exitosa , de lo contrario retornar null.

abstract public function accederDatosConfiguracionProceso(&$fuente_datos_procesos , $id_proceso = 'hereda') ;
este procedimiento es la opción para acceder a los datos de configuración de un proceso en la estructura de datos (array) &$fuente_datos_procesos, es decir leer, escribir o modificar, y eliminar, datos en la estructura de datos(array) &$fuente_datos_procesos. Estos datos de configuración pueden tener una estructura específica al igual que la forma de gestión de este procedimiento puede ser específica, para cada GEDEE, ya que esta característica de especificación y especialización es el echo de existir de los GEDEEs. A continuación la explicación de parámetros:

&$fuente_datos_procesos: (array) es la estructura de datos sobre la que este procedimiento hará su gestión, esta estructura se pasa al procedimiento por referencia, por lo que se debe tener conciencia de lo que esto implica. Este parámetro lo inyecta o brinda el motor del sistema tomando la estructura de datos de la entidad esquelemod EEoConfiguracion.

$id_proceso: (string) es el identificador o nombre del proceso al que se accederán sus datos de configuración en &$fuente_datos_procesos. El valor por defecto 'hereda' debe hacer que este procedimiento tome como valor del $id_proceso el proceso en ejecución en ese momento, se puede implementar a través de la entidad esquelemod y su un procedimiento $EEoNucleo->idProcesoEjecucion() .

A continuación de estos parámetros pueden ponerse otros que estime conveniente la gestión del procedimiento que hereda de esta abstracción.

Este procedimiento debe retornar true si su gestión fue exitosa para modificar o eliminar, retornar la estructura de datos (array) a leer si su gestión fue exitosa para leer, de lo contrario retornar null.

abstract public function iniciarDatosSeguridadProceso(&$fuente_datos_procesos) ;
este procedimiento inicia los datos de seguridad de un proceso, es decir copia por primera ves los datos de seguridad de un proceso en la estructura de datos (array) &$fuente_datos_procesos, estos datos de seguridad pueden tener una estructura específica para cada GEDEE, ya que esta característica de especificación y especialización es el echo de existir de los GEDEEs. A continuación la explicación de parámetros:

&$fuente_datos_procesos: (array) es la estructura de datos sobre la que este procedimiento hará su gestión, esta estructura se pasa al procedimiento por referencia, por lo que se debe tener conciencia de lo que esto implica. Este parámetro lo inyecta o brinda el motor del sistema tomando la estructura de datos de la entidad esquelemod EEoSeguridad.

A continuación de este parámetro pueden ponerse otros que estime conveniente la gestión del procedimiento que hereda de esta abstracción.

Este procedimiento debe retornar true si su gestión fue exitosa , de lo contrario retornar null.

abstract public function accederDatosSeguridadProceso(&$fuente_datos_procesos , $id_proceso = 'hereda') ;
este procedimiento es la opción para acceder a los datos de seguridad de un proceso en la estructura de datos (array) &$fuente_datos_procesos, es decir leer, escribir o modificar, y eliminar, datos en la estructura de datos(array) &$fuente_datos_procesos. Estos datos de seguridad pueden tener una estructura específica al igual que la forma de gestión de este procedimiento puede ser específica, para cada GEDEE, ya que esta característica de especificación y especialización es el echo de existir de los GEDEEs. A continuación la explicación de parámetros:

&$fuente_datos_procesos: (array) es la estructura de datos sobre la que este procedimiento hará su gestión, esta estructura se pasa al procedimiento por referencia, por lo que se debe tener conciencia de lo que esto implica. Este parámetro lo inyecta o brinda el motor del sistema tomando la estructura de datos de la entidad esquelemod EEoSeguridad.

$id_proceso: (string) es el identificador o nombre del proceso al que se accederán sus datos de seguridad en &$fuente_datos_procesos. El valor por defecto 'hereda' debe hacer que este procedimiento tome como valor del $id_proceso el proceso en ejecución en ese momento, se puede implementar a través de la entidad esquelemod y su un procedimiento $EEoNucleo->idProcesoEjecucion() .

A continuación de estos parámetros pueden ponerse otros que estime conveniente la gestión del procedimiento que hereda de esta abstracción.

Este procedimiento debe retornar true si su gestión fue exitosa para modificar o eliminar, retornar la estructura de datos (array) a leer si su gestión fue exitosa para leer, de lo contrario retornar null.

abstract public function clienteEjecucionProceso(&$fuente_datos_procesos , $id_proceso = 'hereda') ;
este procedimiento es la opción para chequear o saber si un proceso tiene permisos para ejecutar otro proceso, el sistema chequea los datos de seguridad en la estructura de datos (array) &$fuente_datos_procesos. Estos datos de seguridad pueden tener una estructura específica al igual que la forma de gestión de este procedimiento puede ser específica, para cada GEDEE, ya que esta característica de especificación y especialización es el echo de existir de los GEDEEs. A continuación la explicación de parámetros:

&$fuente_datos_procesos: (array) es la estructura de datos sobre la que este procedimiento hará su gestión, esta estructura se pasa al procedimiento por referencia, por lo que se debe tener conciencia de lo que esto implica. Este parámetro lo inyecta o brinda el motor del sistema tomando la estructura de datos de la entidad esquelemod EEoSeguridad.

$id_proceso: (string) es el identificador o nombre del proceso al que se chequea si permite que el proceso actual (en ejecución) lo ejecute, este chequeo se hace en sus datos de seguridad en

&$fuente_datos_procesos. El valor por defecto 'hereda' debe hacer que este procedimiento tome como valor del $id_proceso el proceso en ejecución en ese momento, se puede implementar a través de la entidad esquelemod y su un procedimiento $EEoNucleo->idProcesoEjecucion() .

A continuación de estos parámetros pueden ponerse otros que estime conveniente la gestión del procedimiento que hereda de esta abstracción.

Este procedimiento si el proceso actual tiene permiso para ejecutar el proceso $id_proceso entonces debe retornar el (string) 'ejecucion'. De lo contrario retornar null.

abstract public function clienteAccesoDatosProceso(&$fuente_datos_procesos , $id_proceso = 'hereda') ;
este procedimiento es la opción para chequear o saber si un proceso tiene permisos para acceder datos de otro proceso en la entidad EEoDatos, el sistema chequea los datos de seguridad en la estructura de datos (array) &$fuente_datos_procesos. Estos datos de seguridad pueden tener una estructura específica al igual que la forma de gestión de este procedimiento puede ser específica, para cada GEDEE, ya que esta característica de especificación y especialización es el echo de existir de los GEDEEs. A continuación la explicación de parámetros:

&$fuente_datos_procesos: (array) es la estructura de datos sobre la que este procedimiento hará su gestión, esta estructura se pasa al procedimiento por referencia, por lo que se debe tener conciencia de lo que esto implica. Este parámetro lo inyecta o brinda el motor del sistema tomando la estructura de datos de la entidad esquelemod EEoSeguridad.

$id_proceso: (string) es el identificador o nombre del proceso al que se chequea si permite que el proceso actual (en ejecución) acceda a sus datos($id_proceso) en la entidad EEoDatos, este chequeo se hace en sus datos de seguridad en &$fuente_datos_procesos. El valor por defecto 'hereda' debe hacer que este procedimiento tome como valor del $id_proceso el proceso en ejecución en ese momento, se puede implementar a través de la entidad esquelemod y su un procedimiento $EEoNucleo->idProcesoEjecucion() .

A continuación de estos parámetros pueden ponerse otros que estime conveniente la gestión del procedimiento que hereda de esta abstracción.

Este procedimiento si su gestión fue exitosa debe retornar la combinación de permisos para el acceso a datos (string) separados por el caracter repetido '::' ejemplo: 'leer::modificar::eliminar' o 'leer::eliminar' etc según corresponda a los permisos. De lo contrario retornar null.

abstract public function clienteAccesoConfiguracionProceso(&$fuente_datos_procesos , $id_proceso = 'hereda') ;
este procedimiento es la opción para chequear o saber si un proceso tiene permisos para acceder datos de otro proceso en la entidad EEoConfiguracion, el sistema chequea los datos de seguridad en la estructura de datos (array) &$fuente_datos_procesos. Estos datos de seguridad pueden tener

una estructura específica al igual que la forma de gestión de este procedimiento puede ser específica, para cada GEDEE, ya que esta característica de especificación y especialización es el echo de existir de los GEDEEs. A continuación la explicación de parámetros:

&$fuente_datos_procesos: (array) es la estructura de datos sobre la que este procedimiento hará su gestión, esta estructura se pasa al procedimiento por referencia, por lo que se debe tener conciencia de lo que esto implica. Este parámetro lo inyecta o brinda el motor del sistema tomando la estructura de datos de la entidad esquelemod EEoSeguridad.

$id_proceso: (string) es el identificador o nombre del proceso al que se chequea si permite que el proceso actual (en ejecución) acceda a sus datos($id_proceso) en la entidad EEoConfiguracion, este chequeo se hace en sus datos de seguridad en &$fuente_datos_procesos. El valor por defecto 'hereda' debe hacer que este procedimiento tome como valor del $id_proceso el proceso en ejecución en ese momento, se puede implementar a través de la entidad esquelemod y su un procedimiento $EEoNucleo->idProcesoEjecucion() .

A continuación de estos parámetros pueden ponerse otros que estime conveniente la gestión del procedimiento que hereda de esta abstracción.

Este procedimiento si su gestión fue exitosa debe retornar la combinación de permisos para el acceso a datos (string) separados por el caracter repetido '::' ejemplo: 'leer::modificar::eliminar' o 'leer::eliminar' etc según corresponda a los permisos. De lo contrario retornar null.

abstract public function iniciarDatosSalidaProceso(&$fuente_datos_procesos) ;
este procedimiento inicia los datos de salida de un proceso, es decir copia por primera ves los datos de salida de un proceso en la estructura de datos (array) &$fuente_datos_procesos, estos datos de salida pueden tener una estructura específica para cada GEDEE, ya que esta característica de especificación y especialización es el echo de existir de los GEDEEs. A continuación la explicación de parámetros:

&$fuente_datos_procesos: (array) es la estructura de datos sobre la que este procedimiento hará su gestión, esta estructura se pasa al procedimiento por referencia, por lo que se debe tener conciencia de lo que esto implica. Este parámetro lo inyecta o brinda el motor del sistema tomando la estructura de datos de la entidad esquelemod EEoDatos.

A continuación de este parámetro pueden ponerse otros que estime conveniente la gestión del procedimiento que hereda de esta abstracción.

Este procedimiento debe retornar true si su gestión fue exitosa , de lo contrario retornar null.

abstract public function accederDatosSalidaProceso(&$fuente_datos_procesos , $id_proceso = 'hereda') ;

este procedimiento es la opción para acceder a los datos de salida de un proceso en la estructura de datos (array) &$fuente_datos_procesos, es decir leer, escribir o modificar, y eliminar, datos en la estructura de datos(array) &$fuente_datos_procesos. Estos datos de salida pueden tener una estructura específica al igual que la forma de gestión de este procedimiento puede ser específica, para cada GEDEE, ya que esta característica de especificación y especialización es el echo de existir de los GEDEEs. A continuación la explicación de parámetros:

&$fuente_datos_procesos: (array) es la estructura de datos sobre la que este procedimiento hará su gestión, esta estructura se pasa al procedimiento por referencia, por lo que se debe tener conciencia de lo que esto implica. Este parámetro lo inyecta o brinda el motor del sistema tomando la estructura de datos de la entidad esquelemod EEoDatos.

$id_proceso: (string) es el identificador o nombre del proceso al que se accederán sus datos de salida en &$fuente_datos_procesos. El valor por defecto 'hereda' debe hacer que este procedimiento tome como valor del $id_proceso el proceso en ejecución en ese momento, se puede implementar a través de la entidad esquelemod y su un procedimiento $EEoNucleo->idProcesoEjecucion() .

A continuación de estos parámetros pueden ponerse otros que estime conveniente la gestión del procedimiento que hereda de esta abstracción.

Este procedimiento debe retornar true si su gestión fue exitosa para modificar o eliminar, retornar la estructura de datos (array) a leer si su gestión fue exitosa para leer, de lo contrario retornar null.

class_gedee_enucleo

Este fichero contiene la clase de métodos estáticos GedeeENucleo, que se encarga de un tipo específico de Gestión de Estructuras de Datos en Entidades Esquelemod, GEDEE creado para los procesos tipo núcleo a ejecutarse en al sistema Sockeletom, a continuación explicamos su estructura y contenido:

Namespace: Emod\Nucleo\Gedees
Nombre de la clase: GedeeENucleo
Herencia:

Propiedades:

static private $EEoNucleo = null ;
puntero al objeto EEoNucleo ;

static private $EEoSeguridad = null ;
puntero al objeto $EEoSeguridad ;

static private $classIniciacion = null ;
es un boolean para chequear la iniciación de esta clase

Procedimientos:

static public function iniciarParametrosBase()
es el procedimiento encargado de iniciar elementos base para el funcionamiento correcto de esta entidad. Los parámetros se explican a continuación:

este procedimiento retorna true si su gestión fue exitosa, de lo contrario retorna null.

static private function chequearSeguridadCliente()
es el procedimiento encargado de chequear el proceso cliente que hace los pedidos de gestión a este GEDEE, este procedimiento está implementado para que verifique si el namespace y clase del GEDEE del proceso cliente (que a su ves es el proceso actual, ejecutándose) es igual al namespace y clase del GEDEE del proceso núcleo, si coinciden los pedidos de gestión a este GEDEE serán concedidos, si no coinciden serán denegados.

este procedimiento retorna true si su gestión fue exitosa, de lo contrario retorna null.

static public function existenciaIdProceso(&$fuente_datos_procesos , $id_proceso = 'hereda')
es el procedimiento encargado de responder si existe un proceso determinado en la estructura de datos correspondiente. Los parámetros se explican a continuación:

$fuente_datos_procesos (array), es la fuente de datos sobre la que este procedimiento hará su gestión, es la estructura de datos donde se buscará la existencia de un id_proceso.

$id_proceso (string), es el identificador o nombre del proceso a buscar su existencia en la estructura de datos $fuente_datos_procesos, admite el valor 'hereda' que quiere decir que el id_proceso es el del proceso que está en ejecución en el momento de la llamada a este procedimiento.

este procedimiento retorna true si existe el id_proceso en la estructura $fuente_datos_procesos, de lo contrario retorna null.

static public function iniciarDatosProceso(&$fuente_datos_procesos , $datos)
es el procedimiento encargado de iniciar una estructura de datos determinada, teniendo como propietario el proceso en ejecución en el momento que se hace la llamada a este procedimiento, en la estructura de datos correspondiente. Los parámetros se explican a continuación:

$fuente_datos_procesos (array), es la fuente de datos sobre la que este procedimiento hará su gestión, es la estructura de datos donde se creará una estructura de datos teniendo como propietario el proceso en ejecución en el momento que se hace la llamada a este procedimiento.

$datos (array), son los datos a iniciar en la $fuente_datos_procesos.

este procedimiento retorna true si su gestión fue exitosa, de lo contrario retorna null.

static public function iniciarDatosConfiguracionProceso(&$fuente_datos_procesos , $datos_configuracion)
es el procedimiento encargado de iniciar una estructura de datos determinada, teniendo como propietario el proceso en ejecución en el momento que se hace la llamada a este procedimiento, en la estructura de datos correspondiente a la entidad EEo_Configuracion. Los parámetros se explican a continuación:

$fuente_datos_procesos (array), es la fuente de datos sobre la que este procedimiento hará su gestión, es la estructura de datos donde se creará una estructura de datos teniendo como propietario el proceso en ejecución en el momento que se hace la llamada a este procedimiento. Esta estructura de datos es la correspondiente a la entidad EEo_Configuracion

$datos_configuracion (array), son los datos a iniciar en la $fuente_datos_procesos correspondiente a la entidad EEo_Configuracion.

este procedimiento retorna true si su gestión fue exitosa, de lo contrario retorna null.

static public function accederDatosConfiguracionProceso(&$fuente_datos_procesos , $id_proceso= 'hereda' , $estructura_acceder = array( ) , $tipo_modificacion = 1 , $condicion_modificacion = 0)
es el procedimiento encargado de acceder a la estructura de datos correspondiente a la entidad EEo_Configuracion sobre la que se realizarán las acciones de leer, escribir y/o eliminar datos. Es la vía de acceso a datos en EEo_Configuracion para procesos que quieran acceder a datos propios o de otros procesos. Para ello deberán cumplirse un grupo de condiciones entre las que se encuentran: - existir una sección de datos del proceso al que se quieren acceder los datos en la entidad EEo_Configuracion. Si el proceso intenta acceder a sus propios datos también tiene que haber iniciado datos en la entidad EEo_Configuracion, de lo contrario este procedimiento no hará su gestión. Para iniciar datos en la entidad EEo_Configuracion existe un procedimiento.

Los parámetros se explican a continuación:

$fuente_datos_procesos (array), es la fuente de datos sobre la que este procedimiento hará su gestión, es la estructura de datos sobre la que se realizarán las acciones de leer, escribir y/o eliminar datos. Esta estructura de datos es la correspondiente a la entidad EEo_Configuracion.

$id_proceso (string), es el identificador o nombre del proceso con datos en $fuente_datos_procesos y sobre estos datos es que se quiere tener acceso de de leer, escribir y/o eliminar. Este parámetro

admite el valor 'hereda' que quiere decir que el id_proceso es el del proceso núcleo que está en ejecución en el momento de la llamada a este procedimiento.

$estructura_acceder (array), es la estructura a acceder en el arreglo $fuente_datos_procesos, este parámetro es dependiente del parámetro $tipo_modificacion, a continuación sus diferentes funcionalidades

-para valor 1 (leer) del parámetro $tipo_modificacion, este parámetro puede contener un arreglo vacío para obtener (leer) todos los datos de configuración correspondientes a $id_proceso en la entidad EEo_Configuracion, array() un arreglo vacío es por defecto el valor que tiene este parámetro, también se puede acceder solo a una parte de la estructura de datos de configuración si se le da el valor a este parámetro de una cadena con la estructura a la que se quiere acceder, ejemplo '[dato1][dato2]' y el devolverá el elemento correspondiente a los datos de configuración del proceso $id_proceso en la estructura [dato1][dato2].

-para valores 2 (escribir) y 3 (eliminar) del parámetro $tipo_modificacion este parámetro puede contener un arreglo asociativo con los elementos que se quieren eliminar o modificar en el arreglo de datos de configuración correspondientes a $id_proceso en la entidad EEo_Configuracion, este arreglo contendrá la estructura exacta en la que se encuentra el elemento que se quiere escribir o eliminar, es como un mapa, una imagen de la matriz y su rama o elemento que se quiere escribir o eliminar, llamemosle de ahora en adelante "estructura imagen", en caso de que toda una rama del arreglo se quiera eliminar basta con poner 'elemento'=array() y se eliminará ese elemento junto a la rama que el constituye.

$tipo_modificacion (integer), es el tipo de modificación que se utilizará, sus posibles valores son los siguientes, 1 para leer 2 para escribir o modificar , 3 para eliminar, si su valor es diferente de los valores permitidos este procedimiento retorna el valor null, este parámetro también debe coincidir con los permisos de escribir o eliminar que debe tener el proceso actual (en ejecución) sobre los datos a modificar o eliminar del proceso $id_proceso.

$condicion_modificacion (integer)(string), es la condición para llevar a cabo una modificación, ya sea escribir o eliminar, este parámetro no se toma en cuenta cuando $tipo_modificacion es 1 (leer), solo se toma en cuenta para $tipo_modificacion con valor 2 o 3. La diferencia en el tipo (integer) (string) está dada porque en realidad se convierte el entero en binario y el string se leerá como una representación binaria en string ej '100110111'. Cada lugar del binario representa un tipo de condición a cumplirse para que se realice la modificación en la estructura de datos, esta condición será ejecutada tomando como elementos comparativos el arreglo $estructura_acceder y el arreglo $fuente_datos_procesos, elemento por elemento homólogos de cada uno de ellos; si se escoge una condición y esta no se cumple, el elemento base ($estructura_acceder) no es modificado A continuación los valores posibles para este parámetro.

$condicion_modificacion Para $tipo_modificacion 2 (escribir)

1er bit (bit menos significativo, estremo derecho)en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si sus valores son iguales (==)

2do bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si sus valores son diferentes (!= , <>)

3er bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si sus valores son no idénticos (!==)

4to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es menor que el de la base (<)

5to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es mayor que el de la base (>)

6to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es menor o igual que el de la base (<=)

7mo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es mayor o igual que el de la base (>=)

8vo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin citerios de comparación, pero si existe una estructura o elemento en la imagen que no existe en la base no es modificada la base, es decir no se transfiere el elemento de la imagen a la base.

9no bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a registrar en los resultados, en el elemento 'fallos' de esta función, los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores.

los valores siguientes son excepciones válidas de aclarar

binarios:

000000000 (dec 0) modifica los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación, si existe una estructura o

elemento en la imagen que no existe en la base es modificada la base con el elemento en cuestión, es decir se transfiere el elemento de la imagen a la base.

100000000 (dec 256) modifica los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación pero manteniendo el criterio de registrar en los resultados de esta función, en el elemento 'fallos' de esta función, los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores, además mantiene el comportamiento del 8vo bit según su valor.

110000000 (dec 384) modifica los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación pero manteniendo el criterio de registrar en los resultados de esta función, en el elemento 'fallos' de esta función, los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores, además mantiene el comportamiento del 8vo bit según su valor.

110000000+1 (dec mayor que 384)retorna un valor null
si la lógica de los bit del 1ro al 7mo activados equivalen a modificar todo el arreglo o existe redundancia, la función retorna un valor null
ej: 000000101 o 001010000

el 8vo y 9no bit siempre pueden estar activados, no pertenece a la lógica comparativa del lenguaje

$condicion_modificacion Para $tipo_modificacion 3 (eliminar)

si se escoge una condición y esta no se cumple, el elemento base no es eliminado.

1er bit (bit menos significativo, extremo derecho)en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son iguales (==)

2do bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son idénticos (===)

3er bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son diferentes (!= , <>)

4to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son no idénticos (!==)

5to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es menor que el de la base (<)

6to bit en cero desactivado) no se tiene en cuenta

en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es mayor que el de la base (>)

7mo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es menor o igual que el de la base (<=)

8vo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es mayor o igual que el de la base (>=)

9no bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a registrar en los resultados de esta función los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores

los valores siguientes son excepciones validas de aclarar

binarios:

000000000 (dec 0) elimina los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir elimina todo sin criterios de comparación.

100000000 (dec 256) elimina los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir elimina todo sin criterios de comparación pero manteniendo el criterio de registrar en los resultados de esta función los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores

110000000+1 (dec mayor que 384)retorna un valor null
si la lógica de los bit del 1ro al 8vo activados equivalen a eliminar todo el arreglo o existe redundancia, la función retorna un valor null
ej: 00000101 o 001010000
entre el 1er y el 8vo bit solo pueden coexistir las combinaciones 2do y 3ro, o 2do y 5to, o 2do y 6to

el 9no bit siempre puede estar activado, no pertenece a la lógica comparativa del lenguaje

este procedimiento retorna:
- para la opción de leer retorna el elemento del arreglo base a leer si la operación es satisfactoria, y null si es insatisfactoria

- para la opción de escribir y eliminar retorna true si la operación es satisfactoria, y null si es insatisfactoria

static public function iniciarDatosSeguridadProceso(&$fuente_datos_procesos , $datos_seguridad)
La seguridad de los datos de los procesos tipo núcleo que se gestionan en las Entidades Esquelemod se simplificó, se obligó a que la gestión de esos datos sea estrictamente gestionados si el proceso que los solicita es un proceso de tipo núcleo, estas obligaciones se declaran en este GEDEE, los procedimientos de seguridad de este GEDEE se mantienen existiendo solo su nombre y parámetros pero con una gestión nula, esto se debe a que como solo puede acceder a los datos de todo tipo y ejecución de un proceso núcleo, otro proceso núcleo, la seguridad no es configurable ya viene predeterminada en los demás procedimientos de este GEDEE. Los procedimientos de seguridad existen para no generar errores en caso de que los soliciten, pero siempre retornarán un valor null. Si en un futuro se decide que procesos que no son de tipo núcleo puedan leer o cambiar datos de procesos de tipo núcleo en las Entidades Esquelemod, entonces se hará el cambio pertinente en el código.

este procedimiento retorna null siempre.

static public function accederDatosSeguridadProceso(&$fuente_datos_procesos , $id_proceso = 'hereda' , $estructura_acceder = array( ) , $tipo_accion = 1 , $condicion_modificacion = 0)
La seguridad de los datos de los procesos tipo núcleo que se gestionan en las Entidades Esquelemod se simplificó, se obligó a que la gestión de esos datos sea estrictamente gestionados si el proceso que los solicita es un proceso de tipo núcleo, estas obligaciones se declaran en este GEDEE, los procedimientos de seguridad de este GEDEE se mantienen existiendo solo su nombre y parámetros pero con una gestión nula, esto se debe a que como solo puede acceder a los datos de todo tipo y ejecución de un proceso núcleo, otro proceso núcleo, la seguridad no es configurable ya viene predeterminada en los demás procedimientos de este GEDEE. Los procedimientos de seguridad existen para no generar errores en caso de que los soliciten, pero siempre retornarán un valor null.

este procedimiento retorna null siempre.

static public function clienteEjecucionProceso()
es el procedimiento encargado de gestionar si un proceso id_proceso (ejecutándose , actual) tiene permiso a ejecutar otro proceso. La seguridad de los datos de los procesos tipo núcleo que se gestionan en las Entidades Esquelemod se simplificó, se obligó a que la gestión de esos datos sea estrictamente gestionados si el proceso que los solicita es un proceso de tipo núcleo, de ser así este procedimiento retorna el string 'ejecucion', de lo contrario retorna el valor null.

static public function clienteAccesoDatosProceso(&$fuente_datos_procesos , $id_proceso = 'hereda')
es el procedimiento encargado de gestionar si un proceso id_proceso (ejecutándose , actual) tiene acceso a los datos de otro proceso que haya colocado datos en la entidad EEoDatos. La seguridad

de los datos de los procesos tipo núcleo que se gestionan en las Entidades Esquelemod se simplificó, se obligó a que la gestión de esos datos sea estrictamente gestionados si el proceso que los solicita es un proceso de tipo núcleo, de ser así este procedimiento retorna el string 'leer::escribir::eliminar', de lo contrario retorna el valor null.

static public function clienteAccesoConfiguracionProceso(&$fuente_datos_procesos , $id_proceso = 'hereda')
es el procedimiento encargado de gestionar si un proceso id_proceso (ejecutándose , actual) tiene acceso a los datos de configuración de otro proceso que haya colocado datos en la entidad EEoConfiguracion. La seguridad de los datos de los procesos tipo núcleo que se gestionan en las Entidades Esquelemod se simplificó, se obligó a que la gestión de esos datos sea estrictamente gestionados si el proceso que los solicita es un proceso de tipo núcleo, de ser así este procedimiento retorna el string 'leer::escribir::eliminar', de lo contrario retorna el valor null.

static public function iniciarDatosSalidaProceso(&$fuente_datos_procesos , $datos_salida)
es el procedimiento encargado de gestionar el inicio de datos por parte de un proceso en la entidad EEoDatos.

$fuente_datos_procesos (array), es la fuente de datos sobre la que este procedimiento hará su gestión, es la estructura de datos sobre la que se colocará el contenido del parámetro $datos_salida. Esta estructura de datos es la correspondiente a la entidad EEoDatos.

$datos_salida (array), es una estructura de datos que por voluntad quiere colocar el proceso actual (en ejecución) en la entidad EEoDatos, si su valor es vacío el procedimiento retorna el valor null.

este procedimiento retorna true si es satisfactoria su gestión y null si es insatisfactoria su gestión.

static public function accederDatosSalidaProceso(&$fuente_datos_procesos , $id_proceso = 'hereda' , $estructura_acceder = array( ) , $tipo_modificacion = 1 , $condicion_modificacion = 0)
es el procedimiento encargado de acceder a la estructura de datos correspondiente a la entidad EEoDatos sobre la que se realizarán las acciones de leer, escribir y/o eliminar datos. Es la vía de acceso a datos en EEoDatos para procesos que quieran acceder a datos propios o de otros procesos. Para ello deberán cumplirse un grupo de condiciones entre las que se encuentran:
- existir una sección de datos del proceso al que se quieren acceder los datos en la entidad EEoDatos. Si el proceso intenta acceder a sus propios datos también tiene que haber iniciado datos en la entidad EEoDatos, de lo contrario este procedimiento no hará su gestión. Para iniciar datos en la entidad EEoDatos existe un procedimiento.
- el proceso que ejecuta este procedimiento debe tener como GEDEE Uno con namespace y clase igual que el GEDEE del proceso núcleo, o lo que es lo mismo ser un proceso tipo núcleo.

Los parámetros se explican a continuación:

$fuente_datos_procesos (array), es la fuente de datos sobre la que este procedimiento hará su gestión, es la estructura de datos sobre la que se realizarán las acciones de leer, escribir y/o eliminar datos. Esta estructura de datos es la correspondiente a la entidad EEoDatos.

$id_proceso (string), es el identificador o nombre del proceso con datos en $fuente_datos_procesos y sobre estos datos es que se quiere tener acceso de leer, escribir y/o eliminar. Este parámetro admite el valor 'hereda' que quiere decir que el id_proceso es el del proceso que está en ejecución en el momento de la llamada a este procedimiento.

$estructura_acceder (array), es la estructura a acceder en el arreglo $fuente_datos_procesos, este parámetro es dependiente del parámetro $tipo_modificacion, a continuación sus diferentes funcionalidades

-para valor 1 (leer) del parámetro $tipo_modificacion, este parámetro puede contener un arreglo vacío para obtener (leer) todos los datos correspondientes a $id_proceso en la entidad EEoDatos, array() un arreglo vacío es por defecto el valor que tiene este parámetro, también se puede acceder solo a una parte de la estructura de datos si se le da el valor a este parámetro de una cadena con la estructura a la que se quiere acceder, ejemplo '[dato1][dato2]' y el devolverá el elemento correspondiente a los datos del proceso $id_proceso en la estructura [dato1][dato2].

-para valores 2 (escribir)y 3 (eliminar) del parámetro $tipo_modificacion este parámetro puede contener un arreglo asociativo con los elementos que se quieren eliminar o modificar en el arreglo de datos correspondientes a $id_proceso en la entidad EEoDatos, este arreglo contendrá la estructura exacta en la que se encuentra el elemento que se quiere escribir o eliminar, es como un mapa, una imagen de la matriz y su rama o elemento que se quiere escribir o eliminar, llamémosle de ahora en adelante "estructura imagen", en caso de que toda una rama del arreglo se quiera eliminar basta con poner 'elemento'=array() y se eliminará ese elemento junto a la rama que el constituye.

$tipo_modificacion (integer), es el tipo de modificación que se utilizará, sus posibles valores son los siguientes, 1 para leer 2 para escribir o modificar , 3 para eliminar, si su valor es diferente de los valores permitidos este procedimiento retorna el valor null, este parámetro también debe coincidir con los permisos de escribir o eliminar que debe tener el proceso actual (en ejecución) sobre los datos a modificar o eliminar del proceso $id_proceso para que el procedimiento concluya de forma satisfactoria.

$condicion_modificacion (integer)(string), es la condición para llevar a cabo una modificación, ya sea escribir o eliminar, este parámetro no se toma en cuenta cuando $tipo_modificacion es 1 (leer), solo se toma en cuenta para $tipo_modificacion con valor 2 o 3. La diferencia en el tipo (integer) (string) está dada porque en realidad se convierte el entero en binario y el string se leerá como una representación binaria en string ej '100110111'. Cada lugar del binario representa un tipo de condición a cumplirse para que se realice la modificación en la estructura de datos, esta condición será ejecutada tomando como elementos comparativos el arreglo $estructura_acceder y el arreglo

$fuente_datos_procesos, elemento por elemento homólogos de cada uno de ellos; si se escoge una condición y esta no se cumple, el elemento base ($estructura_acceder) no es modificado A continuación los valores posibles para este parámetro.

$condicion_modificacion Para $tipo_modificacion 2 (escribir)

1er bit (bit menos significativo, estremo derecho)en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si sus valores son iguales (==)

2do bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si sus valores son diferentes (!= , <>)

3er bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si sus valores son no idénticos (!==)

4to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es menor que el de la base (<)

5to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es mayor que el de la base (>)

6to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es menor o igual que el de la base (<=)

7mo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es mayor o igual que el de la base (>=)

8vo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin citerios de comparación, pero si existe una estructura o elemento en la imagen que no existe en la base no es modificada la base, es decir no se transfiere el elemento de la imagen a la base.

9no bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a registrar en los resultados, en el elemento 'fallos' de esta función, los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores.

los valores siguientes son excepciones válidas de aclarar

binarios:

000000000 (dec 0) modifica los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación, si existe una estructura o elemento en la imagen que no existe en la base es modificada la base con el elemento en cuestión, es decir se transfiere el elemento de la imagen a la base.

100000000 (dec 256) modifica los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación pero manteniendo el criterio de registrar en los resultados de esta función, en el elemento 'fallos' de esta función, los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores, además mantiene el comportamiento del 8vo bit según su valor.

110000000 (dec 384) modifica los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación pero manteniendo el criterio de registrar en los resultados de esta función, en el elemento 'fallos' de esta función, los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores, además mantiene el comportamiento del 8vo bit según su valor.

110000000+1 (dec mayor que 384)retorna un valor null
si la lógica de los bit del 1ro al 7mo activados equivalen a modificar todo el arreglo o existe redundancia, la función retorna un valor null
ej: 000000101 o 001010000

el 8vo y 9no bit siempre pueden estar activados, no pertenece a la lógica comparativa del lenguaje

$condicion_modificacion Para $tipo_modificacion 3 (eliminar)

si se escoge una condición y esta no se cumple, el elemento base no es eliminado.

1er bit (bit menos significativo, extremo derecho)en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son iguales (==)

2do bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son idénticos (===)

3er bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son diferentes (!= , <>)

4to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son no idénticos (!==)

5to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es menor que el de la base (<)

6to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es mayor que el de la base (>)

7mo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es menor o igual que el de la base (<=)

8vo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es mayor o igual que el de la base (>=)

9no bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a registrar en los resultados de esta función los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores

los valores siguientes son excepciones validas de aclarar

binarios:

000000000 (dec 0) elimina los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir elimina todo sin criterios de comparación.

100000000 (dec 256) elimina los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir elimina todo sin criterios de comparación pero manteniendo el criterio de registrar en los resultados de esta función los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores

110000000+1 (dec mayor que 384)retorna un valor null
si la lógica de los bit del 1ro al 8vo activados equivalen a eliminar todo el arreglo o existe redundancia, la función retorna un valor null
ej: 00000101 o 001010000
entre el 1er y el 8vo bit solo pueden coexistir las combinaciones 2do y 3ro, o 2do y 5to, o 2do y 6to

el 9no bit siempre puede estar activado, no pertenece a la lógica comparativa del lenguaje

este procedimiento retorna:
- para la opción de leer retorna el elemento del arreglo base a leer si la operación es satisfactoria, y null si es insatisfactoria

- para la opción de escribir y eliminar retorna true si la operación es satisfactoria, y null si es insatisfactoria

class_gedee_ecomun

Este fichero contiene la clase de métodos estáticos GedeeEComun, que se encarga de un tipo específico de Gestión de Estructuras de Datos en Entidades Esquelemod, a continuación explicamos su estructura y contenido:

Namespace: Emod\Nucleo\Gedees
Nombre de la clase: GedeeEComun
Herencia:

Propiedades:

static private $EEoNucleo = null ;
puntero al objeto EEoNucleo ;

static private $EEoSeguridad = null ;
puntero al objeto $EEoSeguridad ;

static private $namespaceEntidad = null ;
es un string que contiene el namespace de esta entidad clase

static private $claseEntidad = null ;
es un string que contiene el nombre de esta entidad clase

static private $idEntidad = null ;
es un string que contiene el nombre o identificador de esta entidad clase

static private $classIniciacion = null ;
es un boolean para chequear la iniciación de esta clase

static private $ambitoSeguridad = 'restrictivo' ;
es un string que contiene el valor del ámbito de seguridad por defecto que acataran los procedimientos de esta clase, sus valores posibles son 'restrictivo' o 'permisivo'.

Procedimientos:

static public function iniciarParametrosBase($id_entidad)

es el procedimiento encargado de iniciar elementos base para el funcionamiento correcto de esta entidad. Los parámetros se explican a continuación:

$id_entidad: (string) que contiene el nombre o identificador de esta entidad clase

este procedimiento retorna true si su gestión fue exitosa, de lo contrario retorna null.

static public function existenciaIdProceso(&$fuente_datos_procesos , $id_proceso = 'hereda')
es el procedimiento encargado de responder si existe un proceso determinado en la estructura de datos correspondiente. Los parámetros se explican a continuación:

$fuente_datos_procesos (array), es la fuente de datos sobre la que este procedimiento hará su gestión, es la estructura de datos donde se buscará la existencia de un id_proceso.

$id_proceso (string), es el identificador o nombre del proceso a buscar su existencia en la estructura de datos $fuente_datos_procesos, admite el valor 'hereda' que quiere decir que el id_proceso es el del proceso que está en ejecución en el momento de la llamada a este procedimiento.

este procedimiento retorna true si existe el id_proceso en la estructura $fuente_datos_procesos, de lo contrario retorna null.

static public function iniciarDatosProceso(&$fuente_datos_procesos , $datos)
es el procedimiento encargado de iniciar una estructura de datos determinada, teniendo como propietario el proceso en ejecución en el momento que se hace la llamada a este procedimiento, en la estructura de datos correspondiente. Los parámetros se explican a continuación:

$fuente_datos_procesos (array), es la fuente de datos sobre la que este procedimiento hará su gestión, es la estructura de datos donde se creará una estructura de datos teniendo como propietario el proceso en ejecución en el momento que se hace la llamada a este procedimiento.

$datos (array), son los datos a iniciar en la $fuente_datos_procesos.

este procedimiento retorna true si su gestión fue exitosa, de lo contrario retorna null.

static public function iniciarDatosConfiguracionProceso(&$fuente_datos_procesos , $datos_configuracion)
es el procedimiento encargado de iniciar una estructura de datos determinada, teniendo como propietario el proceso en ejecución en el momento que se hace la llamada a este procedimiento, en la estructura de datos correspondiente a la entidad EEo_Configuracion. Los parámetros se explican a continuación:

$fuente_datos_procesos (array), es la fuente de datos sobre la que este procedimiento hará su gestión, es la estructura de datos donde se creará una estructura de datos teniendo como propietario

el proceso en ejecución en el momento que se hace la llamada a este procedimiento. Esta estructura de datos es la correspondiente a la entidad EEo_Configuracion

$datos_configuracion (array), son los datos a iniciar en la $fuente_datos_procesos correspondiente a la entidad EEo_Configuracion.

este procedimiento retorna true si su gestión fue exitosa, de lo contrario retorna null.

static public function accederDatosConfiguracionProceso(&$fuente_datos_procesos , $id_proceso= 'hereda' , $estructura_acceder = array( ) , $tipo_modificacion = 1 , $condicion_modificacion = 0)
es el procedimiento encargado de acceder a la estructura de datos correspondiente a la entidad EEo_Configuracion sobre la que se realizarán las acciones de leer, escribir y/o eliminar datos. Es la vía de acceso a datos en EEo_Configuracion para procesos que quieran acceder a datos propios o de otros procesos. Para ello deberán cumplirse un grupo de condiciones entre las que se encuentran:
- existir una sección de datos del proceso al que se quieren acceder los datos en la entidad EEo_Configuracion. Si el proceso intenta acceder a sus propios datos también tiene que haber iniciado datos en la entidad EEo_Configuracion, de lo contrario este procedimiento no hará su gestión. Para iniciar datos en la entidad EEo_Configuracion existe un procedimiento.
- existir una sección de seguridad del proceso al que se quieren acceder los datos en la entidad EEoSeguridad. Solo se prescinde de esta condición si el proceso intenta acceder a sus propios datos.
- existir permiso de acceso para el proceso que ejecuta este procedimiento en la estructura de datos de seguridad del proceso $id_proceso en la entidad EEoSeguridad. Además de corresponderse la acción de acceso con los permisos otorgados. Solo se prescinde de esta condición si el proceso intenta acceder a sus propios datos.

Los parámetros se explican a continuación:

$fuente_datos_procesos (array), es la fuente de datos sobre la que este procedimiento hará su gestión, es la estructura de datos sobre la que se realizarán las acciones de leer, escribir y/o eliminar datos. Esta estructura de datos es la correspondiente a la entidad EEo_Configuracion.

$id_proceso (string), es el identificador o nombre del proceso con datos en $fuente_datos_procesos y sobre estos datos es que se quiere tener acceso de de leer, escribir y/o eliminar. Este parámetro admite el valor 'hereda' que quiere decir que el id_proceso es el del proceso que está en ejecución en el momento de la llamada a este procedimiento.

$estructura_acceder (array), es la estructura a acceder en el arreglo $fuente_datos_procesos, este parámetro es dependiente del parámetro $tipo_modificacion, a continuación sus diferentes funcionalidades

-para valor 1 (leer) del parámetro $tipo_modificacion, este parámetro puede contener un arreglo vacío para obtener (leer) todos los datos de configuración correspondientes a $id_proceso en la entidad EEo_Configuracion, array() un arreglo vacío es por defecto el valor que tiene este parámetro, también se puede acceder solo a una parte de la estructura de datos de configuración si se le da el valor a este parámetro de una cadena con la estructura a la que se quiere acceder, ejemplo '[dato1][dato2]' y el devolverá el elemento correspondiente a los datos de configuración del proceso $id_proceso en la estructura [dato1][dato2].

-para valores 2 (escribir) y 3 (eliminar) del parámetro $tipo_modificacion este parámetro puede contener un arreglo asociativo con los elementos que se quieren eliminar o modificar en el arreglo de datos de configuración correspondientes a $id_proceso en la entidad EEo_Configuracion, este arreglo contendrá la estructura exacta en la que se encuentra el elemento que se quiere escribir o eliminar, es como un mapa, una imagen de la matriz y su rama o elemento que se quiere escribir o eliminar, llamemosle de ahora en adelante "estructura imagen", en caso de que toda una rama del arreglo se quiera eliminar basta con poner 'elemento'=array() y se eliminará ese elemento junto a la rama que el constituye.

$tipo_modificacion (integer), es el tipo de modificación que se utilizará, sus posibles valores son los siguientes, 1 para leer 2 para escribir o modificar , 3 para eliminar, si su valor es diferente de los valores permitidos este procedimiento retorna el valor null, este parámetro también debe coincidir con los permisos de escribir o eliminar que debe tener el proceso actual (en ejecución) sobre los datos a modificar o eliminar del proceso $id_proceso.

$condicion_modificacion (integer)(string), es la condición para llevar a cabo una modificación, ya sea escribir o eliminar, este parámetro no se toma en cuenta cuando $tipo_modificacion es 1 (leer), solo se toma en cuenta para $tipo_modificacion con valor 2 o 3. La diferencia en el tipo (integer) (string) está dada porque en realidad se convierte el entero en binario y el string se leerá como una representación binaria en string ej '100110111'. Cada lugar del binario representa un tipo de condición a cumplirse para que se realice la modificación en la estructura de datos, esta condición será ejecutada tomando como elementos comparativos el arreglo $estructura_acceder y el arreglo $fuente_datos_procesos, elemento por elemento homólogos de cada uno de ellos; si se escoge una condición y esta no se cumple, el elemento base ($estructura_acceder) no es modificado A continuación los valores posibles para este parámetro.

$condicion_modificacion Para $tipo_modificacion 2 (escribir)

1er bit (bit menos significativo, estremo derecho)en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si sus valores son iguales (==)

2do bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si sus valores son diferentes (!= , <>)

3er bit en cero desactivado) no se tiene en cuenta

en 1 (activado)corresponde a modificar los elementos si sus valores son no idénticos (!==)

4to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es menor que el de la base (<)

5to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es mayor que el de la base (>)

6to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es menor o igual que el de la base (<=)

7mo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es mayor o igual que el de la base (>=)

8vo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin citerios de comparación, pero si existe una estructura o elemento en la imagen que no existe en la base no es modificada la base, es decir no se transfiere el elemento de la imagen a la base.

9no bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a registrar en los resultados, en el elemento 'fallos' de esta función, los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores.

los valores siguientes son excepciones válidas de aclarar

binarios:

000000000 (dec 0) modifica los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación, si existe una estructura o elemento en la imagen que no existe en la base es modificada la base con el elemento en cuestión, es decir se transfiere el elemento de la imagen a la base.

100000000 (dec 256) modifica los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación pero manteniendo el criterio de registrar en los resultados de esta función, en el elemento 'fallos' de esta función, los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores, además mantiene el comportamiento del 8vo bit según su valor.

110000000 (dec 384) modifica los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación pero manteniendo el criterio de registrar en los resultados de esta función, en el elemento 'fallos' de esta función, los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores, además mantiene el comportamiento del 8vo bit según su valor.

110000000+1 (dec mayor que 384)retorna un valor null
si la lógica de los bit del 1ro al 7mo activados equivalen a modificar todo el arreglo o existe redundancia, la función retorna un valor null
ej: 000000101 o 001010000

el 8vo y 9no bit siempre pueden estar activados, no pertenece a la lógica comparativa del lenguaje

$condicion_modificacion Para $tipo_modificacion 3 (eliminar)

si se escoge una condición y esta no se cumple, el elemento base no es eliminado.

1er bit (bit menos significativo, extremo derecho)en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son iguales (==)

2do bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son idénticos (===)

3er bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son diferentes (!= , <>)

4to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son no idénticos (!==)

5to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es menor que el de la base (<)

6to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es mayor que el de la base (>)

7mo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es menor o igual que el de la base (<=)

8vo bit en cero desactivado) no se tiene en cuenta

en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es mayor o igual que el de la base (>=)

9no bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a registrar en los resultados de esta función los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores

los valores siguientes son excepciones validas de aclarar

binarios:

000000000 (dec 0) elimina los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir elimina todo sin criterios de comparación.

100000000 (dec 256) elimina los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir elimina todo sin criterios de comparación pero manteniendo el criterio de registrar en los resultados de esta función los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores

110000000+1 (dec mayor que 384)retorna un valor null
si la lógica de los bit del 1ro al 8vo activados equivalen a eliminar todo el arreglo o existe redundancia, la función retorna un valor null
ej: 00000101 o 001010000
entre el 1er y el 8vo bit solo pueden coexistir las combinaciones 2do y 3ro, o 2do y 5to, o 2do y 6to

el 9no bit siempre puede estar activado, no pertenece a la lógica comparativa del lenguaje

este procedimiento retorna:
- para la opción de leer retorna el elemento del arreglo base a leer si la operación es satisfactoria, y null si es insatisfactoria

- para la opción de escribir y eliminar retorna true si la operación es satisfactoria, y null si es insatisfactoria

static public function iniciarDatosSeguridadProceso(&$fuente_datos_procesos , $datos_seguridad)
es el procedimiento encargado de iniciar una estructura de datos determinada, teniendo como propietario el proceso en ejecución en el momento que se hace la llamada a este procedimiento, en la estructura de datos correspondiente a la entidad EEoSeguridad. Los parámetros se explican a continuación:

$fuente_datos_procesos (array), es la fuente de datos sobre la que este procedimiento hará su gestión, es la estructura de datos donde se creará una estructura de datos teniendo como propietario

el proceso en ejecución en el momento que se hace la llamada a este procedimiento. Esta estructura de datos es la correspondiente a la entidad EEoSeguridad

$datos_seguridad (array), son los datos a iniciar en la $fuente_datos_procesos correspondiente a la entidad EEoSeguridad.

este procedimiento retorna true si su gestión fue exitosa, de lo contrario retorna null.

static public function accederDatosSeguridadProceso(&$fuente_datos_procesos , $id_proceso = 'hereda' , $estructura_acceder = array( ) , $tipo_accion = 1 , $condicion_modificacion = 0)
es el procedimiento encargado de acceder a la estructura de datos correspondiente a la entidad EEoSeguridad sobre la que se realizarán las acciones de leer, escribir y/o eliminar datos. Es la vía de acceso a datos en EEoSeguridad para procesos que quieran acceder a datos propios o de otros procesos. Para ello deberán cumplirse un grupo de condiciones entre las que se encuentran:
- existir una sección de datos del proceso al que se quieren acceder los datos en la entidad EEoSeguridad. Si el proceso intenta acceder a sus propios datos también tiene que haber iniciado datos en la entidad EEoSeguridad, de lo contrario este procedimiento no hará su gestión. Para iniciar datos en la entidad EEoSeguridad existe un procedimiento.
- existir permiso de acceso para el proceso que ejecuta este procedimiento en la estructura de datos de seguridad del proceso $id_proceso en la entidad EEoSeguridad. Además de corresponderse la acción de acceso con los permisos otorgados. Solo se prescinde de esta condición si el proceso intenta acceder a sus propios datos.

Los parámetros se explican a continuación:

$fuente_datos_procesos (array), es la fuente de datos sobre la que este procedimiento hará su gestión, es la estructura de datos sobre la que se realizarán las acciones de leer, escribir y/o eliminar datos. Esta estructura de datos es la correspondiente a la entidad EEoSeguridad.

$id_proceso (string), es el identificador o nombre del proceso con datos en $fuente_datos_procesos y sobre estos datos es que se quiere tener acceso de de leer, escribir y/o eliminar. Este parámetro admite el valor 'hereda' que quiere decir que el id_proceso es el del proceso que está en ejecución en el momento de la llamada a este procedimiento.

$estructura_acceder (array), es la estructura a acceder en el arreglo $fuente_datos_procesos, este parámetro es dependiente del parámetro $tipo_modificacion, a continuación sus diferentes funcionalidades

-para valor 1 (leer) del parámetro $tipo_modificacion, este parámetro puede contener un arreglo vacío para obtener (leer) todos los datos de seguridad correspondientes a $id_proceso en la entidad EEoSeguridad, array() un arreglo vacío es por defecto el valor que tiene este parámetro, también se puede acceder solo a una parte de la estructura de datos de seguridad si se le da el valor a este

parámetro de una cadena con la estructura a la que se quiere acceder, ejemplo '[dato1][dato2]' y el devolverá el elemento correspondiente a los datos de seguridad del proceso $id_proceso en la estructura [dato1][dato2].

-para valores 2 (escribir) y 3 (eliminar) del parámetro $tipo_modificacion este parámetro puede contener un arreglo asociativo con los elementos que se quieren eliminar o modificar en el arreglo de datos de seguridad correspondientes a $id_proceso en la entidad EEoSeguridad, este arreglo contendrá la estructura exacta en la que se encuentra el elemento que se quiere escribir o eliminar, es como un mapa, una imagen de la matriz y su rama o elemento que se quiere escribir o eliminar, llamemosle de ahora en adelante "estructura imagen", en caso de que toda una rama del arreglo se quiera eliminar basta con poner 'elemento'=array() y se eliminará ese elemento junto a la rama que el constituye.

$tipo_modificacion (integer), es el tipo de modificación que se utilizará, sus posibles valores son los siguientes, 1 para leer 2 para escribir o modificar , 3 para eliminar, si su valor es diferente de los valores permitidos este procedimiento retorna el valor null, este parámetro también debe coincidir con los permisos de escribir o eliminar que debe tener el proceso actual (en ejecución) sobre los datos a modificar o eliminar del proceso $id_proceso para que el procedimiento concluya de forma satisfactoria.

$condicion_modificacion (integer)(string), es la condición para llevar a cabo una modificación, ya sea escribir o eliminar, este parámetro no se toma en cuenta cuando $tipo_modificacion es 1 (leer), solo se toma en cuenta para $tipo_modificacion con valor 2 o 3. La diferencia en el tipo (integer) (string) está dada porque en realidad se convierte el entero en binario y el string se leerá como una representación binaria en string ej '100110111'. Cada lugar del binario representa un tipo de condición a cumplirse para que se realice la modificación en la estructura de datos, esta condición será ejecutada tomando como elementos comparativos el arreglo $estructura_acceder y el arreglo $fuente_datos_procesos, elemento por elemento homólogos de cada uno de ellos; si se escoge una condición y esta no se cumple, el elemento base ($estructura_acceder) no es modificado A continuación los valores posibles para este parámetro.

$condicion_modificacion Para $tipo_modificacion 2 (escribir)

1er bit (bit menos significativo, estremo derecho)en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si sus valores son iguales (==)

2do bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si sus valores son diferentes (!= , <>)

3er bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si sus valores son no idénticos (!==)

4to bit en cero desactivado) no se tiene en cuenta

en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es menor que el de la base (<)

5to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es mayor que el de la base (>)

6to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es menor o igual que el de la base (<=)

7mo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es mayor o igual que el de la base (>=)

8vo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin citerios de comparación, pero si existe una estructura o elemento en la imagen que no existe en la base no es modificada la base, es decir no se transfiere el elemento de la imagen a la base.

9no bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a registrar en los resultados, en el elemento 'fallos' de esta función, los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores.

los valores siguientes son excepciones válidas de aclarar

binarios:

000000000 (dec 0) modifica los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación, si existe una estructura o elemento en la imagen que no existe en la base es modificada la base con el elemento en cuestión, es decir se transfiere el elemento de la imagen a la base.

100000000 (dec 256) modifica los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación pero manteniendo el criterio de registrar en los resultados de esta función, en el elemento 'fallos' de esta función, los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores, además mantiene el comportamiento del 8vo bit según su valor.

110000000 (dec 384) modifica los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación pero manteniendo el criterio de

registrar en los resultados de esta función, en el elemento 'fallos' de esta función, los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores, además mantiene el comportamiento del 8vo bit según su valor.

110000000+1 (dec mayor que 384)retorna un valor null
si la lógica de los bit del 1ro al 7mo activados equivalen a modificar todo el arreglo o existe redundancia, la función retorna un valor null
ej: 000000101 o 001010000

el 8vo y 9no bit siempre pueden estar activados, no pertenece a la lógica comparativa del lenguaje

$condicion_modificacion Para $tipo_modificacion 3 (eliminar)

si se escoge una condición y esta no se cumple, el elemento base no es eliminado.

1er bit (bit menos significativo, extremo derecho)en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son iguales (==)

2do bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son idénticos (===)

3er bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son diferentes (!= , <>)

4to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son no idénticos (!==)

5to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es menor que el de la base (<)

6to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es mayor que el de la base (>)

7mo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es menor o igual que el de la base (<=)

8vo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es mayor o igual que el de la base (>=)

9no bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a registrar en los resultados de esta función los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores

los valores siguientes son excepciones validas de aclarar

binarios:

000000000 (dec 0) elimina los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir elimina todo sin criterios de comparación.

100000000 (dec 256) elimina los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir elimina todo sin criterios de comparación pero manteniendo el criterio de registrar en los resultados de esta función los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores

110000000+1 (dec mayor que 384)retorna un valor null
si la lógica de los bit del 1ro al 8vo activados equivalen a eliminar todo el arreglo o existe redundancia, la función retorna un valor null
ej: 00000101 o 001010000
entre el 1er y el 8vo bit solo pueden coexistir las combinaciones 2do y 3ro, o 2do y 5to, o 2do y 6to

el 9no bit siempre puede estar activado, no pertenece a la lógica comparativa del lenguaje

este procedimiento retorna:
- para la opción de leer retorna el elemento del arreglo base a leer si la operación es satisfactoria, y null si es insatisfactoria

- para la opción de escribir y eliminar retorna true si la operación es satisfactoria, y null si es insatisfactoria

static public function clienteEjecucionProceso(&$fuente_datos_procesos , $id_proceso = 'hereda') es el procedimiento encargado de gestionar si un proceso id_proceso (ejecutándose , actual) tiene permiso a ejecutar otro proceso, para tener un resultado 100 % cierto de este procedimiento el proceso que se quiere ejecutar ya debe tener sus datos de seguridad en la entidad $EEoConfiguracion del sistema Sockeletom, en un futuro este procedimiento tiene que controlar si un proceso que todavía no se ha ejecutado(y por esto no tiene configuración registrada en el sistema que pueda responder a este procedimiento)puede realmente ser ejecutado por el proceso actual/en ejecución, para ello recomiendo que haya un fichero estándar a la lectura de este procedimiento en el árbol de ficheros de todos los procesos, que contenga una estructura de datos con los procesos que no permite su ejecución.

$fuente_datos_procesos (array), es la fuente de datos sobre la que este procedimiento hará su gestión, es la estructura de datos sobre la que se buscará el permiso de ejecución. Esta estructura de datos es la correspondiente a la entidad EEoSeguridad.

$id_proceso (string), es el identificador del proceso al que se quiere chequear si el proceso actual (en ejecución) puede ejecutar, si su valor es vacío el procedimiento retorna el valor null.

si el proceso actual(ejecutándose) es cliente de ejecutar el proceso al que se chequea, este procedimiento devuelve la cadena 'ejecucion'

este procedimiento retorna null si es insatisfactorio o si el proceso actual no tiene permiso de ejecutar el proceso analizado.

si el $id_proceso no existe en la estructura de datos que proporciona el parámetro $fuente_datos_procesos de este procedimiento entonces se retornará el valor 'ejecucion', este es el caso típico cuando un proceso no ha registrado datos de seguridad en la entidad EEoSeguridad ya sea por voluntad o por no haberse ejecutado por primera vez, esta característica está sujeta a discusión por motivos de seguridad-accesibilidad.

static public function clienteAccesoDatosProceso(&$fuente_datos_procesos , $id_proceso = 'hereda')
es el procedimiento encargado de gestionar si un proceso id_proceso (ejecutándose , actual) tiene acceso a los datos de otro proceso que haya colocado datos en la entidad EEoDatos.

$fuente_datos_procesos (array), es la fuente de datos sobre la que este procedimiento hará su gestión, es la estructura de datos sobre la que se buscará el permiso de acceso a datos. Esta estructura de datos es la correspondiente a la entidad EEoSeguridad.

$id_proceso (string), es el identificador del proceso al que se quiere chequear si el proceso actual (en ejecución) puede acceder a sus datos colocados en la entidad EEoDatos, si su valor es vacío el procedimiento retorna el valor null.

si el proceso actual(ejecutándose) es cliente de acceder a datos colocados por $id_proceso en la entidad EEoDatos , este procedimiento devuelve un string con el tipo de acceso permitido(siempre el permitido, nunca el denegado)('leer', 'escribir', 'eliminar'), el string cuenta con la estructura siguiente 'leer::escribir::eliminar' omitiéndose el acceso no deseado , ej:'leer' o 'leer::eliminar'

este procedimiento retorna null si es insatisfactoria su gestión o si el proceso actual no tiene permiso alguno sobre los datos del proceso analizado.

static public function clienteAccesoConfiguracionProceso(&$fuente_datos_procesos , $id_proceso = 'hereda')

es el procedimiento encargado de gestionar si un proceso id_proceso (ejecutándose , actual) tiene acceso a los datos de configuración de otro proceso que haya colocado datos en la entidad EEoConfiguracion.

$fuente_datos_procesos (array), es la fuente de datos sobre la que este procedimiento hará su gestión, es la estructura de datos sobre la que se buscará el permiso de acceso a datos. Esta estructura de datos es la correspondiente a la entidad EEoSeguridad.

$id_proceso (string), es el identificador del proceso al que se quiere chequear si el proceso actual (en ejecución) puede acceder a sus datos colocados en la entidad EEoConfiguracion, si su valor es vacío el procedimiento retorna el valor null.

si el proceso actual(ejecutándose) es cliente de acceder a datos colocados por $id_proceso en la entidad EEoConfiguracion, este procedimiento devuelve un string con el tipo de acceso permitido(siempre el permitido, nunca el denegado)('leer', 'escribir', 'eliminar'), el string cuenta con la estructura siguiente 'leer::escribir::eliminar' omitiéndose el acceso no deseado , ej:'leer' o 'leer::eliminar'

este procedimiento retorna null si es insatisfactoria su gestión o si el proceso actual no tiene permiso alguno sobre los datos del proceso analizado.

static public function iniciarDatosSalidaProceso(&$fuente_datos_procesos , $datos_salida)
es el procedimiento encargado de gestionar el inicio de datos por parte de un proceso en la entidad EEoDatos.

$fuente_datos_procesos (array), es la fuente de datos sobre la que este procedimiento hará su gestión, es la estructura de datos sobre la que se colocará el contenido del parámetro $datos_salida. Esta estructura de datos es la correspondiente a la entidad EEoDatos.

$datos_salida (array), es una estructura de datos que por voluntad quiere colocar el proceso actual (en ejecución) en la entidad EEoDatos, si su valor es vacío el procedimiento retorna el valor null.

este procedimiento retorna true si es satisfactoria su gestión y null si es insatisfactoria su gestión.

static public function accederDatosSalidaProceso(&$fuente_datos_procesos , $id_proceso = 'hereda' , $estructura_acceder = array( ) , $tipo_modificacion = 1 , $condicion_modificacion = 0)
es el procedimiento encargado de acceder a la estructura de datos correspondiente a la entidad EEoDatos sobre la que se realizarán las acciones de leer, escribir y/o eliminar datos. Es la vía de acceso a datos en EEoDatos para procesos que quieran acceder a datos propios o de otros procesos. Para ello deberán cumplirse un grupo de condiciones entre las que se encuentran:
- existir una sección de datos del proceso al que se quieren acceder los datos en la entidad EEoDatos. Si el proceso intenta acceder a sus propios datos también tiene que haber iniciado datos

en la entidad EEoDatos, de lo contrario este procedimiento no hará su gestión. Para iniciar datos en la entidad EEoDatos existe un procedimiento.
- existir una sección de seguridad del proceso al que se quieren acceder los datos en la entidad EEoSeguridad. Solo se prescinde de esta condición si el proceso intenta acceder a sus propios datos.
- existir permiso de acceso para el proceso que ejecuta este procedimiento en la estructura de datos de seguridad del proceso $id_proceso en la entidad EEoSeguridad. Además de corresponderse la acción de acceso con los permisos otorgados. Solo se prescinde de esta condición si el proceso intenta acceder a sus propios datos.

Los parámetros se explican a continuación:

$fuente_datos_procesos (array), es la fuente de datos sobre la que este procedimiento hará su gestión, es la estructura de datos sobre la que se realizarán las acciones de leer, escribir y/o eliminar datos. Esta estructura de datos es la correspondiente a la entidad EEoDatos.

$id_proceso (string), es el identificador o nombre del proceso con datos en $fuente_datos_procesos y sobre estos datos es que se quiere tener acceso de leer, escribir y/o eliminar. Este parámetro admite el valor 'hereda' que quiere decir que el id_proceso es el del proceso que está en ejecución en el momento de la llamada a este procedimiento.

$estructura_acceder (array), es la estructura a acceder en el arreglo $fuente_datos_procesos, este parámetro es dependiente del parámetro $tipo_modificacion, a continuación sus diferentes funcionalidades

-para valor 1 (leer) del parámetro $tipo_modificacion, este parámetro puede contener un arreglo vacío para obtener (leer) todos los datos correspondientes a $id_proceso en la entidad EEoDatos, array() un arreglo vacío es por defecto el valor que tiene este parámetro, también se puede acceder solo a una parte de la estructura de datos si se le da el valor a este parámetro de una cadena con la estructura a la que se quiere acceder, ejemplo '[dato1][dato2]' y el devolverá el elemento correspondiente a los datos del proceso $id_proceso en la estructura [dato1][dato2].

-para valores 2 (escribir)y 3 (eliminar) del parámetro $tipo_modificacion este parámetro puede contener un arreglo asociativo con los elementos que se quieren eliminar o modificar en el arreglo de datos correspondientes a $id_proceso en la entidad EEoDatos, este arreglo contendrá la estructura exacta en la que se encuentra el elemento que se quiere escribir o eliminar, es como un mapa, una imagen de la matriz y su rama o elemento que se quiere escribir o eliminar, llamemosle de ahora en adelante "estructura imagen", en caso de que toda una rama del arreglo se quiera eliminar basta con poner 'elemento'=array() y se eliminará ese elemento junto a la rama que el constituye.

$tipo_modificacion (integer), es el tipo de modificación que se utilizará, sus posibles valores son los siguientes, 1 para leer 2 para escribir o modificar , 3 para eliminar, si su valor es diferente de los

valores permitidos este procedimiento retorna el valor null, este parámetro también debe coincidir con los permisos de escribir o eliminar que debe tener el proceso actual (en ejecución) sobre los datos a modificar o eliminar del proceso $id_proceso para que el procedimiento concluya de forma satisfactoria.

$condicion_modificacion (integer)(string), es la condición para llevar a cabo una modificación, ya sea escribir o eliminar, este parámetro no se toma en cuenta cuando $tipo_modificacion es 1 (leer), solo se toma en cuenta para $tipo_modificacion con valor 2 o 3. La diferencia en el tipo (integer) (string) está dada porque en realidad se convierte el entero en binario y el string se leerá como una representación binaria en string ej '100110111'. Cada lugar del binario representa un tipo de condición a cumplirse para que se realice la modificación en la estructura de datos, esta condición será ejecutada tomando como elementos comparativos el arreglo $estructura_acceder y el arreglo $fuente_datos_procesos, elemento por elemento homólogos de cada uno de ellos; si se escoge una condición y esta no se cumple, el elemento base ($estructura_acceder) no es modificado A continuación los valores posibles para este parámetro.

$condicion_modificacion Para $tipo_modificacion 2 (escribir)

1er bit (bit menos significativo, estremo derecho)en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si sus valores son iguales (==)

2do bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si sus valores son diferentes (!= , <>)

3er bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si sus valores son no idénticos (!==)

4to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es menor que el de la base (<)

5to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es mayor que el de la base (>)

6to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es menor o igual que el de la base (<=)

7mo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los elementos si el valor de la imagen es mayor o igual que el de la base (>=)

8vo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a modificar los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin citerios de comparación, pero si existe una estructura o elemento en la imagen que no existe en la base no es modificada la base, es decir no se transfiere el elemento de la imagen a la base.

9no bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a registrar en los resultados, en el elemento 'fallos' de esta función, los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores.

los valores siguientes son excepciones válidas de aclarar

binarios:

000000000 (dec 0) modifica los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación, si existe una estructura o elemento en la imagen que no existe en la base es modificada la base con el elemento en cuestión, es decir se transfiere el elemento de la imagen a la base.

100000000 (dec 256) modifica los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación pero manteniendo el criterio de registrar en los resultados de esta función, en el elemento 'fallos' de esta función, los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores, además mantiene el comportamiento del 8vo bit según su valor.

110000000 (dec 384) modifica los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación pero manteniendo el criterio de registrar en los resultados de esta función, en el elemento 'fallos' de esta función, los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores, además mantiene el comportamiento del 8vo bit según su valor.

110000000+1 (dec mayor que 384)retorna un valor null
si la lógica de los bit del 1ro al 7mo activados equivalen a modificar todo el arreglo o existe redundancia, la función retorna un valor null
ej: 000000101 o 001010000

el 8vo y 9no bit siempre pueden estar activados, no pertenece a la lógica comparativa del lenguaje

$condicion_modificacion Para $tipo_modificacion 3 (eliminar)

si se escoge una condición y esta no se cumple, el elemento base no es eliminado.

1er bit (bit menos significativo, extremo derecho)en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son iguales (==)

2do bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son idénticos (===)

3er bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son diferentes (!= , <>)

4to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si sus valores son no idénticos (!==)

5to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es menor que el de la base (<)

6to bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es mayor que el de la base (>)

7mo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es menor o igual que el de la base (<=)

8vo bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a eliminar los elementos si el valor de la imagen es mayor o igual que el de la base (>=)

9no bit en cero desactivado) no se tiene en cuenta
en 1 (activado)corresponde a registrar en los resultados de esta función los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores

los valores siguientes son excepciones validas de aclarar

binarios:

000000000 (dec 0) elimina los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir elimina todo sin criterios de comparación.

100000000 (dec 256) elimina los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir elimina todo sin criterios de comparación pero manteniendo el criterio de registrar en los resultados de esta función los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores

110000000+1 (dec mayor que 384)retorna un valor null
si la lógica de los bit del 1ro al 8vo activados equivalen a eliminar todo el arreglo o existe redundancia, la función retorna un valor null
ej: 00000101 o 001010000
entre el 1er y el 8vo bit solo pueden coexistir las combinaciones 2do y 3ro, o 2do y 5to, o 2do y 6to

el 9no bit siempre puede estar activado, no pertenece a la lógica comparativa del lenguaje

este procedimiento retorna:
- para la opción de leer retorna el elemento del arreglo base a leer si la operación es satisfactoria, y null si es insatisfactoria

- para la opción de escribir y eliminar retorna true si la operación es satisfactoria, y null si es insatisfactoria

class_gedees

Este fichero contiene la clase de métodos estáticos GEDEEs, que se encarga de la gestión y administración para el sistema Sockeletom de los GEDEEs y sus propiedades. Gestiona la existencia del GEDEE al que se haga una primera petición y luego guarda sus datos y referencia para posteriores llamadas al GEDEE, estas entre otras funciones realiza esta clase que a continuación explicamos su estructura y contenido:

Namespace: Emod\Nucleo
Nombre de la clase: GEDEEs
Herencia: use \Emod\Nucleo\Herramientas\GECO

Propiedades:

private static $EEoNucleo = null ;
puntero al objeto EEoNucleo ;

private static $lbGestionSeguridad = false ;
es un boolean perteneciente a la sección de seguridad de esta clase GEDEEs, que define si se gestionaran los GEDEEs bajo condiciones de seguridad, true se gestiona con seguridad, false no se gestiona con seguridad, estas condiciones de seguridad estarán declaradas en esta clase y son gestionadas por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen elementos de seguridad en el parámetro de datos entrante .

private static $lbPropietarioEntidad = true ;

es un boolean perteneciente a la sección de seguridad de esta clase GEDEEs, que define si al ingresar entidades GEDEEs en esta clase, se registra el proceso que ingresa la entidad GEDEE como su propietario, esto tiene como objetivo que la opción de eliminar entidades GEDEEs sea posible solo por su propietario o por cualquier proceso que decida eliminar una entidad GEDEE del ámbito de gestión de esta clase, true se registra el proceso propietario, false no se registra el proceso propietario, este elemento de seguridad es gestionado por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen parámetros de seguridad en el parámetro de datos entrante. No está afectado por la propiedad $lbGestionSeguridad ya que aunque este no esté activado, con valor true, si $lbPropietarioEntidad si está activado, con valor true, se lleva a cavo su función y objetivo. El proceso núcleo siempre tiene permiso de eliminar una entidad GEDEE determinada.

private static $lsAmbitoSeguridad = null ;
es un string perteneciente a la sección de seguridad de esta clase GEDEEs, que define el ámbito de seguridad a aplicar a la lista de entidades GEDEEs declaradas a permitir o denegar su gestión por parte de esta clase GEDEEs, sus valores posibles son “permisivo” y ” restrictivo” , está directamente relacionada con el elemento de seguridad $laDatosSeguridad que se expondrá más adelante, el valor “permisivo” permite la gestión de cualquier entidad GEDEE excepto las declaradas en $laDatosSeguridad, el valor “restrictivo” permite la gestión solo de las entidades GEDEEs declaradas en $laDatosSeguridad y deniega las demás entidades GEDEEs, este elemento de seguridad es gestionado por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen parámetros de seguridad en el parámetro de datos entrante. Esta propiedad está afectada por la propiedad $lbGestionSeguridad, si no está activada la propiedad $lbGestionSeguridad, con valor true, se omite la existencia y valor de esta propiedad $lsAmbitoSeguridad.

private static $lbActualizarDatosSeguridad = null ;
es un boolean perteneciente a la sección de seguridad de esta clase GEDEEs, que define la posibilidad o no de que los procesos puedan actualizar la lista de entidades GEDEEs declaradas a permitir o denegar su gestión por parte de esta clase GEDEEs. sus valores posibles son “true” y ” false” , está directamente relacionada con el elemento de seguridad $laDatosSeguridad que se expondrá más adelante, el valor “true” permite que cualquier proceso actualice la lista de entidades GEDEEs declaradas a permitir o denegar su gestión por parte de esta clase GEDEEs, false no permite lo anterior, el proceso núcleo si tiene permisos para actualizar la lista antes mencionada independientemente del valor de esta propiedad , este elemento de seguridad es gestionado por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen parámetros de seguridad en el parámetro de datos entrante. Esta propiedad está afectada por la propiedad $lbGestionSeguridad, si no está activada la propiedad $lbGestionSeguridad, con valor true, se omite la existencia y valor de esta propiedad $lbActualizarDatosSeguridad.

private static $laDatosSeguridad = array() ;
es un array perteneciente a la sección de seguridad de esta clase GEDEEs, que define la lista de entidades GEDEEs declaradas a permitir o denegar su gestión por parte de esta clase GEDEEs, está

directamente relacionada con las propiedades $lsAmbitoSeguridad y $lbActualizarDatosSeguridad de esta clase GEDEEs. Este elemento de seguridad es a nivel de clase y no de entidades GEDEEs específicas, se da permiso o no a la clase, no a una entidad correspondiente a la clase. Este elemento de seguridad es gestionado por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen parámetros de seguridad en el parámetro de datos entrante. Esta propiedad está afectada por la propiedad $lbGestionSeguridad, si no está activada la propiedad $lbGestionSeguridad, con valor true, se omite la existencia y valor de esta propiedad $laDatosSeguridad. Existe la posibilidad de poner el valor * (asterisco) que significa todos los elementos posibles, se puede hacer a nivel de $laDatosSeguridad o a nivel de un namespace. A continuación se muestra la estructura de este arreglo:

$laDatosSeguridad:
namespace_entidades_gedees 1:
clase_entidades_gedees 1
clase_entidades_gedees 2
clase_entidades_gedees N
namespace_entidades_gedees 2: *
namespace_entidades_ gedee N:
clase_entidades_gedees 1
clase_entidades_gedees 2
clase_entidades_gedees N

también $laDatosSeguridad puede tener el valor *

$laDatosSeguridad: *

namespace_entidades_gedees (string) es el nombre del namespace al que pertenecen las entidades GEDEEs. (obligatorio)

clase_entidades_gedees (string) es el nombre de la clase a la que pertenecen las entidades GEDEEs. (obligatorio)

$lsPathDirRaizEntidades = null ;
propiedad para contener el valor del path del directorio base donde se alojarán las entidades GEDEEs en el sistema Sockeletom, es el directorio que el sistema propone como contenedor de las entidades GEDEEs que se utilicen por el núcleo del sistema y los demás códigos clientes del Sockeletom. Este parámetro tiene a su vez como path raíz al directorio esquelemod, perteneciente a la estructura de directorios del sistema Sockeletom. Tomando como ejemplo el valor ¨ gedees ¨ para esta propiedad, el valor absoluto del path donde se alojaran las entidades GEDEEs sería: raíz_sistema_Sockeletom/esquelemod/gedees

private static $iniciacion = null ;
propiedad para chequear la iniciación de esta clase

private static $laEntidades = array() ;
es un arreglo que contendrá las entidades GEDEEs y sus características operacionales, entre estas características están:

namespace: (string) es el nombre del namespace al que pertenece la entidad GEDEE (obligatorio)

clase: (string) es el nombre de la clase a la que pertenece la entidad GEDEE(obligatorio)

'path_entidad_clase': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece la entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en esta clase la entidad GEDEE a gestionar, si el fichero no existe no se ingresa la entidad GEDEE en esta propiedad $laEntidades de esta clase. (obligatorio)

'tipo_entidad': (string) contiene el tipo de entidad, es decir si es de tipo clase u objeto la entidad GEDEE a gestionar, sus posibles valores son ‘clase’ o ‘objeto. (obligatorio)

'iniciacion': (string) es la forma como se iniciará una entidad GEDEEs, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades GEDEEs al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades GEDEEs le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades GEDEEs le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad GEDEE sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad GEDEE del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento ['instancias'][id_gedee] ['parámetros_iniciacion'] de la clase base, por lo que cada entidad GEDEE puede tener sus valores propios, excepto en caso de que la entidad GEDEE sea de tipo clase (tipo_entidad = 'clase') que con el primer id_gedee se inicia la clase base. (opcional)

'proceso_propietario': (string) es el proceso que hace el ingreso de la clase entidad GEDEE, este valor el procedimiento lo toma del objeto EEoNucleo conformado por el GEDEE del proceso en ejecución, seguido de :: y el id del proceso en ejecución (self::$EEoNucleo->gedeeProcesoEjecucion . '::' . self::$EEoNucleo->idProcesoEjecucion()) y es para definir el único proceso que puede eliminar la entidad namespace/clase de este arreglo, este elemento está sujeto a la propiedad $lbPropietarioEntidad de la sección de seguridad de esta clase GEDEEs. Cuando el proceso propietario de la entidad namespace/clase decide eliminar esta entidad, se eliminará junto con todas las instancias contenidas de esta namespace/clase.

'instancias': (array) contiene las instancias que se creen de la clase correspondiente, contiene las instancias y los datos correspondientes a cada instancia, entre estos datos están:

id_entidad_gedee: (string) es el nombre o identificador de la entidad GEDEE (obligatorio), los id_entidad_gedee pueden ser ingresados en el momento que se ingresa el namespace/clase de la clase entidad_gedee o posteriormente haciendo una petición de ingreso de instancia de una entidad_gedee a través del procedimiento * ingresarEntidad * de esta clase.

'registro': (boolean) en caso de 'tipo_entidad' tener valor 'clase' entonces esta propiedad contiene el valor true si ha sido solicitada al menos una vez.

'objeto': (&obj) en caso de 'tipo_entidad' tener valor 'objeto' entonces esta propiedad contiene la referencia al objeto, entidad GEDEE a gestionar.

'proceso_propietario': (string) es el proceso que hace el ingreso de la entidad GEDEE, este valor el procedimiento lo toma del objeto EEoNucleo conformado por el gedee del proceso en ejecución, seguido de :: y el id del proceso en ejecución (self::$EEoNucleo->gedeeProcesoEjecucion . '::' . self::$EEoNucleo->idProcesoEjecucion()) y es para definir el único proceso que puede eliminar la entidad instancia de este arreglo, este elemento está sujeto a la propiedad $lbPropietarioEntidad de la sección de seguridad de esta clase GEDEEs. Se aplica solo a la entidad GEDEE específica, no tiene que ser el mismo propietario que el de la entidad namespace/clase de esta entidad GEDEE.

'datos': es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad GEDEE en esta clase GEDEEs; estos datos se crean en el momento del ingreso de la entidad GEDEE sin que permita cambios posteriores.

la estructura del arreglo quedaría así:

```
$laEntidades = array(
                    'nombre namespace' => array(
                                        'nombre clase' => array(
                                                            'path_entidad_clase' => 'path_entidad_clase',
                                                            'tipo_entidad' => 'tipo_entidad',
                                                            'iniciacion' => 'procedimiento'
                                                            'proceso_propietario' => 'proceso_que_ingresa_la_entidad',
                                                            'instancias' => array(
                                                                        'id_entidad_gedee1' => array(
                                                                                    'registro' => 'true o false,
                                                                                    'objeto' => &objeto,
                                                                                    'proceso_propietario' => 'proceso_que_ingresa_la_entidad',
                                                                                    'datos' => array( 'dato1' => 'valor_dato' , 'datoN' =>datoN')
                                                                                            ) ,
                                                                        'id_entidad_gedeeN' => array(
                                                                                    'registro' => 'true o false,
                                                                                    'objeto' => &objeto,
                                                                                    'proceso_propietario' => 'proceso_que_ingresa_la_entidad',
                                                                                    'datos' => array( 'dato1' => 'valor_dato' , 'datoN' =>datoN')
                                                                                            )
                                                                                )
                                                                    )
```

)
)

Procedimientos:

public static function iniciacion($La_configuracion_nucleo_gedees)
es el procedimiento encargado de iniciar el funcionamiento de esta clase y además guardar en las propiedades de esta clase los elementos existentes como valores de la sección gedees propuestas por el sistema Sockeletom en la configuración del sistema Sockeletom. Además realiza el ingreso de las entidades GEDEEs propuestas a gestionar en la configuración del sistema Sockeletom. Para un mejor entendimiento de los parámetros que la configuración del sistema proporciona a este procedimiento, debe consultarse la sección gedees del fichero de configuración del sistema Sockeletom en este documento.

este procedimiento retorna null si ya estaba inicializada la clase GEDEEs , retorna true si la gestión es exitosa, detiene el proceso php si los parámetros son incompatibles con el procedimiento, emitiendo un error fatal (exit).

public static function pathRaizEntidades ()
es el procedimiento encargado de brindar, retornar el path del directorio raíz para los recursos de GEDEEs en el sistema Sockeletom. Este será el fragmento de path que completará el path del directorio base de los recursos de GEDEEs, del directorio que el sistema propone como contenedor de los recursos de GEDEEs que se utilicen por el núcleo del sistema y los demás códigos clientes del Sockeletom. Este parámetro tiene además como path raíz al directorio esquelemod, pertenecientes a la estructura de directorios del sistema Sockeletom. Tomando como ejemplo el valor ¨ gedees ¨, quedará como resultado el path:
raíz_sistema_Sockeletom/esquelemod/gedees

este procedimiento retorna el path gestionado o null en caso de no existir el path a gestionar.

public static function actualizarDatosSeguridadEntidades ($La_datos , $Ls_tipo_actualizacion)
es el procedimiento encargado de gestionar la actualización de la propiedad self:: $laDatosSeguridad de esta clase GEDEEs, la gestión de actualización está sujeta al valor de la propiedad self::$lbActualizarDatosSeguridad perteneciente a la sección de seguridad de esta clase GEDEEs. Los parámetros se explican a continuación:

$La_datos: (array) es el arreglo con los datos a actualizar.

$Ls_tipo_actualizacion: (string) es el tipo de actualización a realizar, sus posibles valores son:

'renovar': es la sustitución del arreglo $La_datos por el arreglo actual en self::$laDatosSeguridad.

'adicionar': es la adición del contenido del arreglo $La_datos en el arreglo actual self:: $laDatosSeguridad.

'eliminar': es la eliminación del contenido del arreglo $La_datos en el arreglo actual self:: $laDatosSeguridad.

este procedimiento retorna true si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

public static function permisionEntidad ($Ls_namespace , $Ls_clase)
es el procedimiento encargado de gestionar si una clase entidad GEDEE determinada tiene permiso de ser gestionado por esta clase GEDEEs, el permiso es a nivel de clase y no de entidades específicas de esta, se da permiso o no a la clase no a una entidad correspondiente a la clase; la gestión de chequeo de permisos está sujeto al valor de las propiedades self::$lbGestionSeguridad, self::$lsAmbitoSeguridad y self::$laDatosSeguridad pertenecientes a la sección de seguridad de esta clase GEDEEs; específicamente chequea las clases permitidas o denegadas en self:: $laDatosSeguridad según el self::$lsAmbitoSeguridad declarado. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la clase entidad GEDEE a chequear su permiso de gestión (obligatorio)

$Ls_clase: (string) es el nombre de la clase a la que pertenece la clase entidad GEDEE a chequear su permiso de gestión (obligatorio)

este procedimiento retorna true si la clase entidad GEDEE tiene permiso a ser gestionada por esta clase, de lo contrario retorna null.

public static function gestionControlIngresoEntidad ($Ls_path_control_gedee , $Ls_referencia_path_control = 'relativo' , $Ls_namespace = null , $Ls_clase = null , $Ls_path_entidad_clase = null , $Ls_referencia_path_entidad = null , $Ls_tipo_entidad = null , $La_instancias = null , $Ls_iniciacion = null)
es el procedimiento encargado de gestionar el ingreso de una entidad GEDEE al contenedor $laEntidades de esta clase, utilizando entre sus parámetros un parámetro que hace referencia al path de un fichero control.

Se llama fichero control a un fichero o script php que se necesita su ejecución como parte de la gestión de ingreso de la entidad GEDEE a esta clase GEDEEs. El control puede utilizarse para chequear, gestionar o poner a punto algún elemento necesario para el ingreso de la entidad GEDEE a esta clase.

El fichero control es ejecutado (require_once) por este procedimiento y puede utilizarse en conjunto con los demás parámetros de este procedimiento o gestionar el mismo la inclusión (require_once) de la clase entidad GEDEE y retornar el mismo los parámetros que necesita este procedimiento para la gestión del ingreso de la entidad GEDEE a esta clase.

Las diferentes combinaciones que pueden utilizarse devienen diferentes funcionalidades que son:

Funcionalidad 1 donde el control debe ejecutarse y las características de la entidad se pasan como parámetros independientes del control. Si se declaran parámetros para el control y a la vez se declaran los parámetros
$Ls_namespace, $Ls_clase, $Ls_path_entidad_clase, $Ls_referencia_path_entidad, $Ls_tipo_entidad , $La_instancias = null , $Ls_iniciacion este procedimiento incluirá el fichero control y además utilizará los parámetros antes mencionados para la gestión de ingreso de la entidad GEDEE , no importa que el control retorne también los parámetros antes mencionados porque este procedimiento da prioridad a los valores de sus parámetros y nunca evalúa los parámetros homólogos retornados por el control. Tienen que existir todos los parámetros antes mencionados para que esta opción se ejecute, si solo uno de estos parámetros es vacío entonces se pasa a las siguientes funcionalidades.

antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de la entidad GEDEE, si no existe el fichero el procedimiento no ingresa la entidad GEDEE y retorna null. Ingresar una entidad GEDEE no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de esta entidad GEDEE. Los parámetros se explican a continuación:

$Ls_path_control_gedee : (string) es el path del fichero control a ejecutar para el ingreso de la entidad GEDEEs. (obligatorio)

$Ls_referencia_path_control: (string) es la forma que se hace referencia a $Ls_path_control_gedee, tiene tres posibles valores, *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el $Ls_path_control_gedee relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el $Ls_path_control_gedee relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el $Ls_path_control_gedee de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad GEDEE.

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad GEDEE.

$Ls_path_entidad_clase: (string) contiene el path del fichero que alberga el código de la clase a la que pertenece la entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, si el fichero no existe no se ingresa la entidad GEDEE al arreglo.

$Ls_referencia_path_entidad: (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el $Ls_path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el $Ls_path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el $Ls_path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

$Ls_tipo_entidad: (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

$Ls_iniciacion: (string) es la forma como se iniciará cada id_entidad_gedee, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de $Ls_iniciacion son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades GEDEEs al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades GEDEEs le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades GEDEEs le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad GEDEE sea de tipo clase ($Ls_tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad GEDEE del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento [$La_instancias][id_entidad_gedee]['parametros_iniciacion'] de la clase base, por lo que cada entidad GEDEE puede tener sus valores propios, excepto en caso de que la entidad GEDEE sea de tipo clase ($Ls_tipo_entidad = 'clase') que con el primer id_entidad_gedee se inicia la clase base. Los elementos [$La_instancias][id_entidad_gedee]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

$La_instancias (array) es un arreglo con los identificadores de entidades GEDEEs a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_gedee. (obligatorio)

id_entidad_gedee (string) es un nombre o identificador de cada entidad GEDEE instancia de la clase correspondiente. y a su vez este id_entidad_gedee es un arreglo asociativo al que se puede

insertar una llave de nombre 'parametros_iniciacion' y otra de nombre 'datos', posteriormente este arreglo id_entidad_gedee se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_gedee como sea necesario ya que la entidad GEDEE puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad GEDEE (id_entidad_gedee), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad GEDEE a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad_gedee declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad GEDEE es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad GEDEE en esta clase GEDEEs; estos datos se crean en el momento del ingreso de la entidad GEDEE sin que permita cambios posteriores. (opcional)

Para el resto de las funcionalidades. Es necesario volver a aclarar que si se declaran parámetros en este procedimiento para el fichero control y al menos uno de los restantes es vacío entonces este procedimiento ejecuta (require_once) el control, desecha los demás parámetros pasados a este procedimiento y espera en forma de retorno(return) desde el fichero control requerido(require_once), un arreglo asociativo con las posibles siguientes estructuras:

Funcionalidad 2 donde el control debe ejecutarse, y este mismo ya incluyó la clase y nos retorna las características de la entidad incluido la o las instancias(id_entidad_gedee)(objeto) de la clase, para que se ejecute esta funcionalidad esta estructura tiene que estar completa, si faltase algún elemento no se ejecutará esta funcionalidad, también tiene como condición que el elemento $La_resultado_require['referencia_path_entidad'] no exista, la responsabilidad de incluir la estructura correcta de los id_entidad_gedee instanciadas (objeto) o registradas(clase) es responsabilidad del control, también es obligatorio que exista al menos una id_entidad_gedee en la estructura de datos que se espera por retorno. La estructura que se espera por retorno es la siguiente:

$La_resultado_require['namespace'] (obligatorio)
$La_resultado_require['clase'] (obligatorio)
$La_resultado_require['path_entidad_clase'] (obligatorio)
$La_resultado_require['tipo_entidad'] (obligatorio)
$La_resultado_require['iniciacion'] (opcional)

$La_resultado_require['instancias'] (obligatorio)
$La_resultado_require['instancias'] [id_entidad_gedee '] (obligatorio)
$La_resultado_require['instancias'] [id_entidad_gedee '] ['objeto'] (opcional)
$La_resultado_require['instancias'] [id_entidad_gedee] ['registro'] (opcional)
$La_resultado_require['instancias'] [id_entidad_gedee] [' datos'] (opcional)

'namespace': (string) es el nombre del namespace al que pertenece la entidad GEDEE. (obligatorio)

'clase': (string) es el nombre de la clase a la que pertenece la entidad GEDEE. (obligatorio)

'path_entidad_clase': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad GEDEE al arreglo $laEntidades de esta clase. (obligatorio)

'tipo_entidad': (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

'iniciacion': (string) es la forma como se iniciará cada id_entidad_gedee , este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades GEDEEs al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades GEDEEs le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades GEDEEs le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad GEDEE sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad GEDEE del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento ['instancias'][id_entidad_gedee]['parametros_iniciacion'] de la clase base, por lo que cada entidad GEDEE puede tener sus valores propios, excepto en caso de que la entidad GEDEE sea de tipo clase (tipo_entidad = 'clase') que con el primer id_entidad_gedee se inicia la clase base. Los elementos ['instancias'][id_entidad_gedee]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

'instancias' (array) es un arreglo con los identificadores de entidades GEDEEs a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_gedee. (obligatorio)

id_entidad_gedee (string) es un nombre o identificador de cada entidad GEDEE instancia de la clase correspondiente. y a su vez este id_entidad_gedee es un arreglo asociativo al que se puede insertar una llave de nombre objeto, una llave de nombre registro, y otra de nombre datos, las llaves objeto, y registro no deben coexistir ya que la existencia de una determina la no existencia de la otra ,posteriormente este arreglo id_entidad_gedee se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_gedee como sea necesario ya que la entidad GEDEE puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'objeto': (&obj) es un apuntador al objeto instancia de la clase entidad GEDEE a contener en el contenedor $laEntidades de esta clase. Este elemento es obligatorio si el elemento tipo_entidad tiene como valor el string objeto y estrictamente se considera iniciado.

'registro': (boolean true) es para definir cuando una entidad GEDEE a contener en el contenedor $laEntidades de esta clase, y de tipo_entidad clase (procedimientos estáticos) es iniciada . Este elemento es obligatorio si el elemento tipo_entidad tiene como valor el string clase y estrictamente se considera iniciada.

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad GEDEE en esta clase GEDEEs; estos datos se crean en el momento del ingreso de la entidad GEDEE sin que permita cambios posteriores. (opcional)

Funcionalidad 3 donde el control debe ejecutarse, y este nos retorna(return) los elementos o parámetros para ingresar una clase, y reserva al menos un id_entidad_gedee para su posterior instanciación, no debe tener id_entidades_gedees instanciadas(objeto) o registradas(clase), y es obligatorio que exista en los parámetros retornados por el control el elemento $La_resultado_require['referencia_path_entidad']. Entiéndase que esta funcionalidad solo realiza su gestión si no existe ningún id_entidad_gedee instanciado este factor es obligatorio. La estructura que se espera por retorno es la siguiente:

$La_resultado_require['namespace'] (obligatorio)
$La_resultado_require['clase'] (obligatorio)
$La_resultado_require['path_entidad_clase'] (obligatorio)
$La_resultado_require['referencia_path_entidad '] (obligatorio)
$La_resultado_require['tipo_entidad'] (obligatorio)
$La_resultado_require['iniciacion'] (opcional)
$La_resultado_require['instancias'] (obligatorio)
$La_resultado_require['instancias'] [id_entidad_gedee] (obligatorio)
$La_resultado_require['instancias'] [id_entidad_gedee] ['parametros_iniciacion'] (opcional)
$La_resultado_require['instancias'] [id_entidad_gedee] [' datos'] (opcional)

'namespace': (string) es el nombre del namespace al que pertenece la entidad GEDEE. (obligatorio)

'clase': (string) es el nombre de la clase a la que pertenece la entidad GEDEE. (obligatorio)

'path_entidad_clase': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad GEDEE al arreglo $laEntidades de esta clase. (obligatorio)

'referencia_path_entidad': (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

'tipo_entidad': (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

'iniciacion': (string) es la forma como se iniciará cada id_entidad_gedee, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades GEDEEs al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades GEDEEs le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades GEDEEs le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad GEDEE sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad GEDEE del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento ['instancias'][id_entidad_gedee]['parametros_iniciacion'] de la clase base, por lo que cada entidad GEDEE puede tener sus valores propios, excepto en caso de que la entidad GEDEE sea de tipo clase (tipo_entidad = 'clase') que con el primer id_entidad_gedee se inicia la clase base. Los elementos ['instancias'][id_entidad_gedee]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

'instancias' (array) es un arreglo con los identificadores de entidades GEDEEs a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_gedee. (obligatorio)

id_entidad_gedee (string) es un nombre o identificador de cada entidad GEDEE instancia de la clase correspondiente. y a su vez este id_entidad_gedee es un arreglo asociativo al que se puede insertar una llave de nombre 'parametros_iniciacion' y otra de nombre 'datos', posteriormente este arreglo id_entidad_gedee se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_gedee como sea necesario ya que la entidad GEDEE puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad GEDEE (id_entidad_gedee), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad GEDEE a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad_gedee declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad GEDEE es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad GEDEE en esta clase GEDEEs; estos datos se crean en el momento del ingreso de la entidad GEDEE sin que permita cambios posteriores. (opcional)

La funcionalidad que el control ejecutado devuelva los datos de la clase y sus elementos['instancias'][id_entidad_gedee] no instanciados, es por seguridad ya que existen otras formas para crear controles que retornen estructuras con sus id_entidad_gedee ya instanciados(objeto) o registrados(clase); Esta característica está abierta a cambios futuro en dependencia de la opinión mayoritaria de los usuarios del sistema Sockeletom.

antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de la entidad GEDEE, si no existe el fichero el procedimiento no ingresa la entidad GEDEE y retorna null. Ingresar una entidad GEDEE no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de esta entidad GEDEE.

este procedimiento retorna true si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

```
public static function gestionIngresoEntidad ($Ls_namespace , $Ls_clase , $Ls_path_entidad_clase , $Ls_referencia_path_entidad , $Ls_tipo_entidad , $La_instancias , $Ls_iniciacion = null)
```

es el procedimiento encargado de gestionar el ingreso de una entidad GEDEE al contenedor $laEntidades de esta clase, antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de la entidad GEDEE, si no existe el fichero el procedimiento no ingresa la entidad GEDEE y retorna null. Ingresar una entidad GEDEE no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de esta entidad GEDEE. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad GEDEE. (obligatorio)

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad GEDEE. (obligatorio)

$Ls_path_entidad_clase: (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad GEDEE al arreglo $laEntidades de esta clase. (obligatorio)

$Ls_referencia_path_entidad: (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

$Ls_tipo_entidad: (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

$Ls_iniciacion: (string) es la forma como se iniciará cada id_entidad_gedee, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades GEDEEs al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades GEDEEs le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades GEDEEs le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad GEDEE sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad GEDEE del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento

$La_instancias[id_entidad_gedee]['parametros_iniciacion'] de la clase base, por lo que cada entidad GEDEE puede tener sus valores propios, excepto en caso de que la entidad GEDEE sea de tipo clase (tipo_entidad = 'clase') que con el primer id_ entidad_gedee se inicia la clase base. Los elementos $La_instancias[id_entidad_gedee]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

$La_instancias (array) es un arreglo con los identificadores de entidades GEDEEs a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_gedee. (obligatorio)

id_entidad_gedee (string) es un nombre o identificador de cada entidad GEDEE instancia de la clase correspondiente. y a su vez este id_entidad_gedee es un arreglo asociativo al que se puede insertar una llave de nombre 'parametros_iniciacion' y otra de nombre 'datos', posteriormente este arreglo id_entidad_gedee se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_gedee como sea necesario ya que la entidad GEDEE puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad GEDEE (id_entidad_gedee), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad GEDEE a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad_gedee declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad GEDEE es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad GEDEE en esta clase GEDEEs; estos datos se crean en el momento del ingreso de la entidad GEDEE sin que permita cambios posteriores. (opcional)

public static function gestionIngresosEntidades ($La_propiedades_gedees)
es el procedimiento encargado de gestionar el ingreso de varias entidades GEDEEs al contenedor $laEntidades de esta clase, este procedimiento desglosa un arreglo con datos de varias entidades GEDEEs y realiza la gestión de ingreso según sus características como lo hacen individualmente los procedimientos self::gestionControlIngresoEntidad y self::gestionIngresoEntidad, es importante aclarar que cada entidad GEDEE será procesada estrictamente con los procedimientos self::gestionControlIngresoEntidad o self::gestionIngresoEntidad por lo que los datos de las entidades GEDEEs tienen que cumplir con las condicionantes del procedimiento respectivo para

que la gestión sea exitosa, antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de cada entidad GEDEE, si no existe el fichero el procedimiento no ingresa la entidad GEDEE y continua su gestión. Ingresar una entidad GEDEE no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de esta entidad GEDEE. Los parámetros se explican a continuación:

$La_propiedades_gedees: es un arreglo en el que los elementos llave de este arreglo son los namespaces de las entidades GEDEEs a gestionar su ingreso en esta clase, la llave namespace representa a su vez un arreglo con las clases y algunas características de estas clases que finalmente son las entidades GEDEEs declaradas, la llave clase representa a su vez un arreglo con algunas características necesarias para la correcta gestión de las entidades GEDEEs, un ejemplo de esta estructura se expone a continuación:

```
$La_propiedades_gedees
                    namespace_entidad_gedee1:
                                clase_entidad_gedee 1:
                                            'path_entidad_clase':
                                            'referencia_path_entidad':
                                            'path_control':
                                            'referencia_path_control':
                                            'tipo_entidad':
                                           'iniciación':
                                            'instancias':
                                                        id_entidad_gedee 1:
                                                                    'parametros_iniciacion':
                                                                    'datos':
                                                        id_entidad_gedee 2
                                                                    'parametros_iniciacion':
                                                        id_entidad_gedee 3
                                                        id_entidad_gedee 4:
                                                                    'datos':

                    namespace_entidad_gedee 2:
                                clase_entidad_gedee 1:
                                            'path_entidad_clase':
                                            'referencia_path_entidad':
                                            'path_control':
                                            'referencia_path_control':
                                            'tipo_entidad':
                                            'instancias':
                                                        id_entidad_gedee 1:
                                                                    'datos':
                                                        id_entidad_gedee 2
```

```
                                        id_entidad_gedee 3
                                        id_entidad_gedee 4
                        clase_entidad_gedee 2:
                                'path_entidad_clase':
                                'referencia_path_entidad':
                                'path_control':
                                'referencia_path_control':
                                'tipo_entidad':
                                'iniciación':
                                'instancias':
                                        id_entidad_gedee 1:
                                                'datos':
                                        id_entidad_gedee 2:
                                                'parametros_iniciacion':
                                                'datos':
                                        id_entidad_gedee 3:
                                                'datos':
                                        id_entidad_gedee 4:
                                                'datos':
```

los elementos de este arreglo se explican a continuación:

namespace: (string) es el nombre del namespace al que pertenece la entidad GEDEE. (obligatorio en dependencia del procedimiento gestor)

clase: (string) es el nombre de la clase a la que pertenece la entidad GEDEE. (obligatorio en dependencia del procedimiento gestor)

'path_entidad_clase': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad GEDEE al arreglo $laEntidades de esta clase. (obligatorio en dependencia del procedimiento gestor)

'referencia_path_entidad': (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio en dependencia del procedimiento gestor)

'path_control': es el path de un fichero que controlará condiciones y/o el inicio o inclusión de una entidad GEDEE, el gestor de GEDEEs espera un retorno de este control con un arreglo asociativo donde se encuentren los datos namespace_entidad_gedee, clase_entidad_gedee, id_entidad_gedee, path_entidad_clase, referencia_path_entidad , tipo_entidad , instancias con sus datos, o combinaciones de estos, este arreglo será procesado por esta clase Emod\Nucleo\GEDEEs que es quien gestiona y administra los GEDEEs en el sistema Sockeletom. (obligatorio en dependencia del procedimiento gestor)

'referencia_path_control': (string) es la forma que se hace referencia a $Ls_path_control_gedee, tiene tres posibles valores, *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el $Ls_path_control_gedee relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el $Ls_path_control_gedee relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el $Ls_path_control_gedee de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio en dependencia del procedimiento gestor)

'tipo_entidad': (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio en dependencia del procedimiento gestor)

'iniciacion': (string) es la forma como se iniciará cada id_entidad_gedee, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades GEDEEs al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades GEDEEs le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades GEDEEs le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad GEDEE sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad GEDEE del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento ['instancias'][id_entidad_gedee]['parametros_iniciacion'] de la clase base, por lo que cada entidad GEDEE puede tener sus valores propios, excepto en caso de que la entidad GEDEE sea de tipo clase (tipo_entidad = 'clase') que con el primer id_entidad_gedee se inicia la clase base. Los elementos ['instancias'][id_entidad_gedee]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional en dependencia del procedimiento gestor)

'instancias': (array) es un arreglo con los identificadores de entidades GEDEEs a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado

admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_gedee. (obligatorio en dependencia del procedimiento gestor)

id_entidad_gedee (string) es un nombre o identificador de cada entidad GEDEE instancia de la clase correspondiente. y a su vez este id_entidad_gedee es un arreglo asociativo al que se puede insertar una llave de nombre parametros_iniciacion y otra de nombre datos, posteriormente este arreglo id_entidad_gedee se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_gedee como sea necesario ya que la entidad GEDEE puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1 en dependencia del procedimiento gestor) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad GEDEE (id_entidad_gedee), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad GEDEE a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad_gedee declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad GEDEE es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional en dependencia del procedimiento gestor)

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad GEDEE en esta clase GEDEEs; estos datos se crean en el momento del ingreso de la entidad GEDEE sin que permita cambios posteriores. (opcional en dependencia del procedimiento gestor)

se aconseja por parte del grupo de desarrollo del núcleo Sockeletom que los namespaces se declaren de forma absoluta, comenzando con '\ '

este procedimiento retorna true si la gestión de al menos una entidad GEDEE fue llevada a cabo con éxito, de lo contrario retorna null.

public static function existenciaEntidad ($Ls_namespace , $Ls_clase , $Ls_id_gedee = null)
es el procedimiento encargado de gestionar si existe una entidad GEDEE determinada en el contenedor $laEntidades de esta clase, si existe una entidad GEDEE a disposición de pedido y uso, esta entidad GEDEE no tiene obligatoriamente que estar iniciada, es decir no tiene obligatoriamente que existir la instancia ya creada.

Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad GEDEE. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad GEDEE. (obligatorio).

$Ls_id_gedee: (string) es el nombre o id de la entidad GEDEE hija de una clase ya ingresada en $this->laEntidades, este parámetro puede ser vacío, de ser vacío se entiende que se busca la existencia de una clase(\namespace\clase) ya ingresada en $this->laEntidades, clase base para entidades GEDEEs instancias(objeto) o clase (clase de procedimientos estáticos).

Si $Ls_id_gedee tiene valor y existe como entidad GEDEE hija de la \namespace\clase base en la estructura de datos de $this->laEntidades entonces este procedimiento retorna el tipo_entidad (‘objeto’ o ‘clase’) de la clase base. Si no existe como entidad GEDEE hija el procedimiento retorna null.

Si $Ls_id_gedee no tiene valor, o tiene valor vacío, existe la entidad GEDEE \namespace\clase base en la estructura de datos de $this->laEntidades entonces este procedimiento retorna el tipo_entidad (‘objeto’ o ‘clase’) de la clase base. Si no existe como la entidad GEDEE \namespace\clase base el procedimiento retorna null.

public static function ingresarEntidad ($Ls_namespace , $Ls_clase , $Ls_id_gedee , $La_parametros_iniciacion = null)
es el procedimiento encargado de gestionar el ingreso de entidades GEDEEs hija de la \namespace\clase base en la estructura de datos de $this->laEntidades. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad GEDEE, es obligatorio que exista ya en $this->laEntidades. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad GEDEE, es obligatorio que exista ya en $this->laEntidades. (obligatorio).

$Ls_id_gedee: (string) es el nombre o id de una entidad GEDEE a ingresar en la estructura instancias de la clase base \namespace\clase en la estructura de datos de $this->laEntidades(obligatorio).

$La_parametros_iniciacion: (array) cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad GEDEE ($Ls_id_gedee), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad GEDEE a gestionar sea de tipo clase (tipo_entidad

= 'clase') no tiene efecto este parámetro ya que se gestionó en el primer elemento $Ls_id_gedee que se ingreso en la clase base. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

public static function entidad ($Ls_namespace , $Ls_clase , $Ls_id_gedee , $Ls_tipo_referencia = 'referencia')
es el procedimiento encargado de gestionar la entidad GEDEE hija de la \namespace\clase base en la estructura de datos de $this->laEntidades identificada por $Ls_id_gedee, es decir si la entidad es de tipo ‘objeto’ retronará la referencia a un objeto instancia de la entidad GEDEE, si la entidad es de tipo clase retornará un string compuesto por el namespace y nombre de la clase base. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad GEDEE. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad GEDEE. (obligatorio).

$Ls_id_gedee: (string) es el nombre o id de la entidad GEDEE hija de una clase ya ingresada en $this->laEntidades, el string en $Ls_id_gedee tiene obligatoriamente que existir como elemento en la estructura instancias de la clase base en $this->laEntidades . (obligatorio).

$Ls_tipo_referencia: (string) es el tipo de referencia a objeto que devuelve el procedimiento en caso de que el tipo de entidad sea ‘objeto‘, este parámetro tiene dos valores posibles ‘referencia‘ y ‘clon‘, referencia devuelve un puntero a la entidad guardada en el arreglo $this->laEntidades, y clon devuelve un clon de este objeto. Si el tipo de entidad es ‘clase‘ no se considera este parámetro. Por defecto este parámetro tiene como valor ‘referencia‘.

Este procedimiento devuelve (retorna) una referencia o clon a objeto si el tipo_entidad de la clase base es (‘objeto’); devuelve (retorna) un string de la forma \namespace\clase si el tipo_entidad de la clase base es (‘clase’); de lo contrario retorna null.

La instanciación de los objetos ocurre en el momento que es pedido por primera vez una instancia a través de este procedimiento.

En el caso de la instanciación o iniciación de entidades de tipo ‘objeto’, e iniciación de entidades de tipo ‘clase’, estas entidades pueden emitir una excepción desde su constructor o procedimiento de iniciación si existe algún problema de instanciación o iniciación, ya que este procedimiento tiene un bloque try y catch para manejar las excepciones, el bloque catch espera un objeto de la clase \Exception y emite el mensaje de este, el código es de la siguiente forma:

```
catch( \Exception $e)
  {
      echo '<p>Excepción capturada: '.$e->getMessage(). "<p>" ;
```

```
        return null;
    }
```
Por lo que el lanzamiento de la excepción sería:

```
throw new \Exception('mensaje.');
```

Esta funcionalidad es muy útil para interrumpir la instanciación de una clase si existe algún error en esta instanciación, y así no se tienen objetos con problemas o sin uso por desperfectos, en este contenedor de instancias.

Lo anterior se aplica a entidades de tipo ‘objeto’ que lancen su excepción desde el constructor, o entidades de tipo ‘clase’ que lancen su excepción desde el procedimiento de iniciación; Quedan excluidos las entidades de tipo ‘objeto’ que lancen su excepción desde un procedimiento de iniciación(no __construct) ya que estos ya fueron instanciados existen como instancias en el contenedor, estos casos solo tienen la opción de emitir un mensaje.

public static function datosEntidad ($Ls_namespace , $Ls_clase , $Ls_id_gedee)
es el procedimiento encargado de gestionar y retornar los datos, que de existir, tiene una entidad GEDEE que haya ingresado en esta clase, estos datos son los correspondientes al elemento ‘datos’ que puede traer consigo una entidad GEDEE en el momento de su ingreso al contenedor $laEntidades de esta clase. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad GEDEE. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad GEDEE. (obligatorio).

$Ls_id_gedee: (string) es el nombre o id de la entidad GEDEE hija de una clase ya ingresada en $this->laEntidades, el string en $Ls_id_gedee tiene obligatoriamente que existir como elemento en la estructura instancias de la clase base en $this->laEntidades . (obligatorio).

este procedimiento retorna el arreglo ‘datos’ perteneciente a la entidad GEDEE propietaria si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

public static function idEntidades ($Ls_namespace , $Ls_clase)
es el procedimiento encargado de gestionar y retornar los nombres o identificadores de entidades GEDEEs que hayan ingresado en el namespace y clase correspondientes a los parámetros de este procedimiento, estos datos son los correspondientes al elemento ‘instancias’ que debe tener consigo cada clase base GEDEE en el contenedor $laEntidades de esta clase. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad GEDEE. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad GEDEE. (obligatorio).

si la gestión fue llevada a cabo con éxito este procedimiento retorna las llaves del array ‘instancias’ contenidos en el namespace y clase, clase base GEDEE en el contenedor $laEntidades de esta clase, de lo contrario retorna null.

public static function eliminarEntidad ($Ls_namespace , $Ls_clase , $Ls_id_gedee = null)
es el procedimiento encargado de eliminar una entidad GEDEE clase base, o una entidad GEDEE de la estructura ‘instancias’ de una clase base, existentes ambas en el contenedor $laEntidades de esta clase. Este procedimiento está sujeto al valor de la propiedad self::$lbPropietarioEntidad perteneciente a la sección de seguridad de esta clase GEDEEs, si esta propiedad tiene valor true solo puede eliminar la entidad GEDEE el proceso que figura como su propietario, de lo contrario puede eliminarse desde cualquier proceso, el proceso núcleo siempre tiene derechos de eliminar cualquier entidad GEDEE.

Existen dos posibilidades de uso de este procedimiento:

La primera es eliminar una entidad GEDEE clase base, es decir una namespace\clase registrada como clase base de entidades GEDEEs, para ello se pasan los valores correspondientes al namespace y clase de la entidad a eliminar, dejando vacío el parámetro $Ls_id_gedee, recordar que esta operación está sujeto a la sección de seguridad de esta clase, donde se define quién puede eliminar una entidad determinada.

La segunda es eliminar una entidad GEDEE de la estructura ‘instancias’ de una clase base, para ello debe existir la entidad GEDEE en la estructura $laEntidades[$Ls_namespace][$Ls_clase] [‘instancias’] de esta clase, el parámetro $Ls_id_gedee no puede ser vacío, recordar que esta operación está sujeto a la sección de seguridad de esta clase, donde se define quien puede eliminar una entidad determinada.

Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad GEDEE. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad GEDEE. (obligatorio).

$Ls_id_gedee: (string) es el nombre o id de la entidad GEDEE hija de una clase ya ingresada en $this->laEntidades, el string en $Ls_id_gedee tiene obligatoriamente que existir como elemento en la estructura instancias de la clase base en $this->laEntidades . (obligatorio).

este procedimiento retorna true si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

class_eherramientas.php

Este fichero contiene la clase de procedimientos estáticos Herramientas. Esta clase está compuesta por propiedades, y procedimientos estáticos para la gestión de entidades herramientas, desde la inclusión del script o clase como fichero hasta el retorno de referencias o clonaciones a objetos herramientas, gestiona y sirve como contenedor de estas entidades, una opción clara para la inyección de dependencias. En su gestión las entidades herramientas no son incluidas o instanciadas hasta el momento que se hace la petición de la instancia para su uso. Para un mejor entendimiento de cómo se gestionan las herramientas se debe consultar cada propiedad y procedimiento de esta clase

A continuación mostramos una lista de las propiedades y procedimientos que esta clase pone a disposición de los códigos clientes del Sockeletom:

Namespace: Emod\Nucleo
Nombre de la clase: Herramientas
Herencia: use \Emod\Nucleo\Herramientas\GECO

Propiedades:

private static $EEoNucleo = null ;
puntero al objeto EEoNucleo ;

private static $lbGestionSeguridad = false ;
es un boolean perteneciente a la sección de seguridad de esta clase Herramientas, que define si se gestionaran los útiles bajo condiciones de seguridad, true se gestiona con seguridad, false no se gestiona con seguridad, estas condiciones de seguridad estarán declaradas en esta clase y son gestionadas por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen elementos de seguridad en el parámetro de datos entrante .

private static $lbPropietarioEntidad = true ;
es un boolean perteneciente a la sección de seguridad de esta clase Herramientas, que define si al ingresar entidades herramientas en esta clase, se registra el proceso que ingresa la entidad herramienta como su propietario, esto tiene como objetivo que la opción de eliminar entidades herramientas sea posible solo por su propietario o por cualquier proceso que decida eliminar una entidad herramienta del ámbito de gestión de esta clase, true se registra el proceso propietario, false no se registra el proceso propietario, este elemento de seguridad es gestionado por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen parámetros de seguridad en el parámetro de datos entrante. No está afectado por la propiedad

$lbGestionSeguridad ya que aunque este no esté activado, con valor true, si $lbPropietarioEntidad si está activado, con valor true, se lleva a cavo su función y objetivo. El proceso núcleo siempre tiene permiso de eliminar una entidad herramienta determinada.

private static $lsAmbitoSeguridad = null ;
es un string perteneciente a la sección de seguridad de esta clase Herramientas, que define el ámbito de seguridad a aplicar a la lista de entidades herramientas declaradas a permitir o denegar su gestión por parte de esta clase Herramientas, sus valores posibles son “permisivo” y ” restrictivo” , está directamente relacionada con el elemento de seguridad $laDatosSeguridad que se expondrá más adelante, el valor “permisivo” permite la gestión de cualquier entidad herramienta excepto las declaradas en $laDatosSeguridad, el valor “restrictivo” permite la gestión solo de las entidades herramientas declaradas en $laDatosSeguridad y deniega las demás entidades herramientas, este elemento de seguridad es gestionado por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen parámetros de seguridad en el parámetro de datos entrante. Esta propiedad está afectada por la propiedad $lbGestionSeguridad, si no está activada la propiedad $lbGestionSeguridad, con valor true, se omite la existencia y valor de esta propiedad $lsAmbitoSeguridad.

private static $lbActualizarDatosSeguridad = null ;
es un boolean perteneciente a la sección de seguridad de esta clase Herramientas, que define la posibilidad o no de que los procesos puedan actualizar la lista de entidades herramientas declaradas a permitir o denegar su gestión por parte de esta clase Herramientas. sus valores posibles son “true” y ” false” , está directamente relacionada con el elemento de seguridad $laDatosSeguridad que se expondrá más adelante, el valor “true” permite que cualquier proceso actualice la lista de entidades herramientas declaradas a permitir o denegar su gestión por parte de esta clase Herramientas, false no permite lo anterior, el proceso núcleo si tiene permisos para actualizar la lista antes mencionada independientemente del valor de esta propiedad , este elemento de seguridad es gestionado por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen parámetros de seguridad en el parámetro de datos entrante. Esta propiedad está afectada por la propiedad $lbGestionSeguridad, si no está activada la propiedad $lbGestionSeguridad, con valor true, se omite la existencia y valor de esta propiedad $lbActualizarDatosSeguridad.

private static $laDatosSeguridad = array() ;
es un array perteneciente a la sección de seguridad de esta clase Herramientas, que define la lista de entidades herramientas declaradas a permitir o denegar su gestión por parte de esta clase Herramientas, está directamente relacionada con las propiedades $lsAmbitoSeguridad y $lbActualizarDatosSeguridad de esta clase Herramientas. Este elemento de seguridad es a nivel de clase y no de entidades herramientas específicas, se da permiso o no a la clase, no a una entidad correspondiente a la clase. Este elemento de seguridad es gestionado por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen parámetros de seguridad en el parámetro de datos entrante. Esta propiedad está afectada por la propiedad $lbGestionSeguridad, si no está activada la propiedad $lbGestionSeguridad, con valor true, se

omite la existencia y valor de esta propiedad $laDatosSeguridad. Existe la posibilidad de poner el valor * (asterisco) que significa todos los elementos posibles, se puede hacer a nivel de $laDatosSeguridad o a nivel de un namespace. A continuación se muestra la estructura de este arreglo:

$laDatosSeguridad:
namespace_entidades_herramientas 1:
clase_entidades_herramientas 1
clase_entidades_herramientas 2
clase_entidades_herramientas N
namespace_entidades_herramientas 2: *
namespace_entidades_ herramientas N:
clase_entidades_herramientas 1
clase_entidades_herramientas 2
clase_entidades_herramientas N

también $laDatosSeguridad puede tener el valor *

$laDatosSeguridad: *

namespace_entidades_herramientas (string) es el nombre del namespace al que pertenecen las entidades herramientas. (obligatorio)

clase_entidades_herramientas (string) es el nombre de la clase a la que pertenecen las entidades herramientas. (obligatorio)

$lsPathDirRaizEntidades = null ;
propiedad para contener el valor del path del directorio base donde se alojarán las entidades herramientas en el sistema Sockeletom, es el directorio que el sistema propone como contenedor de las entidades herramientas que se utilicen por el núcleo del sistema y los demás códigos clientes del Sockeletom. Este parámetro tiene a su vez como path raíz al directorio esquelemod, perteneciente a la estructura de directorios del sistema Sockeletom. Tomando como ejemplo el valor ¨ herramientas ¨ para esta propiedad, el valor absoluto del path donde se alojaran las entidades herramientas sería: raíz_sistema_Sockeletom/esquelemod/herramientas

private static $iniciacion = null ;
propiedad para chequear la iniciación de esta clase

private static $laEntidades = array() ;
es un arreglo que contendrá las entidades herramientas y sus características operacionales, entre estas características están:

'namespace': (string) es el nombre del namespace al que pertenece la entidad herramienta (obligatorio)

'clase': (string) es el nombre de la clase a la que pertenece la entidad herramienta (obligatorio)

'path_entidad_clase': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece la entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en esta clase la entidad herramienta a gestionar, si el fichero no existe no se ingresa la entidad herramienta en esta propiedad $laEntidades de esta clase. (obligatorio)

'tipo_entidad': (string) contiene el tipo de entidad, es decir si es de tipo clase u objeto la entidad herramienta a gestionar, sus posibles valores son ‘clase’ o ‘objeto. (obligatorio)

'iniciacion': (string) es la forma como se iniciará una entidad herramienta, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades herramientas al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades herramientas le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades herramientas le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad herramienta sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad herramienta del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento ['instancias'] [id_herramienta]['parámetros_iniciacion'] de la clase base, por lo que cada entidad herramienta puede tener sus valores propios, excepto en caso de que la entidad herramienta sea de tipo clase (tipo_entidad = 'clase') que con el primer id_herramienta se inicia la clase base. (opcional)

'proceso_propietario': (string) es el proceso que hace el ingreso de la clase entidad herramienta, este valor el procedimiento lo toma del objeto EEoNucleo conformado por el GEDEE del proceso en ejecución, seguido de :: y el id del proceso en ejecución (self::$EEoNucleo->gedeeProcesoEjecucion . '::' . self::$EEoNucleo->idProcesoEjecucion()) y es para definir el único proceso que puede eliminar la entidad namespace/clase de este arreglo, este elemento está sujeto a la propiedad $lbPropietarioEntidad de la sección de seguridad de esta clase Herramientas. Cuando el proceso propietario de la entidad namespace/clase decide eliminar esta entidad, se eliminará junto con todas las instancias contenidas de esta namespace/clase.

'instancias': (array) contiene las instancias que se creen de la clase correspondiente, contiene las instancias y los datos correspondientes a cada instancia, entre estos datos están:

'id_entidad_herramienta': (string) es el nombre o identificador de la entidad herramienta (obligatorio), los id_entidad_herramienta pueden ser ingresados en el momento que se ingresa el namespace/clase de la clase entidad_herramienta o posteriormente haciendo una petición de ingreso de instancia de una entidad_herramienta a través del procedimiento * ingresarEntidad * de esta clase.

'registro': (boolean) en caso de 'tipo_entidad' tener valor 'clase' entonces esta propiedad contiene el valor true si ha sido solicitada al menos una vez.

'objeto': (&obj) en caso de 'tipo_entidad' tener valor 'objeto' entonces esta propiedad contiene la referencia al objeto, entidad herramienta a gestionar.

'proceso_propietario': (string) es el proceso que hace el ingreso de la entidad herramienta, este valor el procedimiento lo toma del objeto EEoNucleo conformado por el gedee del proceso en ejecución, seguido de :: y el id del proceso en ejecución (self::$EEoNucleo->gedeeProcesoEjecucion . '::' . self::$EEoNucleo->idProcesoEjecucion()) y es para definir el único proceso que puede eliminar la entidad instancia de este arreglo, este elemento está sujeto a la propiedad $lbPropietarioEntidad de la sección de seguridad de esta clase Herramientas. Se aplica solo a la entidad herramienta específica, no tiene que ser el mismo propietario que el de la entidad namespace/clase de esta entidad Herramientas.

'datos': es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad herramienta en esta clase Herramientas; estos datos se crean en el momento del ingreso de la entidad herramienta sin que permita cambios posteriores.

la estructura del arreglo quedaría así:

```
$laEntidades = array(
                    'nombre nameespace' => array(
                                        'nombre clase' => array(
                                                            'path_entidad_clase' => 'path_entidad_clase',
                                                            'tipo_entidad' => 'tipo_entidad',
                                                            'iniciacion' => 'procedimiento'
                                                            'proceso_propietario' => 'proceso_que_ingresa_la_entidad',
                                                            'instancias' => array(
                                                                        'id_entidad_herramienta1' => array(
                                                                                    'registro' => 'true o false,
                                                                                    'objeto' => &objeto,
                                                                                    'proceso_propietario' => 'proceso_que_ingresa_la_entidad',
                                                                                    'datos' => array( 'dato1' => 'valor_dato' , 'datoN' =>datoN')
                                                                                        ) ,
                                                                        'id_entidad_herramientaN' => array(
                                                                                    'registro' => 'true o false,
                                                                                    'objeto' => &objeto,
                                                                                    'proceso_propietario' => 'proceso_que_ingresa_la_entidad',
                                                                                    'datos' => array( 'dato1' => 'valor_dato' , 'datoN' =>datoN')
                                                                                        )
                                                                            )
                                                                    )
                                                        )
```

)

Procedimientos:

public static function iniciacion($La_configuracion_nucleo_herramientas)
es el procedimiento encargado de iniciar el funcionamiento de esta clase y además guardar en las propiedades de esta clase los elementos existentes como valores de la sección herramientas propuestas por el sistema Sockeletom en la configuración del sistema Sockeletom. Además realiza el ingreso de las entidades herramientas propuestas a gestionar en la configuración del sistema Sockeletom. Para un mejor entendimiento de los parámetros que la configuración del sistema proporciona a este procedimiento, debe consultarse la sección herramientas del fichero de configuración del sistema Sockeletom en este documento.

este procedimiento retorna null si ya estaba inicializada la clase Herramientas , retorna true si la gestión es exitosa, detiene el proceso php si los parámetros son incompatibles con el procedimiento, emitiendo un error fatal (exit).

public static function pathRaizEntidades ()
es el procedimiento encargado de brindar, retornar el path del directorio raíz para los recursos de herramientas en el sistema Sockeletom. Este será el fragmento de path que completará el path del directorio base de los recursos de herramientas, del directorio que el sistema propone como contenedor de los recursos de herramientas que se utilicen por el núcleo del sistema y los demás códigos clientes del Sockeletom. Este parámetro tiene además como path raíz al directorio esquelemod, pertenecientes a la estructura de directorios del sistema Sockeletom. Tomando como ejemplo el valor ¨ herramientas ¨, quedará como resultado el path:
raíz_sistema_Sockeletom/esquelemod/herramientas

este procedimiento retorna el path gestionado o null en caso de no existir el path a gestionar.

public static function actualizarDatosSeguridadEntidades ($La_datos , $Ls_tipo_actualizacion)
es el procedimiento encargado de gestionar la actualización de la propiedad self:: $laDatosSeguridad de esta clase Herramientas, la gestión de actualización está sujeta al valor de la propiedad self::$lbActualizarDatosSeguridad perteneciente a la sección de seguridad de esta clase Herramientas. Los parámetros se explican a continuación:

$La_datos: (array) es el arreglo con los datos a actualizar.

$Ls_tipo_actualizacion: (string) es el tipo de actualización a realizar, sus posibles valores son:

'renovar': es la sustitución del arreglo $La_datos por el arreglo actual en self::$laDatosSeguridad.

'adicionar': es la adición del contenido del arreglo $La_datos en el arreglo actual self:: $laDatosSeguridad.

'eliminar': es la eliminación del contenido del arreglo $La_datos en el arreglo actual self:: $laDatosSeguridad.

este procedimiento retorna true si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

public static function permisionEntidad ($Ls_namespace , $Ls_clase)
es el procedimiento encargado de gestionar si una clase entidad herramienta determinada tiene permiso de ser gestionado por esta clase Herramientas, el permiso es a nivel de clase y no de entidades específicas de esta, se da permiso o no a la clase no a una entidad correspondiente a la clase; la gestión de chequeo de permisos está sujeto al valor de las propiedades self:: $lbGestionSeguridad, self::$lsAmbitoSeguridad y self::$laDatosSeguridad pertenecientes a la sección de seguridad de esta clase Herramientas; específicamente chequea las clases permitidas o denegadas en self::$laDatosSeguridad según el self::$lsAmbitoSeguridad declarado. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la clase entidad herramienta a chequear su permiso de gestión (obligatorio)

$Ls_clase: (string) es el nombre de la clase a la que pertenece la clase entidad herramienta a chequear su permiso de gestión (obligatorio)

este procedimiento retorna true si la clase entidad herramienta tiene permiso a ser gestionada por esta clase, de lo contrario retorna null.

public static function gestionControlIngresoEntidad ($Ls_path_control_herramienta , $Ls_referencia_path_control = 'relativo' , $Ls_namespace = null , $Ls_clase = null , $Ls_path_entidad_clase = null , $Ls_referencia_path_entidad = null , $Ls_tipo_entidad = null , $La_instancias = null , $Ls_iniciacion = null)
es el procedimiento encargado de gestionar el ingreso de una entidad herramienta al contenedor $laEntidades de esta clase, utilizando entre sus parámetros un parámetro que hace referencia al path de un fichero control.

Se llama fichero control a un fichero o script php que se necesita su ejecución como parte de la gestión de ingreso de la entidad herramienta a esta clase Herramientas. El control puede utilizarse para chequear, gestionar o poner a punto algún elemento necesario para el ingreso de la entidad herramienta a esta clase.

El fichero control es ejecutado (require_once) por este procedimiento y puede utilizarse en conjunto con los demás parámetros de este procedimiento o gestionar el mismo la inclusión

(require_once) de la clase entidad herramienta y retornar el mismo los parámetros que necesita este procedimiento para la gestión del ingreso de la entidad herramienta a esta clase.

Las diferentes combinaciones que pueden utilizarse devienen diferentes funcionalidades que son:

Funcionalidad 1 donde el control debe ejecutarse y las características de la entidad se pasan como parámetros independientes del control. Si se declaran parámetros para el control y a la vez se declaran los parámetros
$Ls_namespace, $Ls_clase, $Ls_path_entidad_clase, $Ls_referencia_path_entidad, $Ls_tipo_entidad , $La_instancias = null , $Ls_iniciacion este procedimiento incluirá el fichero control y además utilizará los parámetros antes mencionados para la gestión de ingreso de la entidad herramienta , no importa que el control retorne también los parámetros antes mencionados porque este procedimiento da prioridad a los valores de sus parámetros y nunca evalúa los parámetros homólogos retornados por el control. Tienen que existir todos los parámetros antes mencionados para que esta opción se ejecute, si solo uno de estos parámetros es vacío entonces se pasa a las siguientes funcionalidades.

antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de la entidad herramienta, si no existe el fichero el procedimiento no ingresa la entidad herramienta y retorna null. Ingresar una entidad herramienta no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de esta entidad herramienta. Los parámetros se explican a continuación:

$Ls_path_control_herramienta : (string) es el path del fichero control a ejecutar para el ingreso de la entidad herramienta. (obligatorio)

$Ls_referencia_path_control: (string) es la forma que se hace referencia a $Ls_path_control_herramienta, tiene tres posibles valores, *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el $Ls_path_control_herramienta relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el $Ls_path_control_herramienta relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el $Ls_path_control_herramienta de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad herramienta.

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad herramienta.

$Ls_path_entidad_clase: (string) contiene el path del fichero que alberga el código de la clase a la que pertenece la entidad, el fichero que representa este valor ya es verificado que existe cuando se

ingresa en el contenedor $laEntidades de esta clase, si el fichero no existe no se ingresa la entidad herramienta al arreglo.

$Ls_referencia_path_entidad: (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el $Ls_path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el $Ls_path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el $Ls_path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

$Ls_tipo_entidad: (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

$Ls_iniciacion: (string) es la forma como se iniciará cada id_entidad_herramienta, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de $Ls_iniciacion son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades herramientas al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades herramientas le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades herramientas le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad herramientas sea de tipo clase ($Ls_tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad herramienta del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento [$La_instancias][id_entidad_herramienta]['parametros_iniciacion'] de la clase base, por lo que cada entidad herramienta puede tener sus valores propios, excepto en caso de que la entidad herramienta sea de tipo clase ($Ls_tipo_entidad = 'clase') que con el primer id_entidad_herramienta se inicia la clase base. Los elementos [$La_instancias] [id_entidad_herramienta]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

$La_instancias (array) es un arreglo con los identificadores de entidades herramientas a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_herramienta. (obligatorio)

id_entidad_herramienta (string) es un nombre o identificador de cada entidad herramienta instancia de la clase correspondiente. y a su vez este id_entidad_herramienta es un arreglo asociativo al que se puede insertar una llave de nombre 'parametros_iniciacion' y otra de nombre 'datos',

posteriormente este arreglo id_entidad_herramienta se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_herramienta como sea necesario ya que la entidad herramienta puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad herramienta (id_entidad_herramienta), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad herramienta a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad_herramienta declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad herramienta es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad herramienta en esta clase Herramientas; estos datos se crean en el momento del ingreso de la entidad herramienta sin que permita cambios posteriores. (opcional)

Para el resto de las funcionalidades. Es necesario volver a aclarar que si se declaran parámetros en este procedimiento para el fichero control y al menos uno de los restantes es vacío entonces este procedimiento ejecuta (require_once) el control, desecha los demás parámetros pasados a este procedimiento y espera en forma de retorno(return) desde el fichero control requerido(require_once), un arreglo asociativo con las posibles siguientes estructuras:

Funcionalidad 2 donde el control debe ejecutarse, y este mismo ya incluyó la clase y nos retorna las características de la entidad incluido la o las instancias(id_entidad_herramienta)(objeto) de la clase, para que se ejecute esta funcionalidad esta estructura tiene que estar completa, si faltase algún elemento no se ejecutará esta funcionalidad, también tiene como condición que el elemento $La_resultado_require['referencia_path_entidad'] no exista, la responsabilidad de incluir la estructura correcta de los id_entidad_herramienta instanciadas (objeto) o registradas(clase) es responsabilidad del control, también es obligatorio que exista al menos una id_entidad_herramienta en la estructura de datos que se espera por retorno. La estructura que se espera por retorno es la siguiente:

$La_resultado_require['namespace'] (obligatorio)
$La_resultado_require['clase'] (obligatorio)
$La_resultado_require['path_entidad_clase'] (obligatorio)
$La_resultado_require['tipo_entidad'] (obligatorio)
$La_resultado_require['iniciacion'] (opcional)

$La_resultado_require['instancias'] (obligatorio)
$La_resultado_require['instancias'] [id_entidad_herramienta '] (obligatorio)
$La_resultado_require['instancias'] [id_entidad_herramienta '] ['objeto'] (opcional)
$La_resultado_require['instancias'] [id_entidad_herramienta] ['registro'] (opcional)
$La_resultado_require['instancias'] [id_entidad_herramienta] [' datos'] (opcional)

'namespace': (string) es el nombre del namespace al que pertenece la entidad herramienta. (obligatorio)

'clase': (string) es el nombre de la clase a la que pertenece la entidad herramienta. (obligatorio)

'path_entidad_clase': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad herramienta al arreglo $laEntidades de esta clase. (obligatorio)

'tipo_entidad': (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

'iniciacion': (string) es la forma como se iniciará cada id_entidad_herramienta , este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades herramientas al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades herramientas le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades herramientas le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad herramienta sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad herramienta del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento ['instancias'][id_entidad_herramienta]['parametros_iniciacion'] de la clase base, por lo que cada entidad herramienta puede tener sus valores propios, excepto en caso de que la entidad herramienta sea de tipo clase (tipo_entidad = 'clase') que con el primer id_entidad_herramienta se inicia la clase base. Los elementos ['instancias'][id_entidad_herramienta]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

'instancias' (array) es un arreglo con los identificadores de entidades herramientas a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_herramienta. (obligatorio)

id_entidad_herramienta (string) es un nombre o identificador de cada entidad herramienta instancia de la clase correspondiente. y a su vez este id_entidad_herramienta es un arreglo asociativo al que se puede insertar una llave de nombre objeto, una llave de nombre registro, y otra de nombre datos, las llaves objeto, y registro no deben coexistir ya que la existencia de una determina la no existencia de la otra ,posteriormente este arreglo id_entidad_herramienta se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_herramienta como sea necesario ya que la entidad herramienta puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'objeto': (&obj) es un apuntador al objeto instancia de la clase entidad herramienta a contener en el contenedor $laEntidades de esta clase. Este elemento es obligatorio si el elemento tipo_entidad tiene como valor el string objeto y estrictamente se considera iniciado.

'registro': (boolean true) es para definir cuando una entidad herramienta a contener en el contenedor $laEntidades de esta clase, y de tipo_entidad clase (procedimientos estáticos) es iniciada . Este elemento es obligatorio si el elemento tipo_entidad tiene como valor el string clase y estrictamente se considera iniciada.

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad herramienta en esta clase Herramientas; estos datos se crean en el momento del ingreso de la entidad herramienta sin que permita cambios posteriores. (opcional)

Funcionalidad 3 donde el control debe ejecutarse, y este nos retorna(return) los elementos o parámetros para ingresar una clase, y reserva al menos un id_entidad_herramienta para su posterior instanciación, no debe tener id_entidades_herramientas instanciadas(objeto) o registradas(clase), y es obligatorio que exista en los parámetros retornados por el control el elemento $La_resultado_require['referencia_path_entidad']. Entiéndase que esta funcionalidad solo realiza su gestión si no existe ningún id_entidad_herramienta instanciado este factor es obligatorio. La estructura que se espera por retorno es la siguiente:

$La_resultado_require['namespace'] (obligatorio)
$La_resultado_require['clase'] (obligatorio)
$La_resultado_require['path_entidad_clase'] (obligatorio)
$La_resultado_require['referencia_path_entidad '] (obligatorio)
$La_resultado_require['tipo_entidad'] (obligatorio)
$La_resultado_require['iniciacion'] (opcional)
$La_resultado_require['instancias'] (obligatorio)
$La_resultado_require['instancias'] [id_entidad_herramienta] (obligatorio)
$La_resultado_require['instancias'] [id_entidad_herramienta] ['parametros_iniciacion'] (opcional)
$La_resultado_require['instancias'] [id_entidad_herramienta] [' datos'] (opcional)

'namespace': (string) es el nombre del namespace al que pertenece la entidad herramienta. (obligatorio)

'clase': (string) es el nombre de la clase a la que pertenece la entidad herramienta. (obligatorio)

'path_entidad_clase': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad herramienta al arreglo $laEntidades de esta clase. (obligatorio)

'referencia_path_entidad': (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

'tipo_entidad': (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

'iniciacion': (string) es la forma como se iniciará cada id_entidad_herramienta, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades herramientas al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades herramientas le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades herramientas le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad herramienta sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad herramienta del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento ['instancias'][id_entidad_herramienta]['parametros_iniciacion'] de la clase base, por lo que cada entidad herramienta puede tener sus valores propios, excepto en caso de que la entidad herramienta sea de tipo clase (tipo_entidad = 'clase') que con el primer id_entidad_herramienta se inicia la clase base. Los elementos ['instancias'][id_entidad_herramienta]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

'instancias' (array) es un arreglo con los identificadores de entidades herramientas a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_herramienta. (obligatorio)

id_entidad_herramienta (string) es un nombre o identificador de cada entidad herramienta instancia de la clase correspondiente. y a su vez este id_entidad_herramienta es un arreglo asociativo al que se puede insertar una llave de nombre 'parametros_iniciacion' y otra de nombre 'datos', posteriormente este arreglo id_entidad_herramienta se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_herramienta como sea necesario ya que la entidad herramienta puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad herramienta (id_entidad_herramienta), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad herramienta a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad_herramienta declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad herramienta es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad herramienta en esta clase Herramientas; estos datos se crean en el momento del ingreso de la entidad herramienta sin que permita cambios posteriores. (opcional)

La funcionalidad que el control ejecutado devuelva los datos de la clase y sus elementos['instancias'][id_entidad_ herramienta] no instanciados, es por seguridad ya que existen otras formas para crear controles que retornen estructuras con sus id_entidad_ herramienta ya instanciados(objeto) o registrados(clase); Esta característica está abierta a cambios futuro en dependencia de la opinión mayoritaria de los usuarios del sistema Sockeletom.

antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de la entidad herramienta, si no existe el fichero el procedimiento no ingresa la entidad herramienta y retorna null. Ingresar una entidad herramienta no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de esta entidad herramienta.

este procedimiento retorna true si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

public static function gestionIngresoEntidad ($Ls_namespace , $Ls_clase , $Ls_path_entidad_clase , $Ls_referencia_path_entidad , $Ls_tipo_entidad , $La_instancias , $Ls_iniciacion = null)
es el procedimiento encargado de gestionar el ingreso de una entidad herramienta al contenedor $laEntidades de esta clase, antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de la entidad herramienta, si no existe el fichero el procedimiento no ingresa la entidad herramienta y retorna null. Ingresar una entidad herramienta no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de esta entidad herramienta. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad herramienta. (obligatorio)

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad herramienta. (obligatorio)

$Ls_path_entidad_clase: (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad herramienta al arreglo $laEntidades de esta clase. (obligatorio)

$Ls_referencia_path_entidad: (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

$Ls_tipo_entidad: (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

$Ls_iniciacion: (string) es la forma como se iniciará cada id_entidad_herramienta, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades herramientas al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades herramientas le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades herramientas le pasa los parámetros con los

que iniciará la instancia. En caso de que la entidad herramienta sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad herramienta del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento $La_instancias[id_entidad_herramienta]['parametros_iniciacion'] de la clase base, por lo que cada entidad herramienta puede tener sus valores propios, excepto en caso de que la entidad herramienta sea de tipo clase (tipo_entidad = 'clase') que con el primer id_ entidad_herramienta se inicia la clase base. Los elementos $La_instancias[id_entidad_herramienta]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

$La_instancias (array) es un arreglo con los identificadores de entidades herramientas a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_herramienta. (obligatorio)

id_entidad_herramienta (string) es un nombre o identificador de cada entidad herramienta instancia de la clase correspondiente. y a su vez este id_entidad_herramienta es un arreglo asociativo al que se puede insertar una llave de nombre 'parametros_iniciacion' y otra de nombre 'datos', posteriormente este arreglo id_entidad_herramienta se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_herramienta como sea necesario ya que la entidad herramienta puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad herramienta (id_entidad_herramienta), si existe este parámetro, es arreglo y no es vacio entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad herramienta a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad_herramienta declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad herramienta es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad herramienta en esta clase Herramientas; estos datos se crean en el momento del ingreso de la entidad herramienta sin que permita cambios posteriores. (opcional)

public static function gestionIngresosEntidades ($La_propiedades_herramientas)
es el procedimiento encargado de gestionar el ingreso de varias entidades herramientas al contenedor $laEntidades de esta clase, este procedimiento desglosa un arreglo con datos de varias entidades herramientas y realiza la gestión de ingreso según sus características como lo hacen individualmente los procedimientos self::gestionControlIngresoEntidad y self::gestionIngresoEntidad, es importante aclarar que cada entidad herramienta será procesada estrictamente con los procedimientos self::gestionControlIngresoEntidad o self::gestionIngresoEntidad por lo que los datos de las entidades herramientas tienen que cumplir con las condicionantes del procedimiento respectivo para que la gestión sea exitosa, antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de cada entidad herramienta, si no existe el fichero el procedimiento no ingresa la entidad herramienta y continua su gestión. Ingresar una entidad herramienta no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de esta entidad herramienta. Los parámetros se explican a continuación:

$La_propiedades_herramientas: es un arreglo en el que los elementos llave de este arreglo son los namespaces de las entidades herramientas a gestionar su ingreso en esta clase, la llave namespace representa a su vez un arreglo con las clases y algunas características de estas clases que finalmente son las entidades herramientas declaradas, la llave clase representa a su vez un arreglo con algunas características necesarias para la correcta gestión de las entidades herramientas, un ejemplo de esta estructura se expone a continuación:

```
$La_propiedades_herramientas
                    namespace_entidad_herramienta1:
                                clase_entidad_herramienta 1:
                                            'path_entidad_clase':
                                            'referencia_path_entidad':
                                            'path_control':
                                            'referencia_path_control':
                                            'tipo_entidad':
                                           'iniciación':
                                            'instancias':
                                                        id_entidad_herramienta 1:
                                                                    'parametros_iniciacion':
                                                                    'datos':
                                                        id_entidad_herramienta 2
                                                                    'parametros_iniciacion':
                                                        id_entidad_herramienta 3
                                                        id_entidad_herramienta 4:
                                                                    'datos':

                    namespace_entidad_herramienta 2:
```

```
clase_entidad_herramienta 1:
        'path_entidad_clase':
        'referencia_path_entidad':
        'path_control':
        'referencia_path_control':
        'tipo_entidad':
        'instancias':
                id_entidad_herramienta 1:
                        'datos':
                id_entidad_herramienta 2
                id_entidad_herramienta 3
                id_entidad_herramienta 4
clase_entidad_herramienta 2:
        'path_entidad_clase':
        'referencia_path_entidad':
        'path_control':
        'referencia_path_control':
        'tipo_entidad':
        'iniciación':
        'instancias':
                id_entidad_herramienta 1:
                        'datos':
                id_entidad_herramienta 2:
                        'parametros_iniciacion':
                        'datos':
                id_entidad_herramienta 3:
                        'datos':
                id_entidad_herramienta 4:
                        'datos':
```

los elementos de este arreglo se explican a continuación:

namespace: (string) es el nombre del namespace al que pertenece la entidad herramienta. (obligatorio en dependencia del procedimiento gestor)

clase: (string) es el nombre de la clase a la que pertenece la entidad herramienta. (obligatorio en dependencia del procedimiento gestor)

'path_entidad_clase': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad herramienta al arreglo $laEntidades de esta clase. (obligatorio en dependencia del procedimiento gestor)

'referencia_path_entidad': (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio en dependencia del procedimiento gestor)

'path_control': es el path de un fichero que controlará condiciones y/o el inicio o inclusión de una entidad herramienta, el gestor de herramientas espera un retorno de este control con un arreglo asociativo donde se encuentren los datos namespace_entidad_herramienta, clase_entidad_herramienta, id_entidad _herramienta, path_entidad_clase, referencia_path_entidad , tipo_entidad , instancias con sus datos, o combinaciones de estos, este arreglo será procesado por esta clase Emod\\Nucleo\Herramientas que es quien gestiona y administra los herramientas en el sistema Sockeletom. (obligatorio en dependencia del procedimiento gestor)

'referencia_path_control': (string) es la forma que se hace referencia a $Ls_path_control_herramienta, tiene tres posibles valores, *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el $Ls_path_control_herramienta relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el $Ls_path_control_herramienta relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el $Ls_path_control_herramienta de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio en dependencia del procedimiento gestor)

'tipo_entidad': (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio en dependencia del procedimiento gestor)

'iniciacion': (string) es la forma como se iniciará cada id_entidad_herramienta, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades herramientas al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades herramientas le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades herramientas le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad herramienta sea de tipo clase (tipo_entidad =

'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad herramienta del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento ['instancias'][id_entidad_herramienta]['parametros_iniciacion'] de la clase base, por lo que cada entidad herramienta puede tener sus valores propios, excepto en caso de que la entidad herramienta sea de tipo clase (tipo_entidad = 'clase') que con el primer id_entidad_herramienta se inicia la clase base. Los elementos ['instancias'][id_entidad_herramienta]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional en dependencia del procedimiento gestor)

'instancias': (array) es un arreglo con los identificadores de entidades herramientas a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_herramienta. (obligatorio en dependencia del procedimiento gestor)

id_entidad_herramienta (string) es un nombre o identificador de cada entidad herramienta instancia de la clase correspondiente. y a su vez este id_entidad_herramienta es un arreglo asociativo al que se puede insertar una llave de nombre parametros_iniciacion y otra de nombre datos, posteriormente este arreglo id_entidad_herramienta se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_herramienta como sea necesario ya que la entidad herramienta puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1 en dependencia del procedimiento gestor) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad herramienta (id_entidad_herramienta), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad herramienta a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad_herramienta declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad herramienta es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional en dependencia del procedimiento gestor)

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad herramienta en esta clase Herramientas; estos datos se crean en el momento del ingreso de la entidad herramienta sin que permita cambios posteriores. (opcional en dependencia del procedimiento gestor)

se aconseja por parte del grupo de desarrollo del núcleo Sockeletom que los namespaces se declaren de forma absoluta, comenzando con '\ '

este procedimiento retorna true si la gestión de al menos una entidad herramienta fue llevada a cabo con éxito, de lo contrario retorna null.

public static function existenciaEntidad ($Ls_namespace , $Ls_clase , $Ls_id_herramienta = null)
es el procedimiento encargado de gestionar si existe una entidad herramienta determinada en el contenedor $laEntidades de esta clase, si existe una entidad herramienta a disposición de pedido y uso, esta entidad herramienta no tiene obligatoriamente que estar iniciada, es decir no tiene obligatoriamente que existir la instancia ya creada.

Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad herramienta. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad herramienta. (obligatorio).

$Ls_id_herramienta: (string) es el nombre o id de la entidad herramienta hija de una clase ya ingresada en $this->laEntidades, este parámetro puede ser vacío, de ser vacío se entiende que se busca la existencia de una clase(\namespace\clase) ya ingresada en $this->laEntidades, clase base para entidades herramientas instancias(objeto) o clase (clase de procedimientos estáticos).

Si $Ls_id_herramienta tiene valor y existe como entidad herramienta hija de la \namespace\clase base en la estructura de datos de $this->laEntidades entonces este procedimiento retorna el tipo_entidad (‘objeto’ o ‘clase’) de la clase base. Si no existe como entidad herramienta hija el procedimiento retorna null.

Si $Ls_id_herramienta no tiene valor, o tiene valor vacío, existe la entidad herramienta \namespace\clase base en la estructura de datos de $this->laEntidades entonces este procedimiento retorna el tipo_entidad (‘objeto’ o ‘clase’) de la clase base. Si no existe como la entidad herramienta \namespace\clase base el procedimiento retorna null.

public static function ingresarEntidad ($Ls_namespace , $Ls_clase , $Ls_id_herramienta , $La_parametros_iniciacion = null)
es el procedimiento encargado de gestionar el ingreso de entidades herramientas hija de la \namespace\clase base en la estructura de datos de $this->laEntidades. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad herramienta, es obligatorio que exista ya en $this->laEntidades. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad herramienta, es obligatorio que exista ya en $this->laEntidades. (obligatorio).

$Ls_id_herramienta: (string) es el nombre o id de una entidad herramienta a ingresar en la estructura instancias de la clase base \namespace\clase en la estructura de datos de $this->laEntidades(obligatorio).

$La_parametros_iniciacion: (array) cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad herramienta ($Ls_id_herramienta), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad herramienta a gestionar sea de tipo clase (tipo_entidad = 'clase') no tiene efecto este parámetro ya que se gestionó en el primer elemento $Ls_id_herramienta que se ingreso en la clase base. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

public static function entidad ($Ls_namespace , $Ls_clase , $Ls_id_herramienta , $Ls_tipo_referencia = 'referencia')
es el procedimiento encargado de gestionar la entidad herramienta hija de la \namespace\clase base en la estructura de datos de $this->laEntidades identificada por $Ls_id_herramienta, es decir si la entidad es de tipo ‘objeto’ retronará la referencia a un objeto instancia de la entidad herramienta, si la entidad es de tipo clase retornará un string compuesto por el namespace y nombre de la clase base. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad herramienta. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad herramienta. (obligatorio).

$Ls_id_herramienta: (string) es el nombre o id de la entidad herramienta hija de una clase ya ingresada en $this->laEntidades, el string en $Ls_id_herramienta tiene obligatoriamente que existir como elemento en la estructura instancias de la clase base en $this->laEntidades . (obligatorio).

$Ls_tipo_referencia: (string) es el tipo de referencia a objeto que devuelve el procedimiento en caso de que el tipo de entidad sea ‘objeto‘, este parámetro tiene dos valores posibles ‘referencia‘ y ‘clon‘, referencia devuelve un puntero a la entidad guardada en el arreglo $this->laEntidades, y clon devuelve un clon de este objeto. Si el tipo de entidad es ‘clase‘ no se considera este parámetro. Por defecto este parámetro tiene como valor ‘referencia‘.

Este procedimiento devuelve (retorna) una referencia o clon a objeto si el tipo_entidad de la clase base es (‘objeto’); devuelve (retorna) un string de la forma \namespace\clase si el tipo_entidad de la clase base es (‘clase’); de lo contrario retorna null.

La instanciación de los objetos ocurre en el momento que es pedido por primera vez una instancia a través de este procedimiento.

En el caso de la instanciación o iniciación de entidades de tipo ‘objeto’, e iniciación de entidades de tipo ‘clase’, estas entidades pueden emitir una excepción desde su constructor o procedimiento de iniciación si existe algún problema de instanciación o iniciación, ya que este procedimiento tiene un bloque try y catch para manejar las excepciones, el bloque catch espera un objeto de la clase \Exception y emite el mensaje de este, el código es de la siguiente forma:

```
catch( \Exception $e)
  {
     echo '<p>Excepción capturada: '.$e->getMessage(). "<p>" ;
     return null;
   }
```

Por lo que el lanzamiento de la excepción sería:

```
throw new \Exception('mensaje.');
```

Esta funcionalidad es muy útil para interrumpir la instanciación de una clase si existe algún error en esta instanciación, y así no se tienen objetos con problemas o sin uso por desperfectos, en este contenedor de instancias.

Lo anterior se aplica a entidades de tipo ‘objeto’ que lancen su excepción desde el constructor, o entidades de tipo ‘clase’ que lancen su excepción desde el procedimiento de iniciación; Quedan excluidos las entidades de tipo ‘objeto’ que lancen su excepción desde un procedimiento de iniciación(no __construct) ya que estos ya fueron instanciados existen como instancias en el contenedor, estos casos solo tienen la opción de emitir un mensaje.

public static function datosEntidad ($Ls_namespace , $Ls_clase , $Ls_id_herramienta)
es el procedimiento encargado de gestionar y retornar los datos, que de existir, tiene una entidad herramienta que haya ingresado en esta clase, estos datos son los correspondientes al elemento ‘datos’ que puede traer consigo una entidad herramienta en el momento de su ingreso al contenedor $laEntidades de esta clase. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad herramienta. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad herramienta. (obligatorio).

$Ls_id_herramienta: (string) es el nombre o id de la entidad herramienta hija de una clase ya ingresada en $this->laEntidades, el string en $Ls_id_herramienta tiene obligatoriamente que existir como elemento en la estructura instancias de la clase base en $this->laEntidades . (obligatorio).

este procedimiento retorna el arreglo 'datos' perteneciente a la entidad herramienta propietaria si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

public static function idEntidades ($Ls_namespace , $Ls_clase)
es el procedimiento encargado de gestionar y retornar los nombres o identificadores de entidades herramientas que hayan ingresado en el namespace y clase correspondientes a los parámetros de este procedimiento, estos datos son los correspondientes al elemento 'instancias' que debe tener consigo cada clase base herramienta en el contenedor $laEntidades de esta clase. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad herramienta. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad herramienta. (obligatorio).

si la gestión fue llevada a cabo con éxito este procedimiento retorna las llaves del array 'instancias' contenidos en el namespace y clase, clase base herramienta en el contenedor $laEntidades de esta clase, de lo contrario retorna null.

public static function eliminarEntidad ($Ls_namespace , $Ls_clase , $Ls_id_herramienta = null)
es el procedimiento encargado de eliminar una entidad herramienta clase base, o una entidad herramienta de la estructura 'instancias' de una clase base, existentes ambas en el contenedor $laEntidades de esta clase. Este procedimiento está sujeto al valor de la propiedad self:: $lbPropietarioEntidad perteneciente a la sección de seguridad de esta clase Herramientas, si esta propiedad tiene valor true solo puede eliminar la entidad herramienta el proceso que figura como su propietario, de lo contrario puede eliminarse desde cualquier proceso, el proceso núcleo siempre tiene derechos de eliminar cualquier entidad herramienta.

Existen dos posibilidades de uso de este procedimiento:

La primera es eliminar una entidad herramienta clase base, es decir una namespace\clase registrada como clase base de entidades herramientas, para ello se pasan los valores correspondientes al namespace y clase de la entidad a eliminar, dejando vacío el parámetro $Ls_id_herramienta, recordar que esta operación está sujeto a la sección de seguridad de esta clase, donde se define quién puede eliminar una entidad determinada.

La segunda es eliminar una entidad herramienta de la estructura ‘instancias’ de una clase base, para ello debe existir la entidad herramienta en la estructura $laEntidades[$Ls_namespace][$Ls_clase] [‘instancias’] de esta clase, el parámetro $Ls_id_herramienta no puede ser vacío, recordar que esta operación está sujeto a la sección de seguridad de esta clase, donde se define quien puede eliminar una entidad determinada.

Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad herramienta. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad herramienta. (obligatorio).

$Ls_id_herramienta: (string) es el nombre o id de la entidad herramienta hija de una clase ya ingresada en $this->laEntidades, el string en $Ls_id_herramienta tiene obligatoriamente que existir como elemento en la estructura instancias de la clase base en $this->laEntidades . (obligatorio).

este procedimiento retorna true si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

class_ecro_permanente.php

Este fichero contiene la clase abstracta CroPermanente, que se encarga de la declaración de un grupo de propiedades y procedimientos en función de servir de contenedor de referencias a objetos y servidor de estos, gestiona y sirve punteros por referencia a estos objetos o clonaciones de ellos, con la peculiaridad de no permitir una vez ingresado una referencia se le elimine, a continuación explicamos la estructura y contenido de esta clase:

Namespace: Emod\Nucleo\Herramientas
Nombre de la clase: CroPermanente
Herencia:

Propiedades:
protected static $laContenedorObjetos = null ;
es un arreglo que contendrá las referencias a objetos junto con el id para referenciarlo y el namespace y class a que este objeto pertenece, la estructura quedaría así:

```
$laContenedorObjetos = array(
                                'nombre namespace' = array(
                                                'nombre clase' = array(
                                                                'id_objeto' = objeto(referencia)
                                                                )
                                                )
                                )
```

Procedimientos:

public static function ingresarNuevoObjeto ($id_objeto , &$objeto)
este procedimiento se encarga de ingresar una referencia de objeto al contenedor $laContenedorObjetos. Los parámetros se explican a continuación:

$id_objeto es el nombre o id con el que ingresará la referencia del objeto en el contenedor $laContenedorObjetos, y ese mismo nombre o id será el utilizado para obtener la referencia o clon del objeto desde el contenedor $laContenedorObjetos.

$objeto referencia a objeto.

public static function referenciarObjeto ($id_objeto , $class , $namespace)
este procedimiento se encarga de retornar una referencia a objeto de las contenidas en el contenedor $laContenedorObjetos. Los parámetros se explican a continuación:

$id_objeto es el nombre o id con el que ingresó la referencia del objeto en el contenedor $laContenedorObjetos.

$class es el nombre de la clase a la que pertenece el objeto referenciado.

$namespace es el namespace a la que pertenece el objeto referenciado.

public static function clonarObjeto ($id_objeto , $class , $namespace)
este procedimiento se encarga de clonar y retornar la referencia a la clonación del objeto, de las referencias a objetos contenidas en el contenedor $laContenedorObjetos. Los parámetros se explican a continuación:

$id_objeto es el nombre o id con el que ingresó la referencia del objeto que se quiere clonar en el contenedor $laContenedorObjetos.

$class es el nombre de la clase a la que pertenece el objeto que se quiere clonar.

$namespace es el namespace a la que pertenece el objeto que se quiere clonar.

public static function existenciaObjeto ($id_objeto , $class , $namespace)
este procedimiento se encarga de retornar true si existe un objeto de características coincidentes con los parámetros, y null si no existe un objeto de características coincidentes con los parámetros en el contenedor $laContenedorObjetos. Los parámetros se explican a continuación:

$id_objeto es el nombre o id con el que ingresó la referencia del objeto que se quiere gestionar su existencia en el contenedor $laContenedorObjetos.

$class es el nombre de la clase a la que pertenece el objeto que se quiere gestionar su existencia.

$namespace es el namespace a la que pertenece el objeto que se quiere gestionar su existencia.

class_ecro_variable.php

Este fichero contiene la clase abstracta CroVariable, que se encarga de la declaración de un grupo de propiedades y procedimientos en función de servir de contenedor de referencias a objetos y servidor de estos, gestiona y sirve punteros por referencia a estos objetos o clonaciones de ellos, a continuación explicamos la estructura y contenido de esta clase:

Namespace: Emod\Nucleo\Herramientas
Nombre de la clase: CroVariable
Herencia: extends \Emod\Nucleo\Herramientas\CroPermanente

Propiedades:
protected static $laContenedorObjetos = null ;
es un arreglo que contendrá las referencias a objetos junto con el id para referenciarlo y el namespace y class a que este objeto pertenece, la estructura quedaría así:

```
$laContenedorObjetos = array(
                        'nombre namespace' = array(
                                'nombre clase' = array(
                                        'id_objeto' = objeto(referencia)
                                        )
                                )
                )
```

Procedimientos:

public static function ingresarNuevoObjeto ($id_objeto , &$objeto)
este procedimiento se encarga de ingresar una referencia de objeto al contenedor $laContenedorObjetos. Los parámetros se explican a continuación:

$id_objeto es el nombre o id con el que ingresará la referencia del objeto en el contenedor $laContenedorObjetos, y ese mismo nombre o id será el utilizado para obtener la referencia o clon del objeto desde el contenedor $laContenedorObjetos.

$objeto referencia a objeto.

public static function referenciarObjeto ($id_objeto , $class , $namespace)
este procedimiento se encarga de retornar una referencia a objeto de las contenidas en el contenedor $laContenedorObjetos. Los parámetros se explican a continuación:

$id_objeto es el nombre o id con el que ingresó la referencia del objeto en el contenedor $laContenedorObjetos.

$class es el nombre de la clase a la que pertenece el objeto referenciado.

$namespace es el namespace a la que pertenece el objeto referenciado.

public static function clonarObjeto ($id_objeto , $class , $namespace)
este procedimiento se encarga de clonar y retornar la referencia a la clonación del objeto, de las referencias a objetos contenidas en el contenedor $laContenedorObjetos. Los parámetros se explican a continuación:

$id_objeto es el nombre o id con el que ingresó la referencia del objeto que se quiere clonar en el contenedor $laContenedorObjetos.

$class es el nombre de la clase a la que pertenece el objeto que se quiere clonar.

$namespace es el namespace a la que pertenece el objeto que se quiere clonar.

public static function existenciaObjeto ($id_objeto , $class , $namespace)
este procedimiento se encarga de retornar true si existe un objeto de características coincidentes con los parámetros, y null si no existe un objeto de características coincidentes con los parámetros en el contenedor $laContenedorObjetos. Los parámetros se explican a continuación:

$id_objeto es el nombre o id con el que ingresó la referencia del objeto que se quiere gestionar su existencia en el contenedor $laContenedorObjetos.

$class es el nombre de la clase a la que pertenece el objeto que se quiere gestionar su existencia.

$namespace es el namespace a la que pertenece el objeto que se quiere gestionar su existencia.

public static function eliminarReferenciaObjeto ($id_objeto , $class , $namespace)
este procedimiento se encarga de eliminar la referencia al objeto, de las referencias a objetos contenidas en el contenedor $laContenedorObjetos. Los parámetros se explican a continuación:

$id_objeto es el nombre o id con el que ingresó la referencia del objeto que se quiere eliminar en el contenedor $laContenedorObjetos.

$class es el nombre de la clase a la que pertenece el objeto que se quiere eliminar.

$namespace es el namespace a la que pertenece el objeto que se quiere eliminar.

class_ecrop_nucleo.php

Este fichero contiene la clase CropNucleo, que se encarga de la declaración de un grupo de propiedades y procedimientos en función de servir de contenedor de referencias a objetos que utiliza y brinda el núcleo del Sockeletom, además servir estos objetos, gestiona y sirve punteros por referencia a estos objetos, con la peculiaridad de no permitir una vez ingresado una referencia se le elimine, a continuación explicamos la estructura y contenido de esta clase:

Namespace: Emod\Nucleo
Nombre de la clase: CropNucleo
Herencia: extends \Emod\Nucleo\Herramientas\CroPermanente

Propiedades:

protected static $laContenedorObjetos = null ;
es un arreglo que contendrá las referencias a objetos junto con el id para referenciarlo y el namespace y class a que este objeto pertenece, la estructura quedaría así:

```
$laContenedorObjetos = array(
                        'nombre namespace' = array(
                                'nombre clase' = array(
                                        'id_objeto' = objeto(referencia)
                                        )
                                )
                        )
```

Procedimientos:

public static function ingresarNuevoObjeto ($id_objeto , &$objeto)
este procedimiento se encarga de ingresar una referencia de objeto al contenedor $laContenedorObjetos. Los parámetros se explican a continuación:

$id_objeto es el nombre o id con el que ingresará la referencia del objeto en el contenedor $laContenedorObjetos, y ese mismo nombre o id será el utilizado para obtener la referencia o clon del objeto desde el contenedor $laContenedorObjetos.

$objeto referencia a objeto.

public static function referenciarObjeto ($id_objeto , $class , $namespace)
este procedimiento se encarga de retornar una referencia a objeto de las contenidas en el contenedor $laContenedorObjetos. Los parámetros se explican a continuación:

$id_objeto es el nombre o id con el que ingresó la referencia del objeto en el contenedor $laContenedorObjetos.

$class es el nombre de la clase a la que pertenece el objeto referenciado.

$namespace es el namespace a la que pertenece el objeto referenciado.

public static function clonarObjeto ($id_objeto , $class , $namespace)
este procedimiento retorna null sin importar los parámetros que tiene, los parámetros los mantiene por norma con la clase de la que hereda, pero no considera la clonación por las características propias de los objetos que contendrá.

public static function existenciaObjeto ($id_objeto , $class , $namespace)
este procedimiento se encarga de retornar true si existe un objeto de características coincidentes con los parámetros, y null si no existe un objeto de características coincidentes con los parámetros en el contenedor $laContenedorObjetos. Los parámetros se explican a continuación:

$id_objeto es el nombre o id con el que ingresó la referencia del objeto que se quiere gestionar su existencia en el contenedor $laContenedorObjetos.

$class es el nombre de la clase a la que pertenece el objeto que se quiere gestionar su existencia.

$namespace es el namespace a la que pertenece el objeto que se quiere gestionar su existencia.

trait_egeco.php

Este fichero contiene el trait de procedimientos estáticos GECO (Gestor de Entidades Clases y Objetos). Este trait está compuesta por propiedades, y procedimientos estáticos para la gestión de entidades clases y objetos, desde la inclusión del script o clase como fichero hasta el retorno de referencias o clonaciones a entidades clases u objetos, gestiona y sirve como contenedor de estas entidades, una opción clara para la inyección de dependencias. En su gestión las entidades no son incluidas o instanciadas hasta el momento que se hace la petición de la instancia para su uso. Para un mejor entendimiento de cómo se gestionan las entidades se debe consultar cada propiedad y procedimiento de este trait.

A continuación mostramos una lista de las propiedades y procedimientos que este trait pone a disposición de los códigos clientes del Sockeletom:

Namespace: Emod\Nucleo\Herramientas

Nombre del trait: GECO
Herencia:

Propiedades:

private static $EEoNucleo = null ;

puntero al objeto EEoNucleo ;

private static $lbGestionSeguridad = false ;
es un boolean perteneciente a la sección de seguridad de este trait (GECO), que define si se gestionaran las entidades bajo condiciones de seguridad, true se gestiona con seguridad, false no se gestiona con seguridad, estas condiciones de seguridad estarán declaradas en este trait(GECO) y son gestionadas por el procedimiento iniciación de este trait, es en este procedimiento donde se chequea si existen elementos de seguridad en el parámetro de datos entrante .

private static $lbPropietarioEntidad = true ;
es un boolean perteneciente a la sección de seguridad de este trait (GECO), que define si al ingresar entidades en este trait, se registra el proceso que ingresa la entidad como su propietario, esto tiene como objetivo que la opción de eliminar entidades sea posible solo por su propietario o por cualquier proceso que decida eliminar una entidad del ámbito de gestión de este trait, true se registra el proceso propietario, false no se registra el proceso propietario, este elemento de seguridad es gestionado por el procedimiento iniciación de este trait, es en este procedimiento donde se chequea si existen elementos de seguridad en el parámetro de datos entrante. No está afectado por la propiedad $lbGestionSeguridad ya que aunque este no esté activado, con valor true, si $lbPropietarioEntidad si está activado, con valor true, se lleva a cavo su función y objetivo. El proceso núcleo siempre tiene permiso de eliminar una entidad determinada.

private static $lsAmbitoSeguridad = null ;
es un string perteneciente a la sección de seguridad de este trait (GECO), que define el ámbito de seguridad a aplicar a la lista de entidades declaradas a permitir o denegar su gestión por parte de este trait (GECO), sus valores posibles son “permisivo” y ” restrictivo” , está directamente relacionada con el elemento de seguridad $laDatosSeguridad que se expondrá más adelante, el valor “permisivo” permite la gestión de cualquier entidad excepto las declaradas en $laDatosSeguridad, el valor “restrictivo” permite la gestión solo de las entidad declaradas en $laDatosSeguridad y deniega las demás entidades, este elemento de seguridad es gestionado por el procedimiento iniciación de este trait, es en este procedimiento donde se chequea si existen elementos de seguridad en el parámetro de datos entrante. Esta propiedad está afectada por la propiedad $lbGestionSeguridad, si no está activada la propiedad $lbGestionSeguridad, con valor true, se omite la existencia y valor de esta propiedad $lsAmbitoSeguridad.

private static $lbActualizarDatosSeguridad = null ;
es un boolean perteneciente a la sección de seguridad de este trait (GECO), que define la posibilidad o no de que los procesos puedan actualizar la lista de entidades declaradas a permitir o denegar su gestión por parte de este trait (GECO). sus valores posibles son “true” y ” false” , está directamente relacionada con el elemento de seguridad $laDatosSeguridad que se expondrá más adelante, el valor “true” permite que cualquier proceso actualice la lista de entidades declaradas a permitir o denegar su gestión por parte de este trait (GECO), false no permite lo anterior, el proceso núcleo si tiene permisos para actualizar la lista antes mencionada independientemente del valor de esta propiedad , este elemento de seguridad es gestionado por el procedimiento iniciación de este

trait, es en este procedimiento donde se chequea si existen elementos de seguridad en el parámetro de datos entrante. Esta propiedad está afectada por la propiedad $lbGestionSeguridad, si no está activada la propiedad $lbGestionSeguridad, con valor true, se omite la existencia y valor de esta propiedad $lbActualizarDatosSeguridad.

private static $laDatosSeguridad = array() ;
es un array perteneciente a la sección de seguridad de este trait (GECO), que define la lista de entidades declaradas a permitir o denegar su gestión por parte de este trait (GECO), está directamente relacionada con las propiedades $lsAmbitoSeguridad y $lbActualizarDatosSeguridad de este trait (GECO). Este elemento de seguridad es a nivel de clase y no de entidades específicas, se da permiso o no a la clase base, no a una entidad (clase u objeto) correspondiente a la clase base. Este elemento de seguridad es gestionado por el procedimiento iniciación de este trait, es en este procedimiento donde se chequea si existen elementos de seguridad en el parámetro de datos entrante. Esta propiedad está afectada por la propiedad $lbGestionSeguridad, si no está activada la propiedad $lbGestionSeguridad, con valor true, se omite la existencia y valor de esta propiedad $laDatosSeguridad. Existe la posibilidad de poner el valor * (asterisco) que significa todos los elementos posibles, se puede hacer a nivel de $laDatosSeguridad o a nivel de un namespace. A continuación se muestra la estructura de este arreglo:

$laDatosSeguridad:
namespace_entidades1:
clase_entidades1
clase_entidades2
clase_entidadesN
namespace_entidades2: *
namespace_entidadesN:
clase_entidades1
clase_entidades2
clase_entidadesN

también $laDatosSeguridad puede tener el valor *

$laDatosSeguridad: *

namespace_entidades (string) es el nombre del namespace al que pertenecen las entidades. (obligatorio)

clase_entidades (string) es el nombre de la clase a la que pertenecen las entidades. (obligatorio)

$lsPathDirRaizEntidades = null ;
propiedad para contener el valor del path del directorio base donde se alojarán las entidades en el sistema Sockeletom, es el directorio que se propone como contenedor de las entidades que se utilicen por el núcleo del sistema y los demás códigos clientes del Sockeletom. Este parámetro tiene

a su vez como path raíz al directorio esquelemod, perteneciente a la estructura de directorios del sistema Sockeletom. Tomando como ejemplo el valor ¨ entidades ¨ para esta propiedad, el valor absoluto del path donde se alojaran las herramientas sería:
raíz_sistema_Sockeletom/esquelemod/entidades

private static $iniciacion = null ;
propiedad para chequear la iniciación de este trait

private static $laEntidades = array() ;
es un arreglo que contendrá las entidades y sus características operacionales, entre estas características están:

namespace: (string) es el nombre del namespace al que pertenece la entidad (obligatorio)

clase: (string) es el nombre de la clase a la que pertenece la entidad (obligatorio)

'path_entidad_clase': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece la entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en este trait la entidad a gestionar, si el fichero no existe no se ingresa la entidad en esta propiedad $laEntidades de este trait. (obligatorio)

'tipo_entidad': (string) contiene el tipo de entidad, es decir si es de tipo clase u objeto la entidad a gestionar, sus posibles valores son ‘clase’ o ‘objeto. (obligatorio)

'iniciacion': (string) es la forma como se iniciará la entidad, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento ['instancias'][id_entidad]['parametros_iniciacion'] de la clase base, por lo que cada entidad puede tener sus valores propios, excepto en caso de que la entidad sea de tipo clase (tipo_entidad = 'clase') que con el primer id_entidad se inicia la clase base. (opcional)

'proceso_propietario': (string) es el proceso que hace el ingreso de la clase entidad, este valor el procedimiento lo toma del objeto EEoNucleo conformado por el gedee del proceso en ejecución,

seguido de :: y el id del proceso en ejecución (self::$EEoNucleo->gedeeProcesoEjecucion . '::' . self::$EEoNucleo->idProcesoEjecucion()) y es para definir el único proceso que puede eliminar la entidad namespace/clase de este arreglo, este elemento está sujeto a la propiedad $lbPropietarioEntidad de la sección de seguridad de este trait (GECO). Cuando el proceso propietario de la entidad namespace/clase decide eliminar esta entidad, se eliminará junto con todas las instancias contenidas de esta namespace/clase.

'instancias': (array) contiene las instancias que se creen de la clase correspondiente, contiene las instancias y los datos correspondientes a cada instancia, entre estos datos están:

id_entidad: (string) es el nombre o identificador de la entidad (obligatorio), los id_entidad pueden ser ingresados en el momento que se ingresa el namespace/clase de la clase entidad o posteriormente haciendo una petición de ingreso de instancia de una entidad a través del procedimiento * ingresarEntidad * de este trait (GECO).

'registro': (boolean) en caso de 'tipo_entidad' tener valor 'clase' entonces esta propiedad contiene el valor true si ha sido solicitada al menos una vez.

'objeto': (&obj) en caso de 'tipo_entidad' tener valor 'objeto' entonces esta propiedad contiene la referencia al objeto, entidad a gestionar.

'proceso_propietario': (string) es el proceso que hace el ingreso de la entidad, este valor el procedimiento lo toma del objeto EEoNucleo conformado por el GEDEE del proceso en ejecución, seguido de :: y el id del proceso en ejecución (self::$EEoNucleo->gedeeProcesoEjecucion . '::' . self::$EEoNucleo->idProcesoEjecucion()) y es para definir el único proceso que puede eliminar la entidad instancia de este arreglo, este elemento está sujeto a la propiedad $lbPropietarioEntidad de la sección de seguridad de este trait (GECO). Se aplica solo a la entidad específica, no tiene que ser el mismo propietario que el de la entidad namespace/clase base de esta entidad específica.

'datos': es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad en este trait (GECO); estos datos se crean en el momento del ingreso de la entidad sin que permita cambios posteriores.

la estructura del arreglo quedaría así:

```
$laEntidades = array(
                    'nombre namespace' => array(
                                        'nombre clase' => array(
                                                            'path_entidad_clase' => 'path_entidad_clase',
                                                            'tipo_entidad' => 'tipo_entidad',
                                                            'iniciacion' => 'procedimiento'
                                                            'proceso_propietario' => 'proceso_que_ingresa_la_entidad',
                                                            'instancias' => array(
                                                                        'id_entidad1' => array(
                                                                                    'registro' => 'true o false,
```

```
                                                'objeto' => &objeto,
                                                'proceso_propietario' => 'proceso_que_ingresa_la_entidad',
                                                'datos' => array( 'dato1' => 'valor_dato' , 'datoN' =>datoN')
                                                        ) ,
                                        'id_entidadN' => array(
                                                'registro' => 'true o false,
                                                'objeto' => &objeto,
                                                'proceso_propietario' => 'proceso_que_ingresa_la_entidad',
                                                'datos' => array( 'dato1' => 'valor_dato' , 'datoN' =>datoN')
                                                        )
                                                )
                                        )
                                )
                        )
```

Procedimientos:

```
public static function iniciacion( $La_datos_iniciacion )
```

es el procedimiento encargado de iniciar el funcionamiento de este trait. A continuación la descripción de la estructura de datos del arreglo $La_datos_iniciacion:
$La_datos_iniciacion (array) que contiene los datos necesarios para la iniciación de este trait, sus elementos son los siguientes:

```
$La_datos_iniciacion['seguridad']['gestion_seguridad']
$La_datos_iniciacion['seguridad']['propietario_entidad']
$La_datos_iniciacion['seguridad']['ambito_seguridad']
$La_datos_iniciacion['seguridad']['actualizar_datos_seguridad']
$La_datos_iniciacion['seguridad']['datos_seguridad']

$La_datos_iniciacion['path_dir_raiz']
```

A continuación explicamos cada elemento:

$La_datos_iniciacion['seguridad'] (array)(asociativo), define una serie de parámetros que aportan niveles de seguridad a la gestión y administración de entidades.

```
        $La_datos_iniciacion['seguridad']:
                        gestion_seguridad: true
                        propietario_entidad: true
                        ambito_seguridad: permisivo o restrictivo
                        actualizar_datos_seguridad: false
                        datos_seguridad:
                                nombre namespace1:
                                        nombre clase1
                                        nombre clase2
                                        nombre claseN
                                nombre namespaceN:
                                        nombre clase1
```

nombre clase2
nombre claseN

- gestion_seguridad (boolean) es quien define si se toman o no, en cuenta los parámetros de seguridad para la gestión y administración de entidades.

- propietario_entidad (boolean) en true define si al ingresar una entidad en este trait gestor de entidades se registra el proceso que hiso el ingreso de la entidad, esto se hace con el objetivo de que la opción de eliminar la entidad sea controlada y solo pueda eliminarla el proceso que la creó, de lo contrario cualquier proceso puede eliminar la entidad ingresada. Este elemento es el único que aunque este desactivado el control de seguridad (gestion_seguridad: false) funciona según su valor.

- ambito_seguridad (string) es el ámbito de seguridad por el que se desarrollará el chequeo de la lista de entidades, que se admiten o deniegan su ingreso en este trait gestor de entidades. Sus valores posibles son *permisivo* o *restrictivo*, permisivo permite todos y restringe solo los elementos de la lista, restrictivo restringe todos y permite solo los elementos de la lista.

- actualizar_datos_seguridad (boolean) define la posibilidad de que cualquier proceso pueda modificar el elemento datos_seguridad, este elemento con valor true permite que cualquier proceso modifique la lista de procesos permitidos o denegados su gestión por parte de este trait gestora de entidades.

- datos_seguridad (array)(asociativo) es donde se listan las entidades que se permitirán o denegarán su gestión por parte de este trait gestor de entidades, la estructura de este arreglo está dada por las clases y sus namespace, que serán chequeados, a continuación un ejemplo de la estructura

```
datos_seguridad:
        namespace_ entidades1:
                clase_ entidades 1
                clase_ entidades 2
                clase_ entidades N
        namespace_ entidades N:
                clase_ entidades 1
                clase_ entidades 2
                clase_ entidades N
```

namespace_ entidades (string) es el nombre del namespace al que pertenece la entidad. (obligatorio)

clase_ entidades (string) es el nombre de la clase a la que pertenece la entidad. (obligatorio)

$La_datos_iniciacion['path_dir_raiz'] (string) path del directorio entidades para los recursos de entidades en el sistema Sockeletom. Este será el fragmento de path que completará el path del directorio base de los recursos de entidades, del directorio que el sistema propone como contenedor de las entidades que se gestionen por este trait y los demás códigos clientes del Sockeletom. Este parámetro tiene además como path raíz al directorio esquelemod, pertenecientes a la estructura de directorios del sistema Sockeletom. Tomando como ejemplo el valor ¨ entidades ¨, quedará como resultado el path:
 raíz_sistema_Sockeletom/esquelemod/entidades

este procedimiento retorna null si ya estaba inicializada la clase, retorna true si la gestión es exitosa, detiene el proceso php si los parámetros son incompatibles con el procedimiento, emitiendo un error fatal (exit).

public static function pathRaizEntidades()
es el procedimiento encargado de brindar, retornar el path del directorio raíz para las entidades en el sistema Sockeletom. Este será el fragmento de path que completará el path del directorio base de los recursos de entidades, del directorio que el sistema propone como contenedor de los recursos de entidades que se utilicen por el núcleo del sistema y los demás códigos clientes del Sockeletom. Este parámetro tiene además como path raíz al directorio esquelemod, pertenecientes a la estructura de directorios del sistema Sockeletom. Tomando como ejemplo el valor ¨ entidades ¨, quedará como resultado el path:
 raíz_sistema_Sockeletom/esquelemod/entidades

este procedimiento retorna el path gestionado o null en caso de no existir el path a gestionar.

public static function actualizarDatosSeguridadEntidades($La_datos , $Ls_tipo_actualizacion)
es el procedimiento encargado de gestionar la actualización de la propiedad self:: $laDatosSeguridad de este trait, la gestión de actualización está sujeta al valor de la propiedad self::$lbActualizarDatosSeguridad perteneciente a la sección de seguridad de este trait. Los parámetros se explican a continuación:

$La_datos: (array) es el arreglo con los datos a actualizar.

$Ls_tipo_actualizacion: (string) es el tipo de actualización a realizar, sus posibles valores son:

'renovar': es la sustitución del arreglo $La_datos por el arreglo actual en self::$laDatosSeguridad.

'adicionar': es la adición del contenido del arreglo $La_datos en el arreglo actual self:: $laDatosSeguridad.

'eliminar': es la eliminación del contenido del arreglo $La_datos en el arreglo actual self::$laDatosSeguridad.

este procedimiento retorna true si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

public static function permisionEntidad($Ls_namespace , $Ls_clase)
es el procedimiento encargado de gestionar si una clase entidad determinada tiene permiso de ser gestionada por este trait (GECO), el permiso es a nivel de clase y no de entidades específicas de esta, se da permiso o no a la clase no a una entidad correspondiente a la clase; la gestión de chequeo de permisos está sujeto al valor de las propiedades self::$lbGestionSeguridad, self::$lsAmbitoSeguridad y self::$laDatosSeguridad pertenecientes a la sección de seguridad de este trait (GECO); específicamente chequea las clases permitidas o denegadas en self::$laDatosSeguridad según el self::$lsAmbitoSeguridad declarado. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la clase entidad a chequear su permiso de gestión (obligatorio)

$Ls_clase: (string) es el nombre de la clase a la que pertenece la clase entidad a chequear su permiso de gestión (obligatorio)

este procedimiento retorna true si la clase entidad tiene permiso a ser gestionada por este trait (GECO), de lo contrario retorna null.

public static function gestionControlIngresoEntidad($Ls_path_control_entidad , $Ls_referencia_path_control = 'relativo' , $Ls_namespace = null , $Ls_clase = null , $Ls_path_entidad_clase = null , $Ls_referencia_path_entidad = null , $Ls_tipo_entidad = null , $La_instancias = null , $Ls_iniciacion = null)
es el procedimiento encargado de gestionar el ingreso de una entidad al contenedor $laEntidades de este trait, utilizando entre sus parámetros un parámetro que hace referencia al path de un fichero control.

Se llama fichero control a un fichero o script php que se necesita su ejecución como parte de la gestión de ingreso de la entidad a este trait. El control puede utilizarse para chequear, gestionar o poner a punto algún elemento necesario para el ingreso de la entidad a este trait.

El fichero control es ejecutado (require_once) por este procedimiento y puede utilizarse en conjunto con los demás parámetros de este procedimiento o gestionar el mismo la inclusión (require_once) de la clase entidad y retornar el mismo los parámetros que necesita este procedimiento para la gestión del ingreso de la entidad a este trait.

Las diferentes combinaciones que pueden utilizarse devienen diferentes funcionalidades que son:

Funcionalidad 1 donde el control debe ejecutarse y las características de la entidad se pasan como parámetros independientes del control. Si se declaran parámetros para el control y a la vez se declaran los parámetros
$Ls_namespace, $Ls_clase, $Ls_path_entidad_clase, $Ls_referencia_path_entidad, $Ls_tipo_entidad , $La_instancias = null , $Ls_iniciacion este procedimiento incluirá el fichero control y además utilizará los parámetros antes mencionados para la gestión de ingreso de la entidad , no importa que el control retorne también los parámetros antes mencionados porque este procedimiento da prioridad a los valores de sus parámetros y nunca evalúa los parámetros homólogos retornados por el control. Tienen que existir todos los parámetros antes mencionados para que esta opción se ejecute, si solo uno de estos parámetros es vacío entonces se pasa a las siguientes funcionalidades.

antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de la entidad, si no existe el fichero el procedimiento no ingresa la entidad y retorna null. Ingresar una entidad no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de esta entidad . Los parámetros se explican a continuación:

$Ls_path_control_entidad : (string) es el path del fichero control a ejecutar para el ingreso de la entidad. (obligatorio)

$Ls_referencia_path_control: (string) es la forma que se hace referencia a $Ls_path_control_entidad, tiene tres posibles valores, *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el $Ls_path_control_entidad relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en este trait. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el $Ls_path_control_entidad relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el $Ls_path_control_entidad de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad.

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad.

$Ls_path_entidad_clase: (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de este trait, si el fichero no existe no se ingresa la entidad al arreglo.

$Ls_referencia_path_entidad: (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el $Ls_path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en este trait. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el $Ls_path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el $Ls_path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

$Ls_tipo_entidad: (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

$Ls_iniciacion: (string) es la forma como se iniciará cada id_entidad, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de $Ls_iniciacion son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades (GECO) al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades (GECO) le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades (GECO) le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad sea de tipo clase ($Ls_tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento [$Ls_namespace] [$Ls_clase][‘instancias’][id_entidad]['parametros_iniciacion'] de la clase base, por lo que cada entidad puede tener sus valores propios, excepto en caso de que la entidad sea de tipo clase ($Ls_tipo_entidad = 'clase') que con el primer id_ entidad se inicia la clase base. Los elementos [$La_instancias][id_entidad]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

$La_instancias (array) es un arreglo con los identificadores de entidades a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad. (obligatorio)

id_entidad (string) es un nombre o identificador de cada entidad instancia de la clase correspondiente. y a su vez este id_entidad es un arreglo asociativo al que se puede insertar una llave de nombre parametros_iniciacion y otra de nombre datos, posteriormente este arreglo id_entidad se le incorporan otros elementos con llave asociativa para el trabajo de este trait. Pueden existir tantos id_entidad como sea necesario ya que la entidad puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'parametros_iniciacion: (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad (id_entidad), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la clase entidad base de la entidad a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad, es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

‘datos’: (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad en este trait (GECO); estos datos se crean en el momento del ingreso de la entidad sin que permita cambios posteriores. (opcional)

Para el resto de las funcionalidades. Es necesario volver a aclarar que si se declaran parámetros en este procedimiento para el fichero control y al menos uno de los restantes es vacio entonces este procedimiento ejecuta (require_once) el control, desecha los demás parámetros pasados a este procedimiento y espera en forma de retorno(return) desde el fichero control requerido(require_once), un arreglo asociativo con las posibles siguientes estructuras:

Funcionalidad 2 donde el control debe ejecutarse, y este mismo ya incluyó la clase y nos retorna las características de la entidad incluido la o las instancias(id_entidad)(objeto) de la clase, para que se ejecute esta funcionalidad esta estructura tiene que estar completa, si faltase algún elemento no se ejecutará esta funcionalidad, también tiene como condición que el elemento $La_resultado_require['referencia_path_entidad'] no exista, la responsabilidad de incuir la estructura correcta de los id_entidad instanciadas (objeto) o registradas(clase) es responsabilidad del control, también es obligatorio que exista al menos una id_entidad en la estructura de datos que se espera por retorno. La estructura que se espera por retorno es la siguiente:

$La_resultado_require['namespace'] (obligatorio)
$La_resultado_require['clase'] (obligatorio)
$La_resultado_require['path_entidad_clase'] (obligatorio)
$La_resultado_require['tipo_entidad'] (obligatorio)
$La_resultado_require['iniciacion'] (opcional)
$La_resultado_require['instancias'] (obligatorio)
$La_resultado_require['instancias'] [id_entidad'] (obligatorio)
$La_resultado_require['instancias'] [id_entidad'] ['objeto'] (opcional)
$La_resultado_require['instancias'] [id_entidad] ['registro'] (opcional)
$La_resultado_require['instancias'] [id_ entidad] [' datos'] (opcional)

‘namespace’: (string) es el nombre del namespace al que pertenece la entidad. (obligatorio)

‘clase’: (string) es el nombre de la clase a la que pertenece la entidad . (obligatorio)

‘path_entidad_clase’: (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de este trait, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad al arreglo $laEntidades de este trait. (obligatorio)

‘tipo_entidad’: (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

‘iniciacion’: (string) es la forma como se iniciará cada id_entidad , este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades(GECO) al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades(GECO) le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades (GECO) le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento [namespace][clase]['instancias'][id_entidad]['parametros_iniciacion'] de la clase base, por lo que cada entidad puede tener sus valores propios, excepto en caso de que la entidad sea de tipo clase (tipo_entidad = 'clase') que con el primer id_ entidad se inicia la clase base. Los elementos ['instancias'][id_entidad]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

‘instancias’ (array) es un arreglo con los identificadores de entidades a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad. (obligatorio)

id_entidad (string) es un nombre o identificador de cada entidad instancia de la clase correspondiente. y a su vez este id_entidad es un arreglo asociativo al que se puede insertar una llave de nombre objeto, una llave de nombre registro, y otra de nombre datos, las llaves objeto, y registro no deben coexistir ya que la existencia de una determina la no existencia de la otra ,posteriormente este arreglo id_entidad se le incorporan otros elementos con llave asociativa para el trabajo de este trait. Pueden existir tantos id_entidad como sea necesario ya que la entidad puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

‘objeto’: (&obj) es un apuntador al objeto instancia de la clase entidad a contener en el contenedor $laEntidades de este trait. Este elemento es obligatorio si el elemento tipo_entidad tiene como valor el string objeto y estrictamente se considera iniciado.

‘registro’: (boolean true) es para definir cuando una entidad a contener en el contenedor $laEntidades de este trait, y de tipo_entidad clase (procedimientos estáticos) es iniciada . Este elemento es obligatorio si el elemento tipo_entidad tiene como valor el string clase y estrictamente se considera iniciada.

‘datos’: (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad en este trait; estos datos se crean en el momento del ingreso de la entidad sin que permita cambios posteriores. (opcional)

Funcionalidad 3 donde el control debe ejecutarse, y este nos retorna(return) los elementos o parámetros para ingresar una clase, y reserva al menos un id_entidad para su posterior instanciación, no debe tener id_entidades instanciadas(objeto) o registradas(clase), y es obligatorio que exista en los parámetros retornados por el control el elemento $La_resultado_require['referencia_path_entidad']. Entiéndase que esta funcionalidad solo realiza su gestión si no existe ningún id_entidad instanciado este factor es obligatorio. La estructura que se espera por retorno es la siguiente:

$La_resultado_require['namespace'] (obligatorio)
$La_resultado_require['clase'] (obligatorio)
$La_resultado_require['path_entidad_clase'] (obligatorio)
$La_resultado_require['referencia_path_entidad '] (obligatorio)
$La_resultado_require['tipo_entidad'] (obligatorio)
$La_resultado_require['iniciacion'] (opcional)
$La_resultado_require['instancias'] (obligatorio)
$La_resultado_require['instancias'] [id_entidad] (obligatorio)
$La_resultado_require['instancias'] [id_entidad] ['parametros_iniciacion'] (opcional)
$La_resultado_require['instancias'] [id_entidad] [' datos'] (opcional)

‘namespace’: (string) es el nombre del namespace al que pertenece la entidad. (obligatorio)

‘clase’: (string) es el nombre de la clase a la que pertenece la entidad. (obligatorio)

‘path_entidad_clase’: (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de este trait, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad al arreglo $laEntidades de este trait. (obligatorio)

‘referencia_path_entidad’: (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en este mismo trait. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

‘tipo_entidad’: (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

‘iniciacion’: (string) es la forma como se iniciará cada id_entidad, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades (GECO) al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades (GECO) le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades (GECO) le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento [namespace][clase]['instancias'][id_entidad]['parametros_iniciacion'] de la clase base, por lo que cada entidad puede tener sus valores propios, excepto en caso de que la entidad sea de tipo clase (tipo_entidad = 'clase') que con el primer id_ entidad se inicia la clase base. Los elementos ['instancias'][id_entidad]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

‘instancias’ (array) es un arreglo con los identificadores de entidades a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad. (obligatorio)

id_entidad (string) es un nombre o identificador de cada entidad instancia de la clase correspondiente. y a su vez este id_entidad es un arreglo asociativo al que se puede insertar una llave de nombre ‘parametros_iniciacion’ y otra de nombre ‘datos’, posteriormente este arreglo id_entidad se le incorporan otros elementos con llave asociativa para el trabajo de este trait. Pueden existir tantos id_entidad como sea necesario ya que la entidad puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'parametros_iniciacion’: (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad (id_entidad), si existe este parámetro, es arreglo y no es vacio entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad, es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

‘datos’: (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad en este trait (GECO); estos datos se crean en el momento del ingreso de la entidad sin que permita cambios posteriores. (opcional)

La funcionalidad que el control ejecutado devuelva los datos de la clase y sus elementos ['instancias'][id_entidad] no instanciados, es por seguridad ya que existen otras formas para crear controles que retornen estructuras con sus id_entidad ya instanciados(objeto) o registrados(clase); Esta característica está abierta a cambios futuro en dependencia de la opinión mayoritaria de los usuarios del sistema Sockeletom.

antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de la entidad, si no existe el fichero el procedimiento no ingresa la entidad y retorna null. Ingresar una entidad no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de la entidad.

este procedimiento retorna true si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

public static function gestionIngresoEntidad ($Ls_namespace , $Ls_clase , $Ls_path_entidad_clase , $Ls_referencia_path_entidad , $Ls_tipo_entidad , $La_instancias , $Ls_iniciacion = null)
es el procedimiento encargado de gestionar el ingreso de una entidad al contenedor $laEntidades de este trait, antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de la entidad, si no existe el fichero el procedimiento no ingresa la entidad y retorna null. Ingresar una entidad no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de la entidad. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad. (obligatorio)

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad. (obligatorio)

$Ls_path_entidad_clase: (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de este trait, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad al arreglo $laEntidades de este trait. (obligatorio)

$Ls_referencia_path_entidad: (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en este mismo trait. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

$Ls_tipo_entidad: (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

$Ls_iniciacion: (string) es la forma como se iniciará cada id_entidad, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento $La_instancias[id_entidad]['parametros_iniciacion'] de la clase base, por lo que cada entidad puede tener sus valores propios, excepto en caso de que la entidad sea de tipo clase (tipo_entidad = 'clase') que con el primer id_ entidad se inicia la clase base. Los elementos $La_instancias[id_entidad] ['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

$La_instancias (array) es un arreglo con los identificadores de entidades a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad. (obligatorio)

id_entidad (string) es un nombre o identificador de cada entidad instancia de la clase correspondiente. y a su vez este id_entidad es un arreglo asociativo al que se puede insertar una

llave de nombre ‘parametros_iniciacion’ y otra de nombre ‘datos’, posteriormente este arreglo id_entidad se le incorporan otros elementos con llave asociativa para el trabajo de este trait. Pueden existir tantos id_entidad como sea necesario ya que la entidad puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad (id_entidad), si existe este parámetro, es arreglo y no es vacio entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de que la clase entidad base de la entidad a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

‘datos’: (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad en este trait (GECO); estos datos se crean en el momento del ingreso de la entidad sin que permita cambios posteriores. (opcional)

public static function gestionIngresosEntidades($La_propiedades_entidades)
es el procedimiento encargado de gestionar el ingreso de varias entidades al contenedor $laEntidades de este trait, este procedimiento desglosa un arreglo($La_propiedades_entidades) con datos de varias entidades y realiza la gestión de ingreso según sus características como lo hacen individualmente los procedimientos self::gestionControlIngresoEntidad y self::gestionIngresoEntidad, es importante aclarar que cada entidad será procesada estrictamente con los procedimientos self::gestionControlIngresoEntidad o self::gestionIngresoEntidad por lo que los datos de las entidades tienen que cumplir con las condicionantes del procedimiento respectivo para que la gestión sea exitosa ,antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de cada entidad, si no existe el fichero el procedimiento no ingresa la entidad y continua su gestión. Ingresar una entidad no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de la entidad. Los parámetros se explican a continuación:

$La_propiedades_entidades: es un arreglo en el que los elementos llave de este arreglo son los namespaces de las entidades a gestionar su ingreso en este trait, la llave namespace representa a su vez un arreglo con las clases y algunas características de estas clases que finalmente son las entidades declaradas, la llave clase representa a su vez un arreglo con algunas características necesarias para la correcta gestión de las entidades, un ejemplo de esta estructura se expone a continuación:

```
$La_propiedades_ entidades
        namespace_entidad 1:
                clase_entidad 1:
                        path_entidad_clase:
                        referencia_path_entidad:
                        path_control:
                        referencia_path_control:
                        tipo_entidad:
                        iniciación:
                        instancias:
                                id_entidad 1:
                                        parametros_iniciacion
                                        datos:
                                id_entidad 2
                                        parametros_iniciacion
                                id_entidad 3
                                id_entidad 4:
                                        datos:

        namespace_entidad 2:
                clase_entidad 1:
                        path_entidad_clase:
                        referencia_path_entidad:
                        path_control:
                        referencia_path_control:
                        tipo_entidad:
                        instancias:
                                id_entidad 1:
                                        datos:
                                id_entidad 2
                                id_entidad 3
                                id_entidad 4
                clase_entidad 2:
                        path_entidad_clase:
                        referencia_path_entidad:
                        path_control:
                        referencia_path_control:
                        tipo_entidad:
                        iniciación:
                        instancias:
                                id_entidad 1:
                                        datos:
                                id_entidad 2:
```

parametros_iniciacion:
datos:
id_entidad 3:
datos:
id_entidad 4:
datos:

los elementos de este arreglo se explican a continuación:

namespace: (string) es el nombre del namespace al que pertenece la entidad. (obligatorio en dependencia del procedimiento gestor)

clase: (string) es el nombre de la clase a la que pertenece la entidad. (obligatorio en dependencia del procedimiento gestor)

‘path_entidad_clase’: (string) contiene el path del fichero que alberga el código de la clase a la que pertenece la entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de este trait(GECO), se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad al arreglo $laEntidades de este trait(GECO). (obligatorio en dependencia del procedimiento gestor)

‘referencia_path_entidad’: (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en este trait. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio en dependencia del procedimiento gestor)

‘path_control’: es el path de un fichero que controlará condiciones y/o el inicio o inclusión de una entidad, el procedimiento self::gestionControlIngresoEntidad o self::gestionIngresoEntidad según corresponda, espera un retorno de este control con un arreglo asociativo donde se encuentren los datos namespace_entidad, clase_entidad, id_entidad, path_entidad_clase, referencia_path_entidad , tipo_entidad , instancias con sus datos, o combinaciones de estos. (obligatorio en dependencia del procedimiento gestor)

‘referencia_path_control’: (string) es la forma que se hace referencia a path_control, tiene tres posibles valores, *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_control relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en este trait. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el

path_control relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_control de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio en dependencia del procedimiento gestor)

‘tipo_entidad’: (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio en dependencia del procedimiento gestor)

‘iniciacion’: (string) es la forma como se iniciará cada id_entidad, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento ['instancias'][id_entidad]['parametros_iniciacion'] de la clase base, por lo que cada entidad puede tener sus valores propios, excepto en caso de que la entidad sea de tipo clase (tipo_entidad = 'clase') que con el primer id_ entidad se inicia la clase base. Los elementos ['instancias'][id_entidad] ['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional en dependencia del procedimiento gestor)

‘instancias’ (array) es un arreglo con los identificadores de entidades a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad. (obligatorio en dependencia del procedimiento gestor)

id_entidad (string) es un nombre o identificador de cada entidad instancia de la clase correspondiente. y a su vez este id_entidad es un arreglo asociativo al que se puede insertar una llave de nombre ‘parametros_iniciacion’ y otra de nombre ‘datos’, posteriormente este arreglo id_entidad se le incorporan otros elementos con llave asociativa para el trabajo de este trait. Pueden existir tantos id_entidad como sea necesario ya que la entidad puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1 en dependencia del procedimiento gestor) (opcional ++)

'parametros_iniciacion’: (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad (id_entidad), si existe

este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la clase entidad base de la entidad a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional en dependencia del procedimiento gestor)

‘datos’: (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad en este trait; estos datos se crean en el momento del ingreso de la entidad sin que permita cambios posteriores. (opcional en dependencia del procedimiento gestor)

se aconseja por parte del grupo de desarrollo del núcleo Sockeletom que los namespaces se declaren de forma absoluta, comenzando con '\ '

este procedimiento retorna true si la gestión de al menos una entidad fue llevada a cabo con éxito, de lo contrario retorna null.

public static function existenciaEntidad ($Ls_namespace , $Ls_clase , $Ls_id_entidad = null)
es el procedimiento encargado de gestionar si existe una entidad determinada en el contenedor $laEntidades de este trait, si existe una entidad a disposición de pedido y uso, esta entidad no tiene obligatoriamente que estar iniciada, es decir no tiene obligatoriamente que existir la instancia ya creada.

Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad. (obligatorio).

$Ls_id_entidad: (string) es el nombre o id de la entidad hija de una clase ya ingresada en $this->laEntidades, este parámetro puede ser vacío, de ser vacío se entiende que se busca la existencia de una clase(\namespace\clase) ya ingresada en $this->laEntidades, clase base para entidades instancias(objeto) o clase (clase de procedimientos estáticos).

Si $Ls_id_entidad tiene valor y existe como entidad hija de la \namespace\clase base en la estructura de datos de $this->laEntidades entonces este procedimiento retorna el tipo_entidad (‘objeto’ o ‘clase’) de la clase base. Si no existe como entidad hija el procedimiento retorna null.

Si $Ls_id_entidad no tiene valor, o tiene valor vacío, y existe la entidad \namespace\clase base en la estructura de datos de $this->laEntidades entonces este procedimiento retorna el tipo_entidad (‘objeto’ o ‘clase’) de la clase base. Si no existe como la entidad \namespace\clase base el procedimiento retorna null.

public static function ingresarEntidad ($Ls_namespace , $Ls_clase , $Ls_id_entidad , $La_parametros_iniciacion = null)
es el procedimiento encargado de gestionar el ingreso de entidades hija de la \namespace\clase base en la estructura de datos de $this->laEntidades. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad, es obligatorio que exista ya en $this->laEntidades. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad, es obligatorio que exista ya en $this->laEntidades. (obligatorio).

$Ls_id_entidad: (string) es el nombre o id de una entidad a ingresar en la estructura’ instancias’ de la clase base \namespace\clase, en la estructura de datos de $this->laEntidades (obligatorio).

$La_parametros_iniciacion: (array) cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad ($Ls_id_entidad), si existe este parámetro, es arreglo y no es vacío entonces sus elementos se pasan como parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad clase base de la entidad a gestionar sea de tipo clase (tipo_entidad = 'clase') no tiene efecto este parámetro ya que se gestionó en el primer elemento $Ls_id_entidad que se ingreso en la clase base. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

public static function entidad($Ls_namespace , $Ls_clase , $Ls_id_entidad , $Ls_tipo_referencia = 'referencia')
es el procedimiento encargado de gestionar la entidad hija de la \namespace\clase base en la estructura de datos de $this->laEntidades identificada por $Ls_id_entidad, es decir si la entidad es de tipo ‘objeto’ retornará la referencia a un objeto instancia de la entidad, si la entidad es de tipo clase retornará un string compuesto por el namespace y nombre de la clase base. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad. (obligatorio).

$Ls_id_entidad: (string) es el nombre o id de la entidad hija de una clase ya ingresada en $this->laEntidades, el string en $Ls_id_entidad tiene obligatoriamente que existir como elemento en la estructura instancias de la clase base en $this->laEntidades. (obligatorio).

$Ls_tipo_referencia: (string) es el tipo de referencia a objeto que devuelve el procedimiento en caso de que el tipo de entidad sea ‘objeto‘, este parámetro tiene dos valores posibles ‘referencia‘ y ‘clon‘, referencia devuelve un puntero a la entidad guardada en el arreglo $this->laEntidades, y clon devuelve un clon de este objeto. Si el tipo de entidad es ‘clase‘ no se considera este parámetro. Por defecto este parámetro tiene como valor ‘referencia‘.

Este procedimiento devuelve (retorna) una referencia o clon a objeto si el tipo_entidad de la clase base es (‘objeto’); devuelve (retorna) un string de la forma \namespace\clase si el tipo_entidad de la clase base es (‘clase’); de lo contrario retorna null.

La instanciación de los objetos ocurre en el momento que es pedido por primera vez una instancia a través de este procedimiento.

En el caso de la instanciación o iniciación de entidades de tipo ‘objeto’, e iniciación de entidades de tipo ‘clase’, estas entidades pueden emitir una excepción desde su constructor o procedimiento de iniciación si existe algún problema de instanciación o iniciación, ya que este procedimiento tiene un bloque try y catch para manejar las excepciones, el bloque catch espera un objeto de la clase \Exception y emite el mensaje de este, el código es de la siguiente forma:

```
catch( \Exception $e)
  {
      echo '<p>Excepción capturada: '.$e->getMessage(). "<p>" ;
      return null;
   }
```

Por lo que el lanzamiento de la excepción sería:

```
throw new \Exception('mensaje.');
```

Esta funcionalidad es muy útil para interrumpir la instanciación de una clase si existe algún error en esta instanciación, y así no se tienen objetos con problemas o sin uso por desperfectos, en este contenedor de instancias.

Lo anterior se aplica a entidades de tipo ‘objeto’ que lancen su excepción desde el constructor, o entidades de tipo ‘clase’ que lancen su excepción desde el procedimiento de iniciación; Quedan excluidos las entidades de tipo ‘objeto’ que lancen su excepción desde un procedimiento de iniciación (no __construct) ya que estos ya fueron instanciados existen como instancias en el contenedor, estos casos solo tienen la opción de emitir un mensaje.

```
public static function datosEntidad ( $Ls_namespace , $Ls_clase , $Ls_id_entidad)
```

es el procedimiento encargado de gestionar y retornar los datos, que de existir, tiene una entidad que haya ingresado en este trait, estos datos son los correspondientes al elemento ‘datos’ que puede traer consigo una entidad en el momento de su ingreso al contenedor $laEntidades de este trait. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad. (obligatorio).

$Ls_id_entidad: (string) es el nombre o id de la entidad hija de una clase ya ingresada en $this->laEntidades, el string en $Ls_id_entidad tiene obligatoriamente que existir como elemento en la estructura instancias de la clase base en $this->laEntidades . (obligatorio).

este procedimiento retorna el arreglo ‘datos’ perteneciente a la entidad propietaria si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

public static function idEntidades ($Ls_namespace , $Ls_clase)
es el procedimiento encargado de gestionar y retornar los nombres o identificadores de entidades que hayan ingresado en el namespace y clase correspondientes a los parámetros de este procedimiento, estos datos son los correspondientes al elemento ‘instancias’ que debe tener consigo cada clase base en el contenedor $laEntidades de este trait. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad. (obligatorio).

si la gestión fue llevada a cabo con éxito este procedimiento retorna las llaves del array ‘instancias’ contenidos en el namespace y clase, clase base en el contenedor $laEntidades de este trait(GECO), de lo contrario retorna null.

public static function eliminarEntidad ($Ls_namespace , $Ls_clase , $Ls_id_entidad = null)
es el procedimiento encargado de eliminar una entidad clase base, o una entidad de la estructura ‘instancias’ de una clase base, existentes ambas en el contenedor $laEntidades de este trait. Este procedimiento está sujeto al valor de la propiedad self::$lbPropietarioEntidad perteneciente a la sección de seguridad de este trait(GECO), si esta propiedad tiene valor true solo puede eliminar la entidad el proceso que figura como su propietario, de lo contrario puede eliminarse desde cualquier proceso, el proceso núcleo siempre tiene derechos de eliminar cualquier entidad.

Existen dos posibilidades de uso de este procedimiento:

La primera es eliminar una entidad clase base, es decir una namespace\clase registrada como clase base de entidades, para ello se pasan los valores correspondientes al namespace y clase de la entidad a eliminar, dejando vacío el parámetro $Ls_id_entidad, recordar que esta operación está sujeto a la sección de seguridad de este trait, donde se define quién puede eliminar una entidad determinada.

La segunda es eliminar una entidad de la estructura 'instancias' de una clase base, para ello debe existir la entidad en la estructura $laEntidades[$Ls_namespace][$Ls_clase]['instancias'] de este trait, el parámetro $Ls_id_entidad no puede ser vacío, recordar que esta operación está sujeto a la sección de seguridad de este trait, donde se define quien puede eliminar una entidad determinada.

Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad. (obligatorio).

$Ls_id_entidad: (string) es el nombre o id de la entidad hija de una clase ya ingresada en $this->laEntidades, el string en $Ls_id_entidad tiene obligatoriamente que existir como elemento en la estructura instancias de la clase base en $this->laEntidades . (obligatorio).

este procedimiento retorna true si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

class_einterfaz_datos.php

Este fichero contiene la clase InterfazDatos, que se encarga de gestionar datos y hacer cache de estos, es utilizado por el núcleo Sockeletom para la gestión de sus configuraciones y otros datos. Su funcionamiento se basa en la utilización de:

- un fichero que llamamos interfaz (fichero_interfaz), que es ejecutado con incluye, y que hace el trabajo de interpretar o parsear los datos de un origen de datos determinado, este origen de datos puede ser un fichero u otro tipo de contenedor de datos, del que el fichero interfaz necesite un path o que simplemente el mismo fichero interfaz contiene todos los recursos para obtener los datos.
- Un fichero que llamamos datos (fichero_datos), que hace el trabajo de contener datos en una estructura determinada y que será interpretado o parseado por un fichero_interfaz.
-

El fichero_interfaz es obligatorio que exista. También es obligatorio que retorne con return la estructura de datos gestionada para ser capturada por la instancia de esta clase y continuar su procesamiento.

El fichero_datos es opcional, depende del funcionamiento del fichero_interfaz, es decir de que este último lo requiera, de ser así, el fichero_interfaz en el momento de ser incluido ya contará con la existencia de la variable $Ls_pathfinal_datos disponible en el ámbito de inclusión, esta variable contendrá el path del fichero_datos en caso de haber sido gestionado con éxito por la instancia de esta clase.

Existen varias opciones de creación de cache de los datos gestionados por el fichero_interfaz, opciones muy fáciles de personalizar incluida la opción de no realizar cache de los datos gestionados. Estas opciones son en cuanto al lugar donde crear el fichero cache, la estructura y extensión del fichero serán de tipo php, y el nombre del fichero estará compuesto por un string -identificador- que escogerá el usuario de la instancia de esta clase + el string - nombre del fichero referencia de los datos- que puede ser el fichero_interfaz o el fichero_datos según corresponda + el string '_cache' + el string '.php', quedando de la siguiente forma
identificador+_+nombre del fichero referencia de los datos+_cache.php

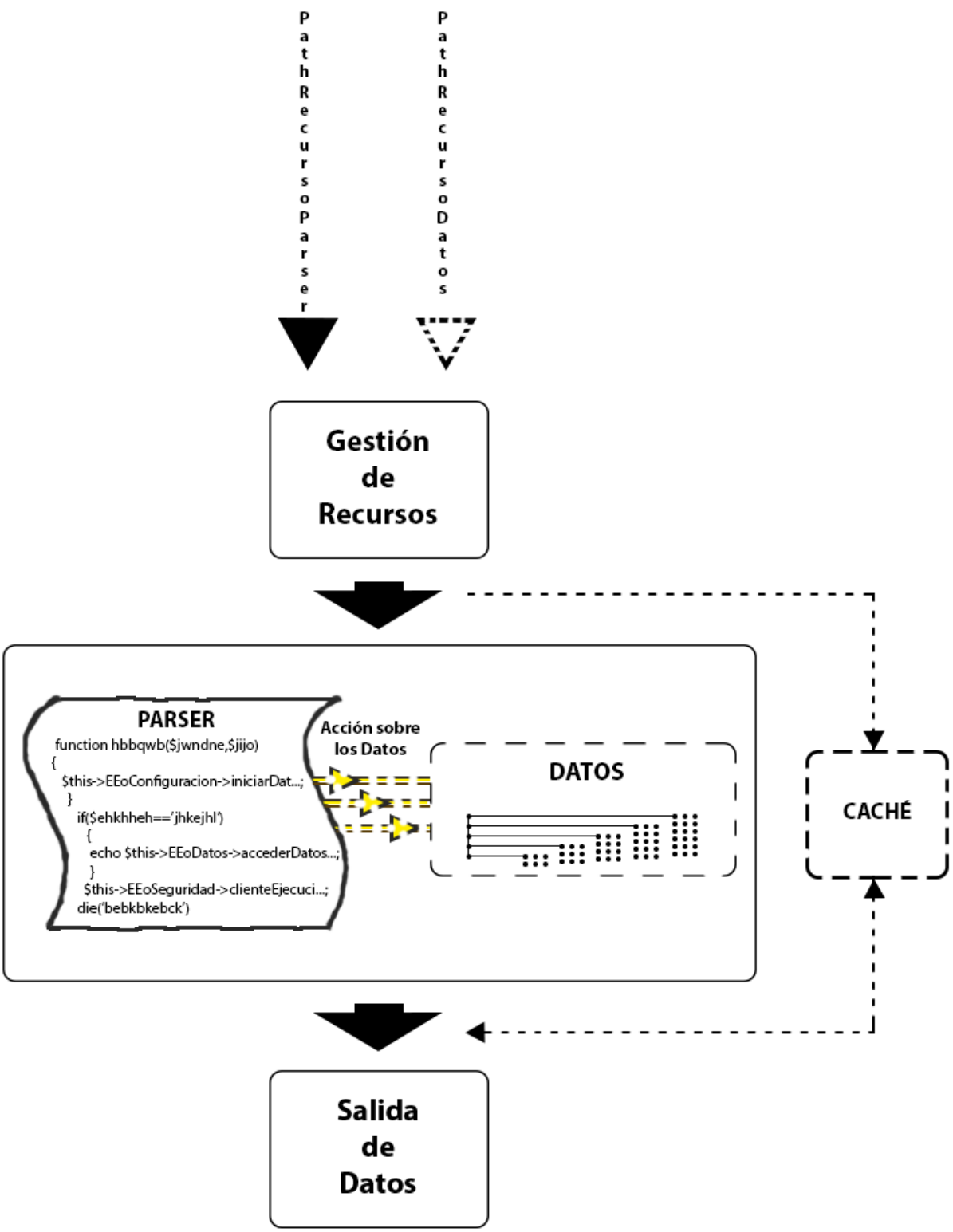
La dinámica de la gestión quedaría de forma siguiente:

Instancia_interfaz_datos -> fichero-cache_datos

Instancia_interfaz_datos <-> fichero-interfaz

Instancia_interfaz_datos <-> fichero-interfaz -> fichero_datos

la siguiente imagen describe lo anterior:


PathRecursoParser
PathRecursoDatos
Gestión de Recursos
PARSER
function hbbqwb($jwndne,$jijo)
{
$this->EEoConfiguracion->iniciarDat...;
}
if($ehkhheh=='jhkejhl')
{
echo $this->EEoDatos->accederDatos...;
}
$this->EEoSeguridad->clienteEjecuci...;
die('bebkbkebck')
Acción sobre los Datos
DATOS
CACHÉ
Salida de Datos

en la imagen el flujo descrito con líneas discontinuas es opcional.

En el caso de los datos, no es que sean opcionales, sino lo que es opcional es la acción de darle al parser el recurso datos para su acceso, ya que el parser puede estar codificado de forma predeterminada que el mismo contenga el path o vía de acceso al recurso datos.

A continuación explicamos la estructura y contenido de esta clase:

Namespace: Emod\Nucleo\Herramientas
Nombre de la clase: InterfazDatos
Herencia: extends \Emod\Nucleo\Herramientas\Multiton
use \Emod\Nucleo\DependenciasEntidadesEmod

Propiedades:

protected $EEoNucleo = null ;
puntero al objeto EEoNucleo ;

private $lsPathFichInterfaz = " ;
esta propiedad contiene el path del fichero interfaz por defecto del objeto instancia de esta clase. Se recomienda que sea absoluto y no relativo en próximas versiones esta propiedad será un arreglo y se guardaran un grupo de interfaces por defecto que vendrán con el Sockeletom.

private $lsPathDirCache = 'adyacente' ;
esta propiedad contiene la forma de decidir el lugar donde se gestionaran (crear, actualizar o consultar) los ficheros caché de datos, sus valores posibles son:

1-'statu_procesos' es solo para los procesos, y creara la cache de datos en el directorio raiz_dir_proceso_ejecucion/statu/cache_datos/, es decir en el directorio statu/cahe_datos/ que se encuentra en el directorio raíz del proceso(ejecutándose) que solicita el procedimiento correspondiente a la gestión de datos. Si el directorio statu/cahe_datos/ no existe el sistema intenta crearlo.

2-'adyacente' se ejecutara primero adyacente_datos y si no es posible se ejecutara adyacente_interfaz, ambas opciones se explican a continuación

3-'adyacente_datos' es para cualquier gestión de datos que necesite cualquier código cliente del sistema Sockeletom, y creará la caché en el mismo directorio que se encuentra el fichero de datos si este existe, en caso de no existir el fichero datos y haberse seleccionado explícitamente el valor adyacente_dato para $lsPathDirCache el sistema retorna un error.

4-'adyacente_interfaz' es para cualquier gestión de datos que necesite cualquier código cliente del sistema Sockeletom, y creará la caché en el mismo directorio que se encuentra el fichero interfaz si este existe, en caso de no existir el fichero interfaz el sistema retorna un error.

5-'path personalizado' es para cualquier gestión de datos que necesite cualquier código cliente del sistema Sockeletom, y creará la caché en el directorio raíz que tendrá como path la ubicación del directorio que se haga explicito en el 'path personalizado'.

6-' valor vació ' que hace referencia a la no implementación (crear, actualizar o consultar) de gestión de caché, solo se gestiona la interfaz y los datos y se devuelve el resultado.

private $lsIniciacionInterfazDatos = null ;
esta propiedad es para declarar la iniciación del objeto instancia de esta clase, con ello no se permite posteriormente la modificación de algunas propiedades del objeto.

Procedimientos:

public function iniciacionImplementacionInterfazDatos($Ls_pathfich_interfaz = " , $Ls_pathdir_cache = ")
este procedimiento se encarga de iniciar el objeto instancia de esta clase con los parámetros que le corresponden para ser utilizados por defecto en procedimientos de esta clase. Los parámetros se explican a continuación:

$Ls_pathfich_interfaz(string) contendrá la dirección de un fichero interfaz para tomarlo como gestor, parser o analizador de datos, por defecto de la instancia de esta clase, este path se recomienda sea absoluto.

$Ls_pathdir_cache(string) este parámetro contiene la forma de decidir el lugar donde se gestionaran (crear, actualizar o consultar) los ficheros caché de datos, sus valores posibles son:

1-'statu_procesos' es solo para los procesos, y creara la cache de datos en el directorio raiz_dir_proceso_ejecucion/statu/cache_datos/, es decir en el directorio statu/cahe_datos/ que se encuentra en el directorio raíz del proceso(ejecutándose) que solicita el procedimiento correspondiente a la gestión de datos. Si el directorio statu/cahe_datos/ no existe el sistema intenta crearlo.

2-'adyacente' se ejecutara primero adyacente_datos y si no es posible se ejecutara adyacente_interfaz, ambas opciones se explican a continuación

3-'adyacente_datos' es para cualquier gestión de datos que necesite cualquier código cliente del sistema Sockeletom, y creará la caché en el mismo directorio que se encuentra el fichero de datos si este existe, en caso de no existir el fichero datos y haberse seleccionado explícitamente el valor adyacente_dato para $lsPathDirCache el sistema retorna un error.

4-'adyacente_interfaz' es para cualquier gestión de datos que necesite cualquier código cliente del sistema Sockeletom, y creará la caché en el mismo directorio que se encuentra el fichero interfaz si este existe, en caso de no existir el fichero interfaz el sistema retorna un error.

5-'path personalizado' es para cualquier gestión de datos que necesite cualquier código cliente del sistema Sockeletom, y creará la caché en el directorio raíz que tendrá como path la ubicación del directorio que se haga explícito en el 'path personalizado'.

6-' valor vació ' que hace referencia a la no implementación (crear, actualizar o consultar) de gestión de caché, solo se gestiona la interfaz y los datos y se devuelve el resultado.

Este procedimiento retorna true si la gestión fue satisfactoria, si al menos un parámetro fue iniciado; retorna null si la gestión fue insatisfactoria.

private function chequeoVersionCache ($Ls_pathfichreferen_datos , $Ls_pathfich_cache)
Este procedimiento se encarga de chequear, en correspondencia con los parámetros, si existe el fichero caché de la interfaz y si este fue modificado en fecha y hora más reciente que el fichero de referencia de los datos que puede ser el fichero interfaz o un fichero de datos. . Los parámetros se explican a continuación:

$Ls_pathfichreferen_datos (string) es el path del fichero que contiene o referencia los datos.

$Ls_pathfich_cache (string) es el path del fichero caché de datos que corresponde al fichero que contiene o referencia los datos.

Este procedimiento retorna true si la versión del fichero que contiene o referencia los datos es más actual que la versión del fichero caché; de lo contrario retorna null.

private function creacionCache($Ls_pathfich_cache , $datos_contenido)
Este procedimiento se encarga de crear, en correspondencia con los parámetros, el fichero caché de datos de los datos correspondientes. Este fichero a crear contendrá una estructura de datos legibles para PHP, partiendo de los datos que se le transfieren a la función a través del parámetro $datos_contenido. Los parámetros se explican a continuación:

$Ls_pathfich_cache (string) es el path donde se creará el fichero caché de datos que corresponde.

$datos_contenido es la estructura de datos a crear caché.

Este procedimiento retorna true si crea el fichero caché, de lo contrario retorna null.

final public function gestionEjecucionInterfazSalida ($Ls_identificativo , $La_fich_interfaz , $La_fich_datos = '' , $Ls_pathdir_cache = 'hereda')
Este procedimiento se encarga de gestionar y ejecutar(include) el fichero interfaz de datos, el fichero interfaz de datos puede tener como objetivo leer o modificar en fuentes de datos. Cuando lee en fuentes de datos, si existen los datos de la interfaz en caché y estos están actualizados los obtiene del fichero caché, si lo anterior no es posible, gestiona y ejecuta el fichero interfaz y el fichero fuente de datos que es analizado por la interfaz en caso de que la interfaz obtenga los datos a través del análisis de un fichero determinado, posteriormente intenta crear un fichero caché con los datos obtenidos(si está habilitada la opción de caché), y emite los datos con return. Cuando modifica en fuentes de datos este procedimiento también devolverá lo que el fichero interfaz retorne de su gestión. Los parámetros se explican a continuación:

$Ls_identificativo(string) es un identificativo que acompañará al nombre de la interfaz o el fichero contenedor de datos según corresponda, y que conformarán el nombre del fichero caché, quedando como nombre del fichero caché
$Ls_identificativo.'_'.'nombrefichreferenciadatos'.'_'.'cache'.'php' . donde nombrefichreferenciadatos es el nombre del fichero que sirve de referencia a los datos a gestionar, estos pueden ser el fichero interfaz o el fichero de datos. Es obligatorio que este parámetro tenga un valor diferente de vacío, de no ser así el procedimiento interrumpe su ejecución y retorna null.

$La_fich_interfaz (array) contendrá los siguientes elementos:

- $La_fich_interfaz['Ls_pathfich_interfaz'] (string) es el path del fichero interfaz, En el caso que se quiera utilizar el path que por defecto tiene asignado el objeto instancia de esta clase, entonces se puede poner el valor 'hereda' o el path explícitamente, de ser así no se tienen en cuenta los demás elementos de este arreglo $La_fich_interfaz. Este elemento debe tener valor diferente de vacío, de lo contrario el procedimiento retorna null

- $La_fich_interfaz['Ls_path_base'] (string) (opcional) es un fragmento de path que se inserta al principio del path $La_fich_interfaz['Ls_pathfich_interfaz'] quedando de la siguiente forma

$La_fich_interfaz['Ls_path_base'].'/'. $La_fich_interfaz['Ls_pathfich_interfaz']

- $La_fich_interfaz['Ls_path_operativo'] (string) (opcional) es un fragmento de path que se inserta al principio del path $La_fich_interfaz['Ls_pathfich_interfaz'] y al final del path $La_fich_interfaz['Ls_path_base'] quedando de la siguiente forma

$La_fich_interfaz['Ls_path_base'] .'/'. $La_fich_interfaz['Ls_path_operativo'] .'/'. $La_fich_interfaz['Ls_pathfich_interfaz']

este elemento, fragmento de path, se recorrerá desde la rama hasta la raíz en busca del fichero interfaz declarado en $La_fich_interfaz['Ls_pathfich_interfaz'], es decir, se puede poner el fichero interfaz en cualquiera de los directorios de este elemento y el sistema lo encontrará y formará el

path real del fichero interfaz, para su tratamiento por parte del procedimiento actual. Esta característica dota al procedimiento de la posibilidad de tener un fichero interfaz para varios subdirectorios dentro de los que se puede considerar la posibilidad de tener o no otro fichero interfaz.

Los elementos anteriores son para uso interno de este procedimiento, pero en caso de querer pasar datos al fichero interfaz, se puede hacer a través de este arreglo, ya que el fichero interfaz tiene alcance a esta variable, un caso muy común es cuando se utiliza este procedimiento para escribir o modificar fuentes de datos, los datos para modificar se pasan en un elemento de este arreglo $La_fich_interfaz o del arreglo $La_fich_datos.

$La_fich_datos (array) (opcional) contendrá los siguientes elementos:

- $La_fich_datos['Ls_pathfich_datos'] (string) (opcional) es el path del fichero contenedor de datos que será procesado, parseado, interpretado por el fichero interfaz.
- $La_fich_datos['Ls_path_base'] (string) (opcional) es un fragmento de path que se inserta al principio del path $La_fich_datos['Ls_pathfich_datos'] quedando de la siguiente forma

$La_fich_datos['Ls_path_base'].'/'. $La_fich_datos['Ls_pathfich_datos']

- $La_fich_datos['Ls_path_operativo'] (string) (opcional) es un fragmento de path que se inserta al principio del path $La_fich_datos['Ls_pathfich_datos'] y al final del path $La_fich_datos['Ls_path_base'] quedando de la siguiente forma

$La_fich_datos['Ls_path_base'] .'/'. $La_fich_datos['Ls_path_operativo'] .'/'. $La_fich_datos['Ls_pathfich_datos']

este elemento, fragmento de path, se recorrerá desde la rama hasta la raíz en busca del fichero datos declarado en $La_fich_datos['Ls_pathfich_datos'], es decir, se puede poner el fichero datos en cualquiera de los directorios de este elemento y el sistema lo encontrará y formará el path real del fichero datos, para su tratamiento por parte del procedimiento actual. Esta característica dota al procedimiento de la posibilidad de tener un fichero datos para varios subdirectorios dentro de los que se puede considerar la posibilidad de tener o no otro fichero datos.

Los elementos anteriores son para uso interno de este procedimiento, pero en caso de querer pasar datos al fichero datos, se puede hacer a través de este arreglo, ya que el fichero interfaz tiene alcance a esta variable, un caso muy común es cuando se utiliza este procedimiento para escribir o modificar fuentes de datos, los datos para modificar se pasan en un elemento de este arreglo $La_fich_datos o del arreglo $La_fich_interfaz.

$Ls_pathdir_cache (string) este parámetro contiene la forma de decidir el lugar donde se gestionaran (crear, actualizar o consultar) los ficheros caché de datos, en caso de que la gestión de

este procedimiento sea de modificación de datos y no de lectura, entonces este parámetro debe tener valor vació (null, ''), sus valores posibles son:

1-'statu_procesos' es solo para los procesos, y creara la cache de datos en el directorio raiz_dir_proceso_ejecucion/statu/cache_datos/, es decir en el directorio statu/cahe_datos/ que se encuentra en el directorio raíz del proceso(ejecutándose) que solicita el procedimiento correspondiente a la gestión de datos. Si el directorio statu/cahe_datos/ no existe el sistema intenta crearlo.

2-'adyacente' se ejecutara primero adyacente_datos y si no es posible se ejecutara adyacente_interfaz, ambas opciones se explican a continuación

3-'adyacente_datos' es para cualquier gestión de datos que necesite cualquier código cliente del sistema Sockeletom, y creará la caché en el mismo directorio que se encuentra el fichero de datos si este existe, en caso de no existir el fichero datos y haberse seleccionado explícitamente el valor adyacente_dato para $lsPathDirCache el sistema retorna un error.

4-'adyacente_interfaz' es para cualquier gestión de datos que necesite cualquier código cliente del sistema Sockeletom, y creará la caché en el mismo directorio que se encuentra el fichero interfaz si este existe, en caso de no existir el fichero interfaz el sistema retorna un error.

5-'path personalizado' es para cualquier gestión de datos que necesite cualquier código cliente del sistema Sockeletom, y creará la caché en el directorio raíz que tendrá como path la ubicación del directorio que se haga explícito en el 'path personalizado'.

6-' valor vació ' que hace referencia a la no implementación (crear, actualizar o consultar) de gestión de caché, solo se gestiona la interfaz y el fichero de datos (en caso de existir este), y se devuelve el resultado de la ejecución (include) del fichero interfaz.

Este procedimiento retorna el valor retornado por el fichero interfaz en su gestión, en caso de que este valor sea vacío, o no haberse ejecutado satisfactoriamente este procedimiento, el valor a retornar por este procedimiento es null.

class_herramienta_earreglo.php

Este fichero contiene la clase EArreglo, que se encarga de ofrecer un grupo de procedimientos dedicados al manejo de arreglos según la funcionalidad del procedimiento, todos los procedimientos son static, a continuación explicamos la estructura y contenido de esta clase:

Namespace: Emod\Nucleo\Herramientas
Nombre de la clase: EArreglo
Herencia:

Propiedades:

Procedimientos:

public static function arregloModificarRecursivo($La_elementos , &$La_base , &$comport = 0 , &$La_estruct_result = NULL , $Ls_estruct_elem = NULL , $Li_nivel = 0)
este procedimiento se encarga de modificar valores de un arreglo ($La_base), basándose en la estructura de otro arreglo como referencia ($La_elementos) para la modificación, esta función resulta solo con php4 o superior y trabaja con arreglos asociativos. Los parámetros se explican a continuación:

$La_elementos (array) es un arreglo asociativo con los elementos que se quieren modificar en el arreglo $La_base (base), esta matriz contendrá la estructura exacta en la que se encuentra el elemento que se quiere modificar, es como un mapa, una imagen de la matriz y su rama o elemento que se quiere modificar, llamémosle de ahora en adelante "estructura imagen", en caso de que toda una rama del arreglo se quiera eliminar basta con poner 'elemento'=array() y se eliminara ese elemento junto a la rama que el constituye, el primer elemento del arreglo $La_elementos será el primer elemento a buscar en el arreglo $La_base , y así sucesivamente.

$La_base (array) es un arreglo asociativo al cual se le modificaran elementos que son representados en la estructura del arreglo $La_elementos.

$comport define condicionantes en el comportamiento de este procedimiento en su gestión con los elementos de los arreglos $La_elementos y $La_base, su valor puede ser un string representando un binario ('011000001') o un decimal, en caso de ser un decimal se convierte a binario, donde si se escoge una condición y esta no se cumple, el elemento base no es modificado. Las condicionantes quedaran representadas de la siguiente manera:

1er bit (bit menos significativo, extremo derecho) en cero (desactivado), no se tiene en cuenta, en 1 (activado) corresponde a modificar los elementos si sus valores son iguales (==).

2do bit en cero (desactivado), no se tiene en cuenta, en 1 (activado) corresponde a modificar los elementos si sus valores son diferentes (!= , <>).

3er bit en cero (desactivado) no se tiene en cuenta, en 1 (activado) corresponde a modificar los elementos si sus valores son no idénticos (!==).

4to bit en cero (desactivado) no se tiene en cuenta, en 1 (activado) corresponde a modificar los elementos si el valor de la imagen es menor que el de la base (<).

5to bit en cero (desactivado) no se tiene en cuenta, en 1 (activado) corresponde a modificar los elementos si el valor de la imagen es mayor que el de la base (>).

6to bit en cero (desactivado) no se tiene en cuenta, en 1 (activado) corresponde a modificar los elementos si el valor de la imagen es menor o igual que el de la base (<=).

7mo bit en cero (desactivado) no se tiene en cuenta, en 1 (activado) corresponde a modificar los elementos si el valor de la imagen es mayor o igual que el de la base (>=).

8vo bit en cero (desactivado) no se tiene en cuenta, en 1 (activado) corresponde a modificar los elementos sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación, pero si existe una estructura o elemento en la imagen que no existe en la base no es modificada la base, es decir no se transfiere el elemento de la imagen a la base.

9no bit en cero (desactivado) no se tiene en cuenta, en 1 (activado) corresponde a registrar en el elemento 'fallos' del arreglo que retorna este procedimiento, los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores

Los valores siguientes son excepciones validas de aclarar

binario: 000000000 (dec 0) modifica los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación, si existe una estructura o elemento en la imagen que no existe en la base es modificada la base con el elemento en cuestión, es decir se transfiere el elemento de la imagen a la base.

binario: 100000000 (dec 256) modifica los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación pero manteniendo el criterio de registrar en los resultados de esta función, en el elemento 'fallos' de esta función, los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores, además mantiene el comportamiento del 8vo bit según su valor.

binario: 110000000 (dec 384) modifica los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir modifica todo sin criterios de comparación pero manteniendo el criterio de registrar en los resultados de esta función, en el elemento 'fallos' de esta función, los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores, además mantiene el comportamiento del 8vo bit según su valor.

binario: 110000000+1 (dec mayor que 384)retorna un valor null.

Si la lógica de los bit del 1ro al 7mo activados equivalen a modificar todo el arreglo o existe redundancia, la función retorna un valor null ej.: 000000101 o 001010000, el 8vo y 9no bit siempre pueden estar activados, no pertenece a la lógica comparativa del lenguaje

$La_estruct_result (array) es para el desarrollo interno de la función y es un arreglo asociativo con los resultados de las operaciones internas de la función, de introducirse un valor en este parámetro no se predice el comportamiento de la función.

$Ls_estruct_elem (string) es para el desarrollo interno de la función ya que esta se llama de forma recursiva, no necesita que le pasen valores y es una cadena que va concatenando a través de los ciclos la estructura a modificar de la matriz, de introducirse un valor en este parámetro no se predice el comportamiento de la función.

$Li_nivel (int) es para el desarrollo interno de la función, es un entero que cuenta los niveles o dimensiones por los que pasa la función, de introducirse un valor en este parámetro no se predice el comportamiento de la función.

La función devolverá un arreglo con las siguientes llaves (keys):
['arreglo_base'] contendrá el arreglo resultante de las modificaciones hechas a $La_base.

['fallos']['comp_valor_igual'] contendrá las estructuras que en la comparación de sus valores no resultaron iguales (==) , los elementos serán strings de la forma '$La_elementos[elemento 1] [elemento n]', para esto debe estar habilitado en el parámetro $comport el bit #1, sino no aparece este elemento en el arreglo.

['fallos']['comp_valor_diferente'] contendrá las estructuras que en la comparación de sus valores no resultaron diferentes (!=) , los elementos serán strings de la forma '$La_elementos[elemnto1] [elemento n]', para esto debe estar habilitado en el parámetro $comport el bit #2, sino no aparece este elemento en el arreglo.

['fallos']['comp_valor_noident'] contendrá las estructuras que en la comparación de sus valores no resultaron no idénticos (!==) , los elementos serán strings de la forma '$La_elementos[elemento1] [elemento n]', para esto debe estar habilitado en el parámetro $comport el bit #3, sino no aparece este elemento en el arreglo.

['fallos']['comp_valor_menor'] contendrá las estructuras que en la comparación de sus valores no resultaron menor que (<) , los elementos serán strings de la forma '$La_elementos[elemento1] [elemento n]', para esto debe estar habilitado en el parámetro $comport el bit #4, sino no aparece este elemento en el arreglo.

['fallos']['comp_valor_mayor'] contendrá las estructuras que en la comparación de sus valores no resultaron mayor que (>) , los elementos serán strings de la forma '$La_elementos[elemento1] [elemento n]', para esto debe estar habilitado en el parámetro $comport el bit #5, sino no aparece este elemento en el arreglo.

['fallos']['comp_valor_menorigual'] contendrá las estructuras que en la comparación de sus valores no resultaron menor o igual que (<=) , los elementos serán strings de la forma '$La_elementos[elemento1][elemento n]', para esto debe estar habilitado en el parámetro $comport el bit #6, sino no aparece este elemento en el arreglo.

['fallos']['comp_valor_mayorigual'] contendrá las estructuras que en la comparación de sus valores no resultaron mayor o igual que (>=) , los elementos serán strings de la forma '$La_elementos[elemento1][elementon]', para esto debe estar habilitado en el parámetro $comport el bit #7, sino no aparece este elemento en el arreglo.

['fallos']['comp_estruct_imgident'] contendrá las estructuras que no se encontraron en $La_base, los elementos serán strings de la forma '$La_elementos[elemento 1][elemento n]', para esto debe estar habilitado en el parámetro $comport el bit #8, sino no aparece este elemento en el arreglo, esto no quiere decir que la operación se detenga cuando la estructura no sea idéntica, solo se registra que no existe el elemento en el arreglo base y se continua el recorrido por el resto del arreglo.

si los parámetros $La_elementos y $La_base son idénticos el procedimiento retornará null.

public static function arregloEliminarRecursivo($La_elementos , &$La_base , &$comport = 0 , &$La_estruct_result = NULL , $Ls_estruct_elem = NULL , $Li_nivel = 0)

Procedimiento para eliminar valores de un arreglo ($La_base), basándose en la estructura de otro arreglo como referencia ($La_elementos), esta función resulta solo con php4 o superior y trabaja con arreglos asociativos. Los parámetros se explican a continuación:

$La_elementos (array) es un arreglo asociativo con los elementos que se quieren eliminar en el arreglo $La_base (base), esta matriz contendrá la estructura exacta en la que se encuentra el elemento que se quiere eliminar, es como un mapa, una imagen de la matriz y su rama o elemento que se quiere eliminar, llamémosle de ahora en adelante "estructura imagen", en caso de que toda una rama del arreglo se quiera eliminar basta con poner 'elemento'=array() y se eliminara ese elemento junto a la rama que el constituye, el primer elemento del arreglo $La_elementos será el primer elemento a buscar en el arreglo $La_base , y así sucesivamente.

$La_base (base) es un arreglo asociativo al cual se le eliminaran elementos que son representados en la estructura del arreglo $La_elementos.

$comport define condicionantes en el comportamiento de este procedimiento en su gestión con los elementos de los arreglos $La_elementos y $La_base, su valor puede ser un string representando un binario ('011000001') o un decimal, en caso de ser un decimal se convierte a binario, donde si se escoge una condición y esta no se cumple, el elemento base no es eliminado. Las condicionantes quedaran representadas de la siguiente manera:

1er bit (bit menos significativo, extremo derecho) en cero (desactivado) no se tiene en cuenta, en 1 (activado) corresponde a eliminar los elementos si sus valores son iguales (==)

2do bit en cero (desactivado) no se tiene en cuenta, en 1 (activado) corresponde a eliminar los elementos si sus valores son idénticos (===)

3er bit en cero (desactivado) no se tiene en cuenta, en 1 (activado) corresponde a eliminar los elementos si sus valores son diferentes (!= , <>)

4to bit en cero (desactivado) no se tiene en cuenta, en 1 (activado) corresponde a eliminar los elementos si sus valores son no idénticos (!==)

5to bit en cero (desactivado) no se tiene en cuenta, en 1 (activado) corresponde a eliminar los elementos si el valor de la imagen es menor que el de la base (<)

6to bit en cero (desactivado) no se tiene en cuenta, en 1 (activado) corresponde a eliminar los elementos si el valor de la imagen es mayor que el de la base (>)

7mo bit en cero (desactivado) no se tiene en cuenta, en 1 (activado) corresponde a eliminar los elementos si el valor de la imagen es menor o igual que el de la base (<=)

8vo bit en cero (desactivado) no se tiene en cuenta, en 1 (activado) corresponde a eliminar los elementos si el valor de la imagen es mayor o igual que el de la base (>=)

9no bit en cero (desactivado) no se tiene en cuenta, en 1 (activado) corresponde a registrar en los resultados de esta función los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores

los valores siguientes son excepciones validas de aclarar

binario: 000000000 (dec 0) elimina los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir elimina todo sin criterios de comparación.

binario: 100000000 (dec 256) elimina los valores sin tener en cuenta las condicionantes de lógica de comparación, es decir elimina todo sin criterios de comparación pero manteniendo el criterio de registrar en los resultados de esta función los elementos existentes en el arreglo imagen que no existen en el arreglo base, solo los elementos no sus valores.

binario: 110000000+1 (dec mayor que 384)retorna un valor null.

si la lógica de los bit del 1ro al 8vo activados equivalen a eliminar todo el arreglo o existe redundancia, la función retorna un valor null, ej:

00000101 o 001010000.

entre el 1er y el 8vo bit solo pueden coexistir las combinaciones 2do y 3ro, o 2do y 5to, o 2do y 6to.

el 9no bit siempre puede estar activado, no pertenece a la lógica comparativa del lenguaje.

$La_estruct_result (array) es para el desarrollo interno del procedimiento y es un arreglo asociativo con los resultados de las operaciones internas de la del procedimiento, de introducirse un valor en este parámetro no se predice el comportamiento del procedimiento.

$Ls_estruct_elem (string) es para el desarrollo interno del procedimiento, ya que esta se llama de forma recursiva, no necesita que le pasen valores y es una cadena que va concatenando a través de los ciclos la estructura a eliminar de la matriz, de introducirse un valor en este parámetro no se predice el comportamiento del procedimiento.

$Li_nivel (int) es para el desarrollo interno del procedimiento, es un entero que cuenta los niveles o dimensiones por los que pasa el procedimiento, de introducirse un valor en este parámetro no se predice el comportamiento del procedimiento.

El procedimiento devolverá un arreglo con las siguientes llaves (keys):

['arreglo_base'] contendrá el arreglo resultante de las modificaciones hechas a $La_base, si al arreglo $La_base le fueron eliminados todos sus elementos quedando un arreglo vacío, este elemento tendrá valor null.

['fallos']['comp_valor_igual'] contendrá las estructuras que en la comparación de sus valores no resultaron iguales (==) , los elementos serán strings de la forma '$La_elementos[elemento 1] [elemento n]', para esto debe estar habilitado en el parámetro $comport el bit #1, sino no aparece este elemento en el arreglo.

['fallos']['comp_valor_ident'] contendrá las estructuras que en la comparación de sus valores no resultaron idénticas (===) , los elementos serán strings de la forma '$La_elementos[elemento1] [elemento n]', para esto debe estar habilitado en el parámetro $comport el bit #2, sino no aparece este elemento en el arreglo.

['fallos']['comp_valor_diferente'] contendrá las estructuras que en la comparación de sus valores no resultaron diferentes (!=) , los elementos serán strings de la forma '$La_elementos[elemento 1] [elemento n]', para esto debe estar habilitado en el parámetro $comport el bit #3, sino no aparece este elemento en el arreglo.

['fallos']['comp_valor_noident'] contendrá las estructuras que en la comparación de sus valores no resultaron no idénticos (!==) , los elementos serán strings de la forma '$La_elementos[elemento1][elemento n]', para esto debe estar habilitado en el parámetro $comport el bit #4, sino no aparece este elemento en el arreglo.

['fallos']['comp_valor_menor'] contendrá las estructuras que en la comparación de sus valores no resultaron menor que (<) , los elementos serán strings de la forma '$La_elementos[elemento1][elemento n]', para esto debe estar habilitado en el parámetro $comport el bit #5, sino no aparece este elemento en el arreglo.

['fallos']['comp_valor_mayor'] contendrá las estructuras que en la comparación de sus valores no resultaron mayor que (>) , los elementos serán strings de la forma '$La_elementos[elemento1][elemento n]', para esto debe estar habilitado en el parámetro $comport el bit #6, sino no aparece este elemento en el arreglo.

['fallos']['comp_valor_menorigual'] contendrá las estructuras que en la comparación de sus valores no resultaron menor o igual que (<=) , los elementos serán strings de la forma '$La_elementos[elemento1][elemento n]', para esto debe estar habilitado en el parámetro $comport el bit #7, sino no aparece este elemento en el arreglo.

['fallos']['comp_valor_mayorigual'] contendrá las estructuras que en la comparación de sus valores no resultaron mayor o igual que (>=) , los elementos serán strings de la forma '$La_elementos[elemento1][elemento n]', para esto debe estar habilitado en el parámetro $comport el bit #8, sino no aparece este elemento en el arreglo.

['fallos']['comp_estruct_imgident'] contendrá las estructuras que no se encontraron en $La_base, los elementos serán strings de la forma '$La_elementos[elemento 1][elemento n]', para esto debe estar habilitado en el parámetro $comport el bit #9, sino no aparece este elemento en el arreglo, esto no quiere decir que la operación se detenga cuando la estructura no sea idéntica, solo se registra que no existe el elemento en el arreglo base y se continua el recorrido por el resto del arreglo.

Si los parámetros $La_elementos y $La_base son idénticos el procedimiento retornara null .

class_he_datfortxt.php

Este fichero contiene la clase EDatosFormatoTxt, que se encarga de ofrecer un grupo de procedimientos dedicados al manejo de datos con formatos, que son contenidos en ficheros txt, todos los procedimientos son static, a continuación explicamos la estructura y contenido de esta clase:

Namespace: Emod\Nucleo\Herramientas
Nombre de la clase: EDatosFormatoTxt
Herencia:

Propiedades:

Procedimientos:

public static function crearNuevaLinea($Ls_path_fich , $Ls_linea)
este procedimiento se encarga de crear una nueva línea en un fichero txt, y si el fichero no existe lo crea, esta función utiliza file_put_contents para su gestión. Los parámetros se explican a continuación:

$Ls_path_fich (string) es el path o camino del fichero al que se creará la nueva línea.

$Ls_linea (string) es el carácter o cadena de caracteres que componen la nueva línea a crear.

La función devolverá null si alguno de los parámetros tiene valor vacio, de lo contrario devolverá el valor que devuelve la función file_put_contents de php, al crear la nueva línea.

Recordar que cada línea a crear debe llevar el carácter de retorno de carro o línea, antes o después del contenido de la línea, sino se escribirá siempre al final de la última línea del fichero sin crearse una nueva línea.

public static function filtrarEliminarLineaDatos($La_estructura_llaves_base, $La_estructura_filtro, $Ls_separador, $Ls_path_fich, $Ls_llave_unica = null)

Procedimiento para eliminar líneas del fichero txt si estas coinciden con una regla de filtrado declarada en este mismo procedimiento. Los parámetros se explican a continuación:

$La_estructura_llaves_base (array) es un arreglo cuyos elementos serán las llaves que formaran parte de un arreglo asociativo, este arreglo asociativo será construido con estos elementos como llave y las secciones de una línea del fichero txt como valor de estas llaves, es como una matriz de llaves para los valores que se contienen en cada sección de cada línea seccionada con separadores, perteneciente al fichero a filtrar; los resultados de la combinación de estas dos estructuras se hace con array_combine por lo que el resultado de este procedimiento se rige por la gestión de esta función array_combine de php. A continuación un ejemplo de este parámetro:

ejemplo para una línea del fichero con la siguiente estructura
seccion1::seccion2::seccion3::seccionN y un arreglo $La_estructura_llaves_base = array(rojo,verde,azul,amarillo) se formarán los pares:

```
$arregloresult['rojo']= 'seccion1'
$arregloresult['verde']= 'seccion2'
$arregloresult['azul']= 'seccion3'
$arregloresult['amarillo']= 'seccionN'.
```

$La_estructura_filtro (array) es un arreglo con los pares $llave -> $valor que hacen función de filtro o regla de filtrado, para eliminar la o las líneas de datos; estos pares $llave -> $valor deben coincidir en la o las líneas de fichero a eliminar, puede ser un par $llave -> $valor o varios, si son varios pares $llave -> $valor como regla de filtrado para eliminar la línea de fichero, tienen que coincidir todos para que sea eliminada la línea. Tomando como referencia el ejemplo expuesto para el parámetro $La_estructura_llaves_base explicado anteriormente, con estos elementos $llave -> $valor de la estructura de filtro, se compararán cada uno de los elementos de la estructura $arregloresult existente en el ejemplo anterior.

$Ls_separador (string) es el string separador de los elemento de la línea del fichero txt, en el ejemplo expuesto para el parámetro $La_estructura_llaves_base explicado anteriormente, se tomo como separador '::'.

$Ls_path_fich (string) es el path del fichero txt al que se hará el filtrado.

$Ls_llave_unica (string) define una llave de las contenidas en el arreglo $La_estructura_filtro como condición para que cuando esta llave y su valor sean encontrados en la línea del txt, se detenga la búsqueda por las líneas restantes del fichero, es decir se detenga el proceso de filtrado y se pase a al termino de la gestión de este procedimiento, si su valor es vacío no se tomara en cuenta como condición, esta llave puede estar sola o acompañada de otras en $La_estructura_filtro.

El procedimiento devolverá:

retorna true si encuentra elemento al que aplicar la condición de filtrado y lo realiza satisfactoriamente.

retorna false si encuentra elemento al que aplicar la condición de filtrado y no le es posible realizar los cambios en el fichero.

retorna null si se le pasan valores incompatibles a los parámetros del procedimiento, o no encuentra elemento al que aplicar la condición de filtrado.

public static function filtrarLeerLineaDatos($La_estructura_llaves_base, $La_estructura_filtro, $Ls_separador, $Ls_path_fich, $Ls_llave_unica = null)

Procedimiento para leer líneas del fichero txt si estas coinciden con una regla de filtrado declarada en este mismo procedimiento. Los parámetros se explican a continuación:

$La_estructura_llaves_base (array) es un arreglo cuyos elementos serán las llaves que formaran parte de un arreglo asociativo, este arreglo asociativo será construido con estos elementos como llave y las secciones de una línea del fichero txt como valor de estas llaves, es como una matriz de

llaves para los valores que se contienen en cada sección de cada línea seccionada con separadores, perteneciente al fichero a filtrar; los resultados de la combinación de estas dos estructuras se hace con array_combine por lo que el resultado de este procedimiento se rige por la gestión de esta función array_combine de php. A continuación un ejemplo de este parámetro:

ejemplo para una línea del fichero con la siguiente estructura
seccion1::seccion2::seccion3::seccionN y un arreglo $La_estructura_llaves_base = array(rojo,verde,azul,amarillo) se formarán los pares:

```
$arregloresult['rojo']= 'seccion1'
$arregloresult['verde']= 'seccion2'
$arregloresult['azul']= 'seccion3'
$arregloresult['amarillo']= 'seccionN'.
```

$La_estructura_filtro (array) es un arreglo con los pares $llave -> $valor que hacen función de filtro o regla de filtrado, para leer la o las líneas de datos; estos pares $llave -> $valor deben coincidir en la o las líneas de fichero a leer, puede ser un par $llave -> $valor o varios, si son varios pares $llave -> $valor como regla de filtrado para leer la línea de fichero, tienen que coincidir todos para que sea leída la línea. Tomando como referencia el ejemplo expuesto para el parámetro $La_estructura_llaves_base explicado anteriormente, con estos elementos $llave -> $valor de la estructura de filtro, se compararán cada uno de los elementos de la estructura $arregloresult existente en el ejemplo anterior.

$Ls_separador (string) es el string separador de los elemento de la línea del fichero txt, en el ejemplo expuesto para el parámetro $La_estructura_llaves_base explicado anteriormente, se tomo como separador '::'.

$Ls_path_fich (string) es el path del fichero txt al que se hará el filtrado.

$Ls_llave_unica (string) define una llave de las contenidas en el arreglo $La_estructura_filtro como condición para que cuando esta llave y su valor sean encontrados en la línea del txt, se detenga la búsqueda por las líneas restantes del fichero, es decir se detenga el proceso de filtrado y se pase a al termino de la gestión de este procedimiento, si su valor es vacío no se tomara en cuenta como condición, esta llave puede estar sola o acompañada de otras en $La_estructura_filtro.

El procedimiento devolverá:

retorna arreglo no asociativo donde cada uno de sus elementos contendrá como valor un string, estos strings serán las líneas del txt que coincidieron con el filtrado.

retorna null si se le pasan valores incompatibles a los parámetros del procedimiento, o no encuentra elemento al que aplicar la condición de filtrado.

public static function fltrarModificarLineaDatos($La_estructura_llaves_base, $La_estructura_filtro, $La_estructura_sustituta, $Ls_separador, $Ls_path_fich, $Ls_llave_unica = null)

Procedimiento para modificar líneas del fichero txt si estas coinciden con una regla de filtrado declarada en este mismo procedimiento. Los parámetros se explican a continuación:

$La_estructura_llaves_base (array) es un arreglo cuyos elementos serán las llaves que formaran parte de un arreglo asociativo, este arreglo asociativo será construido con estos elementos como llave y las secciones de una línea del fichero txt como valor de estas llaves, es como una matriz de llaves para los valores que se contienen en cada sección de cada línea seccionada con separadores, perteneciente al fichero a filtrar; los resultados de la combinación de estas dos estructuras se hace con array_combine por lo que el resultado de este procedimiento se rige por la gestión de esta función array_combine de php. A continuación un ejemplo de este parámetro:

ejemplo para una línea del fichero con la siguiente estructura
seccion1::seccion2::seccion3::seccionN y un arreglo $La_estructura_llaves_base = array(rojo,verde,azul,amarillo) se formarán los pares:

```
$arregloresult['rojo']= 'seccion1'
$arregloresult['verde']= 'seccion2'
$arregloresult['azul']= 'seccion3'
$arregloresult['amarillo']= 'seccionN'.
```

$La_estructura_filtro (array) es un arreglo con los pares $llave -> $valor que hacen función de filtro o regla de filtrado, para modificar la o las líneas de datos; estos pares $llave -> $valor deben coincidir en la o las líneas de fichero a modificar, puede ser un par $llave -> $valor o varios, si son varios pares $llave -> $valor como regla de filtrado para modificar la línea de fichero, tienen que coincidir todos para que sea modificada la línea. Tomando como referencia el ejemplo expuesto para el parámetro $La_estructura_llaves_base explicado anteriormente, con estos elementos $llave -> $valor de la estructura de filtro, se compararán cada uno de los elementos de la estructura $arregloresult existente en el ejemplo anterior. (importante) en caso de querer modificar valores en todas las líneas del fichero, este parámetro soporta el valor '*', que le indica al procedimiento que realice la sustitución en todas las líneas del fichero sin hacer filtrado.

$La_estructura_sustituta (array) es un arreglo con los pares $llave -> $valor que se sustituirán en la línea del txt si coincide la combinación de filtrado, estos pares pueden o no, ser parte de los pares $llave -> $valor de la combinación de filtrado, es decir, los pares $llave -> $valor que se encuentran en el parámetro $La_estructura_filtro.

$Ls_separador (string) es el string separador de los elemento de la línea del fichero txt, en el ejemplo expuesto para el parámetro $La_estructura_llaves_base explicado anteriormente, se tomo como separador '::'.

$Ls_path_fich (string) es el path del fichero txt al que se hará el filtrado.

$Ls_llave_unica (string) define una llave de las contenidas en el arreglo $La_estructura_filtro como condición para que cuando esta llave y su valor sean encontrados en la línea del txt, se detenga la búsqueda por las líneas restantes del fichero, es decir se detenga el proceso de filtrado y se pase a al termino de la gestión de este procedimiento, si su valor es vacio no se tomara en cuenta como condición, esta llave puede estar sola o acompañada de otras en $La_estructura_filtro.

El procedimiento devolverá:

retorna true si encuentra elemento al que aplicar la condición de filtrado y lo realiza satisfactoriamente.

retorna false si encuentra elemento al que aplicar la condición de filtrado y no le es posible realizar los cambios en el fichero.

retorna null si se le pasan valores incompatibles a los parámetros del procedimiento, o no encuentra elemento al que aplicar la condición de filtrado.

class_eutiles.php

Este fichero contiene la clase de procedimientos estáticos Utiles. Esta clase está compuesta por propiedades, y procedimientos estáticos para la gestión de entidades Útiles, desde la inclusión del script o clase como fichero hasta el retorno de referencias o clonaciones a entidades clases u objetos, gestiona y sirve como contenedor de estas entidades, una opción clara para la inyección de dependencias. En su gestión las entidades Útiles no son incluidas o instanciadas hasta el momento que se hace la petición de la instancia para su uso. Para un mejor entendimiento de cómo se gestionan las entidades Útiles se debe consultar cada propiedad y procedimiento de esta clase.

A continuación mostramos una lista de las propiedades y procedimientos que esta clase pone a disposición de los códigos clientes del Sockeletom:

Namespace: Emod\Nucleo
Nombre de la clase: Utiles
Herencia: use \Emod\Nucleo\Herramientas\GECO

Propiedades:

private static $EEoNucleo = null ;
puntero al objeto EEoNucleo ;

private static $lbGestionSeguridad = false ;

es un boolean perteneciente a la sección de seguridad de esta clase Utiles, que define si se gestionaran los útiles bajo condiciones de seguridad, true se gestiona con seguridad, false no se gestiona con seguridad, estas condiciones de seguridad estarán declaradas en esta clase y son gestionadas por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen elementos de seguridad en el parámetro de datos entrante .

private static $lbPropietarioEntidad = true ;
es un boolean perteneciente a la sección de seguridad de esta clase Utiles, que define si al ingresar entidades útiles en esta clase, se registra el proceso que ingresa la entidad útil como su propietario, esto tiene como objetivo que la opción de eliminar entidades útiles sea posible solo por su propietario o por cualquier proceso que decida eliminar una entidad útil del ámbito de gestión de esta clase, true se registra el proceso propietario, false no se registra el proceso propietario, este elemento de seguridad es gestionado por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen parámetros de seguridad en el parámetro de datos entrante. No está afectado por la propiedad $lbGestionSeguridad ya que aunque este no esté activado, con valor true, si $lbPropietarioEntidad si está activado, con valor true, se lleva a cavo su función y objetivo. El proceso núcleo siempre tiene permiso de eliminar una entidad útil determinada.

private static $lsAmbitoSeguridad = null ;
es un string perteneciente a la sección de seguridad de esta clase Utiles, que define el ámbito de seguridad a aplicar a la lista de entidades útiles declaradas a permitir o denegar su gestión por parte de esta clase Utiles, sus valores posibles son “permisivo” y ” restrictivo” , está directamente relacionada con el elemento de seguridad $laDatosSeguridad que se expondrá más adelante, el valor “permisivo” permite la gestión de cualquier entidad útil excepto las declaradas en $laDatosSeguridad, el valor “restrictivo” permite la gestión solo de las entidades útiles declaradas en $laDatosSeguridad y deniega las demás entidades útiles, este elemento de seguridad es gestionado por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen parámetros de seguridad en el parámetro de datos entrante. Esta propiedad está afectada por la propiedad $lbGestionSeguridad, si no está activada la propiedad $lbGestionSeguridad, con valor true, se omite la existencia y valor de esta propiedad $lsAmbitoSeguridad.

private static $lbActualizarDatosSeguridad = null ;
es un boolean perteneciente a la sección de seguridad de esta clase Utiles, que define la posibilidad o no de que los procesos puedan actualizar la lista de entidades útiles declaradas a permitir o denegar su gestión por parte de esta clase Utiles. sus valores posibles son “true” y ” false” , está directamente relacionada con el elemento de seguridad $laDatosSeguridad que se expondrá más adelante, el valor “true” permite que cualquier proceso actualice la lista de entidades útiles declaradas a permitir o denegar su gestión por parte de esta clase Utiles, false no permite lo anterior, el proceso núcleo si tiene permisos para actualizar la lista antes mencionada independientemente del valor de esta propiedad , este elemento de seguridad es gestionado por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen

parámetros de seguridad en el parámetro de datos entrante. Esta propiedad está afectada por la propiedad $lbGestionSeguridad, si no está activada la propiedad $lbGestionSeguridad, con valor true, se omite la existencia y valor de esta propiedad $lbActualizarDatosSeguridad.

private static $laDatosSeguridad = array() ;
es un array perteneciente a la sección de seguridad de esta clase Utiles, que define la lista de entidades útiles declaradas a permitir o denegar su gestión por parte de esta clase Utiles, está directamente relacionada con las propiedades $lsAmbitoSeguridad y $lbActualizarDatosSeguridad de esta clase Utiles. Este elemento de seguridad es a nivel de clase y no de entidades útiles específicas, se da permiso o no a la clase, no a una entidad correspondiente a la clase. Este elemento de seguridad es gestionado por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen parámetros de seguridad en el parámetro de datos entrante. Esta propiedad está afectada por la propiedad $lbGestionSeguridad, si no está activada la propiedad $lbGestionSeguridad, con valor true, se omite la existencia y valor de esta propiedad $laDatosSeguridad. Existe la posibilidad de poner el valor * (asterisco) que significa todos los elementos posibles, se puede hacer a nivel de $laDatosSeguridad o a nivel de un namespace. A continuación se muestra la estructura de este arreglo:

$laDatosSeguridad:
namespace_entidades_utiles 1:
clase_entidades_utiles 1
clase_entidades_utiles 2
clase_entidades_utiles N
namespace_entidades_utiles 2: *
namespace_entidades_ utiles N:
clase_entidades_utiles 1
clase_entidades_utiles 2
clase_entidades_utiles N

también $laDatosSeguridad puede tener el valor *

$laDatosSeguridad: *

namespace_entidades_utiles (string) es el nombre del namespace al que pertenecen las entidades útiles. (obligatorio)

clase_entidades_utiles (string) es el nombre de la clase a la que pertenecen las entidades útiles. (obligatorio)

$lsPathDirRaizEntidades = null ;
propiedad para contener el valor del path del directorio base donde se alojarán las entidades útiles en el sistema Sockeletom, es el directorio que el sistema propone como contenedor de las entidades útiles que se utilicen por el núcleo del sistema y los demás códigos clientes del Sockeletom. Este

parámetro tiene a su vez como path raíz al directorio esquelemod, perteneciente a la estructura de directorios del sistema Sockeletom. Tomando como ejemplo el valor ¨ utiles ¨ para esta propiedad, el valor absoluto del path donde se alojaran las entidades útiles sería:
raíz_sistema_Sockeletom/esquelemod/utiles

private static $iniciacion = null ;
propiedad para chequear la iniciación de esta clase

private static $laEntidades = array() ;
es un arreglo que contendrá las entidades útiles y sus características operacionales, entre estas características están:

namespace: (string) es el nombre del namespace al que pertenece la entidad útil (obligatorio)

clase: (string) es el nombre de la clase a la que pertenece la entidad útil (obligatorio)

'path_entidad_clase': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece la entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en esta clase la entidad útil a gestionar, si el fichero no existe no se ingresa la entidad útil en esta propiedad $laEntidades de esta clase. (obligatorio)

'tipo_entidad': (string) contiene el tipo de entidad, es decir si es de tipo clase u objeto la entidad útil a gestionar, sus posibles valores son ‘clase’ o ‘objeto. (obligatorio)

'iniciacion': (string) es la forma como se iniciará una entidad útil, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades útiles al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades útiles le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades útiles le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad útil sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad útil del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento ['instancias'][id_util]['parámetros_iniciacion'] de la clase base, por lo que cada entidad útil puede tener sus valores propios, excepto en caso de que la entidad útil sea de tipo clase (tipo_entidad = 'clase') que con el primer id_util se inicia la clase base. (opcional)

'proceso_propietario': (string) es el proceso que hace el ingreso de la clase entidad útil, este valor el procedimiento lo toma del objeto EEoNucleo conformado por el gedee del proceso en ejecución,

seguido de :: y el id del proceso en ejecución (self::$EEoNucleo->gedeeProcesoEjecucion . '::' . self::$EEoNucleo->idProcesoEjecucion()) y es para definir el único proceso que puede eliminar la entidad namespace/clase de este arreglo, este elemento está sujeto a la propiedad $lbPropietarioEntidad de la sección de seguridad de esta clase Utiles. Cuando el proceso propietario de la entidad namespace/clase decide eliminar esta entidad, se eliminará junto con todas las instancias contenidas de esta namespace/clase.

'instancias': (array) contiene las instancias que se creen de la clase correspondiente, contiene las instancias y los datos correspondientes a cada instancia, entre estos datos están:

id_entidad_util: (string) es el nombre o identificador de la entidad útil (obligatorio), los id_entidad_util pueden ser ingresados en el momento que se ingresa el namespace/clase de la clase entidad_util o posteriormente haciendo una petición de ingreso de instancia de una entidad_util a través del procedimiento * ingresarEntidad * de esta clase.

'registro': (boolean) en caso de 'tipo_entidad' tener valor 'clase' entonces esta propiedad contiene el valor true si ha sido solicitada al menos una vez.

'objeto': (&obj) en caso de 'tipo_entidad' tener valor 'objeto' entonces esta propiedad contiene la referencia al objeto, entidad útil a gestionar.

'proceso_propietario': (string) es el proceso que hace el ingreso de la entidad útil, este valor el procedimiento lo toma del objeto EEoNucleo conformado por el gedee del proceso en ejecución, seguido de :: y el id del proceso en ejecución (self::$EEoNucleo->gedeeProcesoEjecucion . '::' . self::$EEoNucleo->idProcesoEjecucion()) y es para definir el único proceso que puede eliminar la entidad instancia de este arreglo, este elemento está sujeto a la propiedad $lbPropietarioEntidad de la sección de seguridad de esta clase Utiles. Se aplica solo a la entidad útil específica, no tiene que ser el mismo propietario que el de la entidad namespace/clase de esta entidad Utiles.

'datos': es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad útil en esta clase Utiles; estos datos se crean en el momento del ingreso de la entidad útil sin que permita cambios posteriores.

la estructura del arreglo quedaría así:

```
$laEntidades = array(
                    'nombre namespace' => array(
                                        'nombre clase' => array(
                                                            'path_entidad_clase' => 'path_entidad_clase',
                                                            'tipo_entidad' => 'tipo_entidad',
                                                            ‘iniciacion’ => ‘procedimiento’
                                                            'proceso_propietario' => 'proceso_que_ingresa_la_entidad',
                                                            'instancias' => array(
                                                                        'id_entidad_util1' => array(
                                                                                    'registro' => 'true o false,
                                                                                    'objeto' => &objeto,
```

```
                                                    'proceso_propietario' => 'proceso_que_ingresa_la_entidad',
                                                    'datos' => array( 'dato1' => 'valor_dato' , 'datoN' =>datoN')
                                                        ) ,
                                            'id_entidad_utilN' => array(
                                                    'registro' => 'true o false,
                                                    'objeto' => &objeto,
                                                    'proceso_propietario' => 'proceso_que_ingresa_la_entidad',
                                                    'datos' => array( 'dato1' => 'valor_dato' , 'datoN' =>datoN')
                                                        )
                                                )
                                        )
                                )
                        )
```

Procedimientos:

public static function iniciacion($La_configuracion_nucleo_utiles)
es el procedimiento encargado de iniciar el funcionamiento de esta clase y además guardar en las propiedades de esta clase los elementos existentes como valores de la sección útiles propuestas por el sistema Sockeletom en la configuración del sistema Sockeletom. Además realiza el ingreso de las entidades útiles propuestas a gestionar en la configuración del sistema Sockeletom. Para un mejor entendimiento de los parámetros que la configuración del sistema proporciona a este procedimiento, debe consultarse la sección útiles del fichero de configuración del sistema Sockeletom en este documento.

este procedimiento retorna null si ya estaba inicializada la clase Utiles , retorna true si la gestión es exitosa, detiene el proceso php si los parámetros son incompatibles con el procedimiento, emitiendo un error fatal (exit).

public static function pathRaizEntidades ()
es el procedimiento encargado de brindar, retornar el path del directorio raíz para los recursos de útiles en el sistema Sockeletom. Este será el fragmento de path que completará el path del directorio base de los recursos de útiles, del directorio que el sistema propone como contenedor de los recursos de útiles que se utilicen por el núcleo del sistema y los demás códigos clientes del Sockeletom. Este parámetro tiene además como path raíz al directorio esquelemod, pertenecientes a la estructura de directorios del sistema Sockeletom. Tomando como ejemplo el valor ¨ utiles ¨, quedará como resultado el path:
raíz_sistema_Sockeletom/esquelemod/utiles

este procedimiento retorna el path gestionado o null en caso de no existir el path a gestionar.

public static function actualizarDatosSeguridadEntidades ($La_datos , $Ls_tipo_actualizacion)
es el procedimiento encargado de gestionar la actualización de la propiedad self:: $laDatosSeguridad de esta clase Utiles, la gestión de actualización está sujeta al valor de la

propiedad self::$lbActualizarDatosSeguridad perteneciente a la sección de seguridad de esta clase Utiles. Los parámetros se explican a continuación:

$La_datos: (array) es el arreglo con los datos a actualizar.

$Ls_tipo_actualizacion: (string) es el tipo de actualización a realizar, sus posibles valores son:

'renovar': es la sustitución del arreglo $La_datos por el arreglo actual en self::$laDatosSeguridad.

'adicionar': es la adición del contenido del arreglo $La_datos en el arreglo actual self:: $laDatosSeguridad.

'eliminar': es la eliminación del contenido del arreglo $La_datos en el arreglo actual self:: $laDatosSeguridad.

este procedimiento retorna true si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

public static function permisionEntidad ($Ls_namespace , $Ls_clase)
es el procedimiento encargado de gestionar si una clase entidad útil determinada tiene permiso de ser gestionado por esta clase Utiles, el permiso es a nivel de clase y no de entidades específicas de esta, se da permiso o no a la clase no a una entidad correspondiente a la clase; la gestión de chequeo de permisos está sujeto al valor de las propiedades self::$lbGestionSeguridad, self:: $lsAmbitoSeguridad y self::$laDatosSeguridad pertenecientes a la sección de seguridad de esta clase Utiles; específicamente chequea las clases permitidas o denegadas en self::$laDatosSeguridad según el self::$lsAmbitoSeguridad declarado. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la clase entidad útil a chequear su permiso de gestión (obligatorio)

$Ls_clase: (string) es el nombre de la clase a la que pertenece la clase entidad útil a chequear su permiso de gestión (obligatorio)

este procedimiento retorna true si la clase entidad útil tiene permiso a ser gestionada por esta clase, de lo contrario retorna null.

public static function gestionControlIngresoEntidad ($Ls_path_control_util , $Ls_referencia_path_control = 'relativo' , $Ls_namespace = null , $Ls_clase = null , $Ls_path_entidad_clase = null , $Ls_referencia_path_entidad = null , $Ls_tipo_entidad = null , $La_instancias = null , $Ls_iniciacion = null)

es el procedimiento encargado de gestionar el ingreso de una entidad útil al contenedor $laEntidades de esta clase, utilizando entre sus parámetros un parámetro que hace referencia al path de un fichero control.

Se llama fichero control a un fichero o script php que se necesita su ejecución como parte de la gestión de ingreso de la entidad útil a esta clase Utiles. El control puede utilizarse para chequear, gestionar o poner a punto algún elemento necesario para el ingreso de la entidad útil a esta clase.

El fichero control es ejecutado (require_once) por este procedimiento y puede utilizarse en conjunto con los demás parámetros de este procedimiento o gestionar el mismo la inclusión (require_once) de la clase entidad útil y retornar el mismo los parámetros que necesita este procedimiento para la gestión del ingreso de la entidad útil a esta clase.

Las diferentes combinaciones que pueden utilizarse devienen diferentes funcionalidades que son:

Funcionalidad 1 donde el control debe ejecutarse y las características de la entidad se pasan como parámetros independientes del control. Si se declaran parámetros para el control y a la vez se declaran los parámetros
$Ls_namespace, $Ls_clase, $Ls_path_entidad_clase, $Ls_referencia_path_entidad, $Ls_tipo_entidad , $La_instancias = null , $Ls_iniciacion este procedimiento incluirá el fichero control y además utilizará los parámetros antes mencionados para la gestión de ingreso de la entidad útil , no importa que el control retorne también los parámetros antes mencionados porque este procedimiento da prioridad a los valores de sus parámetros y nunca evalúa los parámetros homólogos retornados por el control. Tienen que existir todos los parámetros antes mencionados para que esta opción se ejecute, si solo uno de estos parámetros es vacio entonces se pasa a las siguientes funcionalidades.

antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de la entidad útil, si no existe el fichero el procedimiento no ingresa la entidad útil y retorna null. Ingresar una entidad útil no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de esta entidad útil. Los parámetros se explican a continuación:

$Ls_path_control_util : (string) es el path del fichero control a ejecutar para el ingreso de la entidad útil. (obligatorio)

$Ls_referencia_path_control: (string) es la forma que se hace referencia a $Ls_path_control_util, tiene tres posibles valores, *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el $Ls_path_control_util relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el $Ls_path_control_util relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el $Ls_path_control_util de

forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad útil.

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad útil.

$Ls_path_entidad_clase: (string) contiene el path del fichero que alberga el código de la clase a la que pertenece la entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, si el fichero no existe no se ingresa la entidad útil al arreglo.

$Ls_referencia_path_entidad: (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el $Ls_path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el $Ls_path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el $Ls_path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

$Ls_tipo_entidad: (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

$Ls_iniciacion: (string) es la forma como se iniciará cada id_entidad_util, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de $Ls_iniciacion son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades útiles al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades útiles le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades útiles le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad útiles sea de tipo clase ($Ls_tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad útil del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento [$La_instancias][id_entidad_util] ['parametros_iniciacion'] de la clase base, por lo que cada entidad útil puede tener sus valores propios, excepto en caso de que la entidad útil sea de tipo clase ($Ls_tipo_entidad = 'clase') que con el primer id_ entidad_util se inicia la clase base. Los elementos [$La_instancias] [id_entidad_util]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

$La_instancias (array) es un arreglo con los identificadores de entidades útiles a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_util. (obligatorio)

id_entidad_util (string) es un nombre o identificador de cada entidad útil instancia de la clase correspondiente. y a su vez este id_entidad_util es un arreglo asociativo al que se puede insertar una llave de nombre 'parametros_iniciacion' y otra de nombre 'datos', posteriormente este arreglo id_entidad_util se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_util como sea necesario ya que la entidad útil puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad útil (id_entidad_util), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad útil a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad_util declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad útil es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad útil en esta clase Utiles; estos datos se crean en el momento del ingreso de la entidad útil sin que permita cambios posteriores. (opcional)

Para el resto de las funcionalidades. Es necesario volver a aclarar que si se declaran parámetros en este procedimiento para el fichero control y al menos uno de los restantes es vacio entonces este procedimiento ejecuta (require_once) el control, desecha los demás parámetros pasados a este procedimiento y espera en forma de retorno(return) desde el fichero control requerido(require_once), un arreglo asociativo con las posibles siguientes estructuras:

Funcionalidad 2 donde el control debe ejecutarse, y este mismo ya incluyó la clase y nos retorna las características de la entidad incluido la o las instancias(id_entidad_util)(objeto) de la clase, para que se ejecute esta funcionalidad esta estructura tiene que estar completa, si faltase algún elemento no se ejecutará esta funcionalidad, también tiene como condición que el elemento $La_resultado_require['referencia_path_entidad'] no exista, la responsabilidad de incuir la estructura correcta de los id_entidad_util instanciadas (objeto) o registradas(clase) es responsabilidad del control, también es obligatorio que exista al menos una id_entidad_util en la estructura de datos que se espera por retorno. La estructura que se espera por retorno es la siguiente:

$La_resultado_require['namespace'] (obligatorio)
$La_resultado_require['clase'] (obligatorio)
$La_resultado_require['path_entidad_clase'] (obligatorio)
$La_resultado_require['tipo_entidad'] (obligatorio)
$La_resultado_require['iniciacion'] (opcional)
$La_resultado_require['instancias'] (obligatorio)
$La_resultado_require['instancias'] ['id_entidad_util '] (obligatorio)
$La_resultado_require['instancias'] ['id_entidad_util '] ['objeto'] (opcional)
$La_resultado_require['instancias'] ['id_entidad_util'] ['registro'] (opcional)
$La_resultado_require['instancias'] ['id_entidad_util'] [' datos'] (opcional)

'namespace': (string) es el nombre del namespace al que pertenece la entidad útil. (obligatorio)

'clase': (string) es el nombre de la clase a la que pertenece la entidad útil. (obligatorio)

'path_entidad_clase': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad útil al arreglo $laEntidades de esta clase. (obligatorio)

'tipo_entidad': (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

'iniciacion': (string) es la forma como se iniciará cada id_entidad_util , este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades útiles al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades útiles le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades útiles le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad útil sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad útil del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento
['instancias'][id_entidad_util]['parametros_iniciacion'] de la clase base, por lo que cada entidad útil puede tener sus valores propios, excepto en caso de que la entidad útil sea de tipo clase (tipo_entidad = 'clase') que con el primer id_entidad_util se inicia la clase base. Los elementos

['instancias'][id_entidad_util]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

'instancias' (array) es un arreglo con los identificadores de entidades útiles a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_util. (obligatorio)

id_entidad_util (string) es un nombre o identificador de cada entidad útil instancia de la clase correspondiente. y a su vez este id_entidad_util es un arreglo asociativo al que se puede insertar una llave de nombre objeto, una llave de nombre registro, y otra de nombre datos, las llaves objeto, y registro no deben coexistir ya que la existencia de una determina la no existencia de la otra ,posteriormente este arreglo id_entidad_util se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_util como sea necesario ya que la entidad útil puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'objeto': (&obj) es un apuntador al objeto instancia de la clase entidad útil a contener en el contenedor $laEntidades de esta clase. Este elemento es obligatorio si el elemento tipo_entidad tiene como valor el string objeto y estrictamente se considera iniciado.

'registro': (boolean true) es para definir cuando una entidad útil a contener en el contenedor $laEntidades de esta clase, y de tipo_entidad clase (procedimientos estáticos) es iniciada . Este elemento es obligatorio si el elemento tipo_entidad tiene como valor el string clase y estrictamente se considera iniciada.

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad útil en esta clase Utiles; estos datos se crean en el momento del ingreso de la entidad útil sin que permita cambios posteriores. (opcional)

Funcionalidad 3 donde el control debe ejecutarse, y este nos retorna(return) los elementos o parámetros para ingresar una clase, y reserva al menos un id_entidad_util para su posterior instanciación, no debe tener id_entidades_utiles instanciadas(objeto) o registradas(clase), y es obligatorio que exista en los parámetros retornados por el control el elemento $La_resultado_require['referencia_path_entidad']. Entiéndase que esta funcionalidad solo realiza su gestión si no existe ningún id_entidad_util instanciado este factor es obligatorio. La estructura que se espera por retorno es la siguiente:

$La_resultado_require['namespace'] (obligatorio)
$La_resultado_require['clase'] (obligatorio)
$La_resultado_require['path_entidad_clase'] (obligatorio)
$La_resultado_require['referencia_path_entidad '] (obligatorio)

$La_resultado_require['tipo_entidad'] (obligatorio)
$La_resultado_require['iniciacion'] (opcional)
$La_resultado_require['instancias'] (obligatorio)
$La_resultado_require['instancias'] [id_entidad_util] (obligatorio)
$La_resultado_require['instancias'] [id_entidad_util] ['parametros_iniciacion'] (opcional)
$La_resultado_require['instancias'] [id_entidad_util] [' datos'] (opcional)

'namespace': (string) es el nombre del namespace al que pertenece la entidad útil. (obligatorio)

'clase': (string) es el nombre de la clase a la que pertenece la entidad útil. (obligatorio)

'path_entidad_clase': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad útil al arreglo $laEntidades de esta clase. (obligatorio)

'referencia_path_entidad': (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

'tipo_entidad': (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

'iniciacion': (string) es la forma como se iniciará cada id_entidad_util, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades útiles al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades útiles le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades útiles le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad útil sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad útil del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento['instancias'][id_entidad_util]['parametros_iniciacion'] de la clase base, por lo que cada entidad útil puede tener sus valores propios, excepto en caso de que la

entidad útil sea de tipo clase (tipo_entidad = 'clase') que con el primer id_entidad_util se inicia la clase base. Los elementos ['instancias'][id_entidad_util]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

'instancias' (array) es un arreglo con los identificadores de entidades útiles a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_util. (obligatorio)

id_entidad_util (string) es un nombre o identificador de cada entidad útil instancia de la clase correspondiente. y a su vez este id_entidad_util es un arreglo asociativo al que se puede insertar una llave de nombre 'parametros_iniciacion' y otra de nombre 'datos', posteriormente este arreglo id_entidad_util se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_util como sea necesario ya que la entidad útil puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad útil (id_entidad_util), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad útil a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad_util declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad útil es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad útil en esta clase Utiles; estos datos se crean en el momento del ingreso de la entidad útil sin que permita cambios posteriores. (opcional)

La funcionalidad que el control ejecutado devuelva los datos de la clase y sus elementos['instancias'][id_entidad_util] no instanciados, es por seguridad ya que existen otras formas para crear controles que retornen estructuras con sus id_entidad_util ya instanciados(objeto) o registrados(clase); Esta característica está abierta a cambios futuro en dependencia de la opinión mayoritaria de los usuarios del sistema Sockeletom.

antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de la entidad útil, si no existe el fichero el procedimiento no ingresa la entidad útil y retorna null. Ingresar una entidad útil no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de esta entidad útil.

este procedimiento retorna true si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

public static function gestionIngresoEntidad ($Ls_namespace , $Ls_clase , $Ls_path_entidad_clase , $Ls_referencia_path_entidad , $Ls_tipo_entidad , $La_instancias , $Ls_iniciacion = null)
es el procedimiento encargado de gestionar el ingreso de una entidad útil al contenedor $laEntidades de esta clase, antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de la entidad útil, si no existe el fichero el procedimiento no ingresa la entidad útil y retorna null. Ingresar una entidad útil no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de esta entidad útil. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad útil. (obligatorio)

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad útil. (obligatorio)

$Ls_path_entidad_clase: (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad útil al arreglo $laEntidades de esta clase. (obligatorio)

$Ls_referencia_path_entidad: (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

$Ls_tipo_entidad: (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

$Ls_iniciacion: (string) es la forma como se iniciará cada id_entidad_util, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades útiles al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades útiles le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades útiles le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad útil sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus

parámetros solo cuando se gestione por primera vez la primera entidad útil del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento $La_instancias[id_entidad_util]['parametros_iniciacion'] de la clase base, por lo que cada entidad útil puede tener sus valores propios, excepto en caso de que la entidad útil sea de tipo clase (tipo_entidad = 'clase') que con el primer id_ entidad_util se inicia la clase base. Los elementos $La_instancias[id_entidad_util]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

$La_instancias (array) es un arreglo con los identificadores de entidades útiles a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_util. (obligatorio)

id_entidad_util (string) es un nombre o identificador de cada entidad útil instancia de la clase correspondiente. y a su vez este id_entidad_util es un arreglo asociativo al que se puede insertar una llave de nombre 'parametros_iniciacion' y otra de nombre 'datos', posteriormente este arreglo id_entidad_util se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_util como sea necesario ya que la entidad útil puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad útil (id_entidad_util), si existe este parámetro, es arreglo y no es vacio entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad útil a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad_util declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad útil es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad útil en esta clase Utiles; estos datos se crean en el momento del ingreso de la entidad útil sin que permita cambios posteriores. (opcional)

public static function gestionIngresosUtiles ($La_propiedades_utiles)
es el procedimiento encargado de gestionar el ingreso de varias entidades útiles al contenedor $laEntidades de esta clase, este procedimiento desglosa un arreglo con datos de varias entidades útiles y realiza la gestión de ingreso según sus características como lo hacen individualmente los procedimientos self::gestionControlIngresoEntidad y self::gestionIngresoEntidad, es importante

aclarar que cada entidad útil será procesada estrictamente con los procedimientos self::gestionControlIngresoEntidad o self::gestionIngresoEntidad por lo que los datos de las entidades útiles tienen que cumplir con las condicionantes del procedimiento respectivo para que la gestión sea exitosa, antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de cada entidad útil, si no existe el fichero el procedimiento no ingresa la entidad útil y continua su gestión. Ingresar una entidad útil no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de esta entidad útil. Los parámetros se explican a continuación:

$La_propiedades_utiles: es un arreglo en el que los elementos llave de este arreglo son los namespaces de las entidades útiles a gestionar su ingreso en esta clase, la llave namespace representa a su vez un arreglo con las clases y algunas características de estas clases que finalmente son las entidades útiles declaradas, la llave clase representa a su vez un arreglo con algunas características necesarias para la correcta gestión de las entidades útiles, un ejemplo de esta estructura se expone a continuación:

$La_propiedades_utiles
namespace_entidad_util1:
clase_entidad_util 1:
'path_entidad_clase':
'referencia_path_entidad':
'path_control':
'referencia_path_control':
'tipo_entidad':
'iniciación':
'instancias':
id_entidad_util 1:
'parametros_iniciacion':
'datos':
id_entidad_util 2
'parametros_iniciacion':
id_entidad_util 3
id_entidad_util 4:
'datos':

namespace_entidad_util 2:
clase_entidad_util 1:
'path_entidad_clase':
'referencia_path_entidad':
'path_control':
'referencia_path_control':
'tipo_entidad':

```
                    'instancias':
                        id_entidad_util 1:
                            'datos':
                        id_entidad_util 2
                        id_entidad_util 3
                        id_entidad_util 4
    clase_entidad_util 2:
                    'path_entidad_clase':
                    'referencia_path_entidad':
                    'path_control':
                    'referencia_path_control':
                    'tipo_entidad':
                    'iniciación':
                    'instancias':
                        id_entidad_util 1:
                            'datos':
                        id_entidad_util 2:
                            'parametros_iniciacion':
                            'datos':
                        id_entidad_util 3:
                            'datos':
                        id_entidad_util 4:
                            'datos':
```

los elementos de este arreglo se explican a continuación:

namespace: (string) es el nombre del namespace al que pertenece la entidad útil. (obligatorio en dependencia del procedimiento gestor)

clase: (string) es el nombre de la clase a la que pertenece la entidad útil. (obligatorio en dependencia del procedimiento gestor)

'path_entidad_clase': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad útil al arreglo $laEntidades de esta clase. (obligatorio en dependencia del procedimiento gestor)

'referencia_path_entidad': (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio en dependencia del procedimiento gestor)

'path_control': es el path de un fichero que controlará condiciones y/o el inicio o inclusión de una entidad útil, el gestor de útiles espera un retorno de este control con un arreglo asociativo donde se encuentren los datos namespace_entidad_util, clase_entidad_util, id_entidad _util, path_entidad_clase, referencia_path_entidad , tipo_entidad , instancias con sus datos, o combinaciones de estos, este arreglo será procesado por esta clase Emod\Nucleo\Utiles que es quien gestiona y administra los útiles en el sistema Sockeletom. (obligatorio en dependencia del procedimiento gestor)

'referencia_path_control': (string) es la forma que se hace referencia a $Ls_path_control_util, tiene tres posibles valores, *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el $Ls_path_control_util relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el $Ls_path_control_util relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el $Ls_path_control_util de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio en dependencia del procedimiento gestor)

'tipo_entidad': (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio en dependencia del procedimiento gestor)

'iniciacion': (string) es la forma como se iniciará cada id_entidad_util, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades útiles al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades útiles le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades útiles le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad útil sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad útil del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un

arreglo no asociativo en el elemento ['instancias'][id_entidad_util]['parametros_iniciacion'] de la clase base, por lo que cada entidad útil puede tener sus valores propios, excepto en caso de que la entidad útil sea de tipo clase (tipo_entidad = 'clase') que con el primer id_entidad_util se inicia la clase base. Los elementos ['instancias'][id_entidad_util]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional en dependencia del procedimiento gestor)

'instancias': (array) es un arreglo con los identificadores de entidades útiles a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_util. (obligatorio en dependencia del procedimiento gestor)

id_entidad_util (string) es un nombre o identificador de cada entidad útil instancia de la clase correspondiente. y a su vez este id_entidad_util es un arreglo asociativo al que se puede insertar una llave de nombre parametros_iniciacion y otra de nombre datos, posteriormente este arreglo id_entidad_util se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_util como sea necesario ya que la entidad útil puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1 en dependencia del procedimiento gestor) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad útil (id_entidad_util), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad útil a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad_util declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad útil es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional en dependencia del procedimiento gestor)

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad útil en esta clase Utiles; estos datos se crean en el momento del ingreso de la entidad útil sin que permita cambios posteriores. (opcional en dependencia del procedimiento gestor)

se aconseja por parte del grupo de desarrollo del núcleo Sockeletom que los namespaces se declaren de forma absoluta, comenzando con '\ '

este procedimiento retorna true si la gestión de al menos una entidad útil fue llevada a cabo con éxito, de lo contrario retorna null.

```
public static function existenciaEntidad ( $Ls_namespace , $Ls_clase , $Ls_id_util = null )
```

es el procedimiento encargado de gestionar si existe una entidad útil determinada en el contenedor $laEntidades de esta clase, si existe una entidad útil a disposición de pedido y uso, esta entidad útil no tiene obligatoriamente que estar iniciada, es decir no tiene obligatoriamente que existir la instancia ya creada.

Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad útil. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad útil. (obligatorio).

$Ls_id_util: (string) es el nombre o id de la entidad útil hija de una clase ya ingresada en $this->laEntidades, este parámetro puede ser vacío, de ser vacío se entiende que se busca la existencia de una clase(\namespace\clase) ya ingresada en $this->laEntidades, clase base para entidades útiles instancias(objeto) o clase (clase de procedimientos estáticos).

Si $Ls_id_util tiene valor y existe como entidad útil hija de la \namespace\clase base en la estructura de datos de $this->laEntidades entonces este procedimiento retorna el tipo_entidad (‘objeto’ o ‘clase’) de la clase base. Si no existe como entidad útil hija el procedimiento retorna null.

Si $Ls_id_util no tiene valor, o tiene valor vacío, existe la entidad útil \namespace\clase base en la estructura de datos de $this->laEntidades entonces este procedimiento retorna el tipo_entidad (‘objeto’ o ‘clase’) de la clase base. Si no existe como la entidad útil \namespace\clase base el procedimiento retorna null.

public static function ingresarEntidad ($Ls_namespace , $Ls_clase , $Ls_id_util , $La_parametros_iniciacion = null)
es el procedimiento encargado de gestionar el ingreso de entidades útiles hija de la \namespace\clase base en la estructura de datos de $this->laEntidades. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad útil, es obligatorio que exista ya en $this->laEntidades. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad útil, es obligatorio que exista ya en $this->laEntidades. (obligatorio).

$Ls_id_util: (string) es el nombre o id de una entidad útil a ingresar en la estructura instancias de la clase base \namespace\clase en la estructura de datos de $this->laEntidades(obligatorio).

$La_parametros_iniciacion: (array) cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad útil ($Ls_id_util), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad útil a gestionar sea de tipo clase (tipo_entidad = 'clase') no tiene efecto este parámetro ya que se gestionó en el primer elemento $Ls_id_util que se ingreso en la clase base. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

public static function entidad ($Ls_namespace , $Ls_clase , $Ls_id_util , $Ls_tipo_referencia = 'referencia')
es el procedimiento encargado de gestionar la entidad útil hija de la \namespace\clase base en la estructura de datos de $this->laEntidades identificada por $Ls_id_util, es decir si la entidad es de tipo ‘objeto’ retronará la referencia a un objeto instancia de la entidad útil, si la entidad es de tipo clase retornará un string compuesto por el namespace y nombre de la clase base. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad útil. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad útil. (obligatorio).

$Ls_id_util: (string) es el nombre o id de la entidad útil hija de una clase ya ingresada en $this->laEntidades, el string en $Ls_id_util tiene obligatoriamente que existir como elemento en la estructura instancias de la clase base en $this->laEntidades . (obligatorio).

$Ls_tipo_referencia: (string) es el tipo de referencia a objeto que devuelve el procedimiento en caso de que el tipo de entidad sea ‘objeto‘, este parámetro tiene dos valores posibles ‘referencia‘ y ‘clon‘, referencia devuelve un puntero a la entidad guardada en el arreglo $this->laEntidades, y clon devuelve un clon de este objeto. Si el tipo de entidad es ‘clase‘ no se considera este parámetro. Por defecto este parámetro tiene como valor ‘referencia‘.

Este procedimiento devuelve (retorna) una referencia o clon a objeto si el tipo_entidad de la clase base es (‘objeto’); devuelve (retorna) un string de la forma \namespace\clase si el tipo_entidad de la clase base es (‘clase’); de lo contrario retorna null.

La instanciación de los objetos ocurre en el momento que es pedido por primera vez una instancia a través de este procedimiento.

En el caso de la instanciación o iniciación de entidades de tipo ‘objeto’, e iniciación de entidades de tipo ‘clase’, estas entidades pueden emitir una excepción desde su constructor o procedimiento de iniciación si existe algún problema de instanciación o iniciación, ya que este procedimiento tiene

un bloque try y catch para manejar las excepciones, el bloque catch espera un objeto de la clase \Exception y emite el mensaje de este, el código es de la siguiente forma:

```
catch( \Exception $e)
  {
     echo '<p>Excepción capturada: '.$e->getMessage(). "<p>" ;
     return null;
   }
```

Por lo que el lanzamiento de la excepción sería:

```
throw new \Exception('mensaje.');
```

Esta funcionalidad es muy útil para interrumpir la instanciación de una clase si existe algún error en esta instanciación, y así no se tienen objetos con problemas o sin uso por desperfectos, en este contenedor de instancias.

Lo anterior se aplica a entidades de tipo ‘objeto’ que lancen su excepción desde el constructor, o entidades de tipo ‘clase’ que lancen su excepción desde el procedimiento de iniciación; Quedan excluidos las entidades de tipo ‘objeto’ que lancen su excepción desde un procedimiento de iniciación(no __construct) ya que estos ya fueron instanciados existen como instancias en el contenedor, estos casos solo tienen la opción de emitir un mensaje.

public static function datosEntidad ($Ls_namespace , $Ls_clase , $Ls_id_util)
es el procedimiento encargado de gestionar y retornar los datos, que de existir, tiene una entidad útil que haya ingresado en esta clase, estos datos son los correspondientes al elemento ‘datos’ que puede traer consigo una entidad útil en el momento de su ingreso al contenedor $laEntidades de esta clase. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad útil. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad útil. (obligatorio).

$Ls_id_util: (string) es el nombre o id de la entidad útil hija de una clase ya ingresada en $this->laEntidades, el string en $Ls_id_util tiene obligatoriamente que existir como elemento en la estructura instancias de la clase base en $this->laEntidades . (obligatorio).

este procedimiento retorna el arreglo ‘datos’ perteneciente a la entidad útil propietaria si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

public static function idEntidades ($Ls_namespace , $Ls_clase)
es el procedimiento encargado de gestionar y retornar los nombres o identificadores de entidades útiles que hayan ingresado en el namespace y clase correspondientes a los parámetros de este

procedimiento, estos datos son los correspondientes al elemento ‘instancias’ que debe tener consigo cada clase base útil en el contenedor $laEntidades de esta clase. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad útil. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad útil. (obligatorio).

si la gestión fue llevada a cabo con éxito este procedimiento retorna las llaves del array ‘instancias’ contenidos en el namespace y clase, clase base útil en el contenedor $laEntidades de esta clase, de lo contrario retorna null.

public static function eliminarEntidad ($Ls_namespace , $Ls_clase , $Ls_id_util = null)
es el procedimiento encargado de eliminar una entidad útil clase base, o una entidad útil de la estructura ‘instancias’ de una clase base, existentes ambas en el contenedor $laEntidades de esta clase. Este procedimiento está sujeto al valor de la propiedad self::$lbPropietarioEntidad perteneciente a la sección de seguridad de esta clase Utiles, si esta propiedad tiene valor true solo puede eliminar la entidad útil el proceso que figura como su propietario, de lo contrario puede eliminarse desde cualquier proceso, el proceso núcleo siempre tiene derechos de eliminar cualquier entidad útil.

Existen dos posibilidades de uso de este procedimiento:

La primera es eliminar una entidad útil clase base, es decir una namespace\clase registrada como clase base de entidades útiles, para ello se pasan los valores correspondientes al namespace y clase de la entidad a eliminar, dejando vacío el parámetro $Ls_id_util, recordar que esta operación está sujeto a la sección de seguridad de esta clase, donde se define quién puede eliminar una entidad determinada.

La segunda es eliminar una entidad útil de la estructura ‘instancias’ de una clase base, para ello debe existir la entidad útil en la estructura $laEntidades[$Ls_namespace][$Ls_clase][‘instancias’] de esta clase, el parámetro $Ls_id_util no puede ser vacío, recordar que esta operación está sujeto a la sección de seguridad de esta clase, donde se define quien puede eliminar una entidad determinada.

Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad útil. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad útil. (obligatorio).

$Ls_id_util: (string) es el nombre o id de la entidad útil hija de una clase ya ingresada en $this->laEntidades, el string en $Ls_id_util tiene obligatoriamente que existir como elemento en la estructura instancias de la clase base en $this->laEntidades . (obligatorio).

este procedimiento retorna true si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

clase_e_errores

Este fichero contiene la clase de procedimientos estáticos Errores. Esta clase está compuesta por propiedades, y procedimientos estáticos para la gestión de entidades Errores, desde la inclusión del script o clase como fichero hasta el retorno de referencias o clonaciones a entidades clases u objetos, gestiona y sirve como contenedor de estas entidades, una opción clara para la inyección de dependencias. En su gestión las entidades Errores no son incluidas o instanciadas hasta el momento que se hace la petición de la instancia para su uso. Para un mejor entendimiento de cómo se gestionan las entidades Errores se debe consultar cada propiedad y procedimiento de esta clase. Debemos aclarar que esta entidad no gestiona un error en específico sino las entidades (clases u objetos) que serán las que si gestionen los errores en específico.

A continuación mostramos una lista de las propiedades y procedimientos que esta clase pone a disposición de los códigos clientes del Sockeletom:

Namespace: Emod\Nucleo
Nombre de la clase: Errores
Herencia: use \Emod\Nucleo\Herramientas\GECO

Propiedades:

private static $EEoNucleo = null ;
puntero al objeto EEoNucleo ;

private static $lbGestionSeguridad = false ;
es un boolean perteneciente a la sección de seguridad de esta clase Errores, que define si se gestionaran los errores bajo condiciones de seguridad, true se gestiona con seguridad, false no se gestiona con seguridad, estas condiciones de seguridad estarán declaradas en esta clase y son gestionadas por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen elementos de seguridad en el parámetro de datos entrante .

private static $lbPropietarioEntidad = true ;
es un boolean perteneciente a la sección de seguridad de esta clase Errores, que define si al ingresar entidades errores en esta clase, se registra el proceso que ingresa la entidad errores como su propietario, esto tiene como objetivo que la opción de eliminar entidades errores sea posible solo

por su propietario o por cualquier proceso que decida eliminar una entidad errores del ámbito de gestión de esta clase, true se registra el proceso propietario, false no se registra el proceso propietario, este elemento de seguridad es gestionado por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen parámetros de seguridad en el parámetro de datos entrante. No está afectado por la propiedad $lbGestionSeguridad ya que aunque este no esté activado, con valor true, si $lbPropietarioEntidad si está activado, con valor true, se lleva a cavo su función y objetivo. El proceso núcleo siempre tiene permiso de eliminar una entidad errores determinada.

private static $lsAmbitoSeguridad = null ;
es un string perteneciente a la sección de seguridad de esta clase Errores, que define el ámbito de seguridad a aplicar a la lista de entidades errores declaradas a permitir o denegar su gestión por parte de esta clase Errores, sus valores posibles son “permisivo” y ” restrictivo” , está directamente relacionada con el elemento de seguridad $laDatosSeguridad que se expondrá más adelante, el valor “permisivo” permite la gestión de cualquier entidad errores excepto las declaradas en $laDatosSeguridad, el valor “restrictivo” permite la gestión solo de las entidades errores declaradas en $laDatosSeguridad y deniega las demás entidades errores, este elemento de seguridad es gestionado por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen parámetros de seguridad en el parámetro de datos entrante. Esta propiedad está afectada por la propiedad $lbGestionSeguridad, si no está activada la propiedad $lbGestionSeguridad, con valor true, se omite la existencia y valor de esta propiedad $lsAmbitoSeguridad.

private static $lbActualizarDatosSeguridad = null ;
es un boolean perteneciente a la sección de seguridad de esta clase Errores, que define la posibilidad o no de que los procesos puedan actualizar la lista de entidades errores declaradas a permitir o denegar su gestión por parte de esta clase Errores. sus valores posibles son “true” y ” false” , está directamente relacionada con el elemento de seguridad $laDatosSeguridad que se expondrá más adelante, el valor “true” permite que cualquier proceso actualice la lista de entidades errores declaradas a permitir o denegar su gestión por parte de esta clase Errores, false no permite lo anterior, el proceso núcleo si tiene permisos para actualizar la lista antes mencionada independientemente del valor de esta propiedad , este elemento de seguridad es gestionado por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen parámetros de seguridad en el parámetro de datos entrante. Esta propiedad está afectada por la propiedad $lbGestionSeguridad, si no está activada la propiedad $lbGestionSeguridad, con valor true, se omite la existencia y valor de esta propiedad $lbActualizarDatosSeguridad.

private static $laDatosSeguridad = array() ;
es un array perteneciente a la sección de seguridad de esta clase Errores, que define la lista de entidades errores declaradas a permitir o denegar su gestión por parte de esta clase Errores, está directamente relacionada con las propiedades $lsAmbitoSeguridad y $lbActualizarDatosSeguridad de esta clase Errores. Este elemento de seguridad es a nivel de clase y no de entidades errores específicas, se da permiso o no a la clase no a una entidad correspondiente a la clase. Este elemento

de seguridad es gestionado por el procedimiento iniciación de esta clase, es en este procedimiento donde se chequea si existen parámetros de seguridad en el parámetro de datos entrante. Esta propiedad está afectada por la propiedad $lbGestionSeguridad, si no está activada la propiedad $lbGestionSeguridad, con valor true, se omite la existencia y valor de esta propiedad $laDatosSeguridad. Existe la posibilidad de poner el valor * (asterisco) que significa todos los elementos posibles, se puede hacer a nivel de $laDatosSeguridad o a nivel de un namespace. A continuación se muestra la estructura de este arreglo:

$laDatosSeguridad:
namespace_entidades_errores1:
clase_entidades_errores1
clase_entidades_errores2
clase_entidades_erroresN
namespace_entidades_errores2: *
namespace_entidades_erroresN:
clase_entidades_errores1
clase_entidades_errores2
clase_entidades_erroresN

también $laDatosSeguridad puede tener el valor *

$laDatosSeguridad: *

namespace_entidades_errores (string) es el nombre del namespace al que pertenecen las entidades errores. (obligatorio)

clase_entidades_errores (string) es el nombre de la clase a la que pertenecen las entidades errores. (obligatorio)

$lsPathDirRaizEntidades = null ;
propiedad para contener el valor del path del directorio base donde se alojarán las entidades errores en el sistema Sockeletom, es el directorio que el sistema propone como contenedor de las entidades errores que se utilicen por el núcleo del sistema y los demás códigos clientes del Sockeletom. Este parámetro tiene a su vez como path raíz al directorio esquelemod, perteneciente a la estructura de directorios del sistema Sockeletom. Tomando como ejemplo el valor ¨ errores ¨ para esta propiedad, el valor absoluto del path donde se alojaran las entidades errores sería:
raíz_sistema_Sockeletom/esquelemod/errores

private static $iniciacion = null ;
propiedad para chequear la iniciación de esta clase

private static $laEntidades = array() ;

es un arreglo que contendrá las entidades Errores y sus características operacionales, entre estas características están:

namespace: (string) es el nombre del namespace al que pertenece la entidad errores (obligatorio)

clase: (string) es el nombre de la clase a la que pertenece la entidad errores (obligatorio)

'path_entidad_clase': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece la entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en esta clase la entidad errores a gestionar, si el fichero no existe no se ingresa la entidad errores en esta propiedad $laEntidades de esta clase. (obligatorio)

'tipo_entidad': (string) contiene el tipo de entidad, es decir si es de tipo clase u objeto la entidad errores a gestionar, sus posibles valores son ‘clase’ o ‘objeto. (obligatorio)

'iniciacion': (string) es la forma como se iniciará una entidad errores, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades errores al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades errores le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades errores le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad errores sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad errores del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento ['instancias'][id_errores] ['parámetros_iniciacion'] de la clase base, por lo que cada entidad errores puede tener sus valores propios, excepto en caso de que la entidad errores sea de tipo clase (tipo_entidad = 'clase') que con el primer id_errores se inicia la clase base. (opcional)

'proceso_propietario': (string) es el proceso que hace el ingreso de la clase entidad errores, este valor el procedimiento lo toma del objeto EEoNucleo conformado por el GEDEE del proceso en ejecución, seguido de :: y el id del proceso en ejecución (self::$EEoNucleo->gedeeProcesoEjecucion . '::' . self::$EEoNucleo->idProcesoEjecucion()) y es para definir el único proceso que puede eliminar la entidad namespace/clase de este arreglo, este elemento está sujeto a la propiedad $lbPropietarioEntidad de la sección de seguridad de esta clase Errores. Cuando el proceso propietario de la entidad namespace/clase decide eliminar esta entidad, se eliminará junto con todas las instancias contenidas de esta namespace/clase.

'instancias': (array) contiene las instancias que se creen de la clase correspondiente, contiene las instancias y los datos correspondientes a cada instancia, entre estos datos están:

id_entidad_errores: (string) es el nombre o identificador de la entidad errores (obligatorio), los id_entidad_errores pueden ser ingresados en el momento que se ingresa el namespace/clase de la clase entidad_errores o posteriormente haciendo una petición de ingreso de instancia de una entidad_errores a través del procedimiento * ingresarEntidad * de esta clase.

'registro': (boolean) en caso de 'tipo_entidad' tener valor 'clase' entonces esta propiedad contiene el valor true si ha sido solicitada al menos una vez.

'objeto': (&obj) en caso de 'tipo_entidad' tener valor 'objeto' entonces esta propiedad contiene la referencia al objeto, entidad errores a gestionar.

'proceso_propietario': (string) es el proceso que hace el ingreso de la entidad errores, este valor el procedimiento lo toma del objeto EEoNucleo conformado por el GEDEE del proceso en ejecución, seguido de :: y el id del proceso en ejecución (self::$EEoNucleo->gedeeProcesoEjecucion . '::' . self::$EEoNucleo->idProcesoEjecucion()) y es para definir el único proceso que puede eliminar la entidad instancia de este arreglo, este elemento está sujeto a la propiedad $lbPropietarioEntidad de la sección de seguridad de esta clase Errores. Se aplica solo a la entidad errores específica, no tiene que ser el mismo propietario que el de la entidad namespace/clase de esta entidad errores.

'datos': es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad errores en esta clase Errores; estos datos se crean en el momento del ingreso de la entidad errores sin que permita cambios posteriores.

la estructura del arreglo quedaría así:

```
$laEntidades = array(
                    'nombre namespace' => array(
                                        'nombre clase' => array(
                                                            'path_entidad_clase' => 'path_entidad_clase',
                                                            'tipo_entidad' => 'tipo_entidad',
                                                            ‘iniciacion’ => ‘procedimiento’
                                                            'proceso_propietario' => 'proceso_que_ingresa_la_entidad',
                                                            'instancias' => array(
                                                                        'id_entidad_error1' => array(
                                                                                    'registro' => 'true o false,
                                                                                    'objeto' => &objeto,
                                                                                    'proceso_propietario' => 'proceso_que_ingresa_la_entidad',
                                                                                    'datos' => array( 'dato1' => 'valor_dato' , 'datoN' =>datoN')
                                                                                                ) ,
                                                                        'id_entidad_erroresN' => array(
                                                                                    'registro' => 'true o false,
                                                                                    'objeto' => &objeto,
                                                                                    'proceso_propietario' => 'proceso_que_ingresa_la_entidad',
                                                                                    'datos' => array( 'dato1' => 'valor_dato' , 'datoN' =>datoN')
                                                                                                )
                                                                                    )
                                                                                )
                                                                            )
                                                                        )
                                                                    )
```

private static $laPilaGestoresErrores = null ;
es un arreglo con función de pila (estructura LIFO) que contendrá las entidades gestores de errores devueltos por la función set_error_handler al implantar nuevas entidades gestores de errores, para que esta pila siempre esté actualizada y realice su función correctamente, si distorsiones, es necesario que el proceso de implantación de nuevos gestores de errores se haga a través de los procedimientos que para ello tiene esta clase errores, de lo contrario esta pila no será una imagen fiel de la realidad del sistema de tratamiento de errores.

La pila tiene la siguiente estructura[índice auto numérico][array(self::$laGestorErroresEjecucion , self::$lsIdGestorErroresEjecucion)]

private static $lsIdGestorErroresEjecucion = 'PHP Nativo';
es un string que contendrá un identificador para el gestor de errores en ejecución, actual(de turno), de modo que se pueda identificar de forma rápida y muy simple cada gestor de errores.

private static $laGestorErroresEjecucion = array( );
es un array que contendrá en el elemento 0 un objeto y en el elemento 1 un string con el nombre del procedimiento perteneciente al objeto (elemento 0) que será el gestor de errores en ejecución, actual(de turno).

Procedimientos:

public static function iniciacion($La_configuracion_nucleo_errores)
es el procedimiento encargado de iniciar el funcionamiento de esta clase y además guardar en las propiedades de esta clase los elementos existentes como valores de la sección errores propuestas por el sistema Sockeletom en la configuración del sistema Sockeletom. Además realiza el ingreso de las entidades errores propuestas a gestionar en la configuración del sistema Sockeletom. Para un mejor entendimiento de los parámetros que la configuración del sistema proporciona a este procedimiento, debe consultarse la sección errores del fichero de configuración del sistema Sockeletom en este documento.

este procedimiento retorna null si ya estaba inicializada la clase Errores , retorna true si la gestión es exitosa, detiene el proceso php si los parámetros son incompatibles con el procedimiento, emitiendo un error fatal (exit).

public static function pathRaizEntidades ()
es el procedimiento encargado de brindar, retornar el path del directorio raíz para los recursos de errores en el sistema Sockeletom. Este será el fragmento de path que completará el path del directorio base de los recursos de errores, del directorio que el sistema propone como contenedor de los recursos de errores que se utilicen por el núcleo del sistema y los demás códigos clientes del Sockeletom. Este parámetro tiene además como path raíz al directorio esquelemod, pertenecientes a

la estructura de directorios del sistema Sockeletom. Tomando como ejemplo el valor ¨ errores ¨, quedará como resultado el path:
raíz_sistema_Sockeletom/esquelemod/errores

este procedimiento retorna el path gestionado o null en caso de no existir el path a gestionar.

public static function actualizarDatosSeguridadEntidades ($La_datos , $Ls_tipo_actualizacion)
es el procedimiento encargado de gestionar la actualización de la propiedad self::$laDatosSeguridad de esta clase Errores, la gestión de actualización está sujeta al valor de la propiedad self::$lbActualizarDatosSeguridad perteneciente a la sección de seguridad de esta clase Errores. Los parámetros se explican a continuación:

$La_datos: (array) es el arreglo con los datos a actualizar.

$Ls_tipo_actualizacion: (string) es el tipo de actualización a realizar, sus posibles valores son:

'renovar': es la sustitución del arreglo $La_datos por el arreglo actual en self::$laDatosSeguridad.

'adicionar': es la adición del contenido del arreglo $La_datos en el arreglo actual self::$laDatosSeguridad.

'eliminar': es la eliminación del contenido del arreglo $La_datos en el arreglo actual self::$laDatosSeguridad.

este procedimiento retorna true si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

public static function permisionEntidad ($Ls_namespace , $Ls_clase)
es el procedimiento encargado de gestionar si una clase entidad error determinada tiene permiso de ser gestionado por esta clase Errores, el permiso es a nivel de clase y no de entidades específicas de esta, se da permiso o no a la clase no a una entidad correspondiente a la clase; la gestión de chequeo de permisos está sujeto al valor de las propiedades self::$lbGestionSeguridad, self::$lsAmbitoSeguridad y self::$laDatosSeguridad pertenecientes a la sección de seguridad de esta clase Errores; específicamente chequea las clases permitidas o denegadas en self::$laDatosSeguridad según el self::$lsAmbitoSeguridad declarado. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la clase entidad error a chequear su permiso de gestión (obligatorio)

$Ls_clase: (string) es el nombre de la clase a la que pertenece la clase entidad error a chequear su permiso de gestión (obligatorio)

este procedimiento retorna true si la clase entidad error tiene permiso a ser gestionada por esta clase, de lo contrario retorna null.

public static function gestionControlIngresoEntidad ($Ls_path_control_errores , $Ls_referencia_path_control = 'relativo' , $Ls_namespace = null , $Ls_clase = null , $Ls_path_entidad_clase = null , $Ls_referencia_path_entidad = null , $Ls_tipo_entidad = null , $La_instancias = null , $Ls_iniciacion = null)
es el procedimiento encargado de gestionar el ingreso de una entidad errores al contenedor $laEntidades de esta clase, utilizando entre sus parámetros un parámetro que hace referencia al path de un fichero control.

Se llama fichero control a un fichero o script php que se necesita su ejecución como parte de la gestión de ingreso de la entidad errores a esta clase Errores. El control puede utilizarse para chequear, gestionar o poner a punto algún elemento necesario para el ingreso de la entidad errores a esta clase.

El fichero control es ejecutado (require_once) por este procedimiento y puede utilizarse en conjunto con los demás parámetros de este procedimiento o gestionar el mismo la inclusión (require_once) de la clase entidad errores y retornar el mismo los parámetros que necesita este procedimiento para la gestión del ingreso de la entidad errores a esta clase.

Las diferentes combinaciones que pueden utilizarse devienen diferentes funcionalidades que son:

Funcionalidad 1 donde el control debe ejecutarse y las características de la entidad se pasan como parámetros independientes del control. Si se declaran parámetros para el control y a la vez se declaran los parámetros
$Ls_namespace, $Ls_clase, $Ls_path_entidad_clase, $Ls_referencia_path_entidad, $Ls_tipo_entidad , $La_instancias = null , $Ls_iniciacion este procedimiento incluirá el fichero control y además utilizará los parámetros antes mencionados para la gestión de ingreso de la entidad errores , no importa que el control retorne también los parámetros antes mencionados porque este procedimiento da prioridad a los valores de sus parámetros y nunca evalúa los parámetros homólogos retornados por el control. Tienen que existir todos los parámetros antes mencionados para que esta opción se ejecute, si solo uno de estos parámetros es vacío entonces se pasa a las siguientes funcionalidades.

antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de la entidad errores, si no existe el fichero el procedimiento no ingresa la entidad errores y retorna null. Ingresar una entidad errores no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de esta entidad errores. Los parámetros se explican a continuación:

$Ls_path_control_errores : (string) es el path del fichero control a ejecutar para el ingreso de la entidad errores. (obligatorio)

$Ls_referencia_path_control: (string) es la forma que se hace referencia a $Ls_path_control_errores, tiene tres posibles valores, *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el $Ls_path_control_errores relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el $Ls_path_control_errores relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el $Ls_path_control_errores de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad errores.

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad errores.

$Ls_path_entidad_clase: (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, si el fichero no existe no se ingresa la entidad errores al arreglo.

$Ls_referencia_path_entidad: (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el $Ls_path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el $Ls_path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el $Ls_path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

$Ls_tipo_entidad: (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

$Ls_iniciacion: (string) es la forma como se iniciará cada id_entidad_errores, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de $Ls_iniciacion son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades errores al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades errores le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades errores le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad errores sea de tipo clase ($Ls_tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se

ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad errores del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento [$La_instancias][id_entidad_errores]['parametros_iniciacion'] de la clase base, por lo que cada entidad errores puede tener sus valores propios, excepto en caso de que la entidad errores sea de tipo clase ($Ls_tipo_entidad = 'clase') que con el primer id_ entidad_errores se inicia la clase base. Los elementos [$La_instancias][id_entidad_errores]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

$La_instancias (array) es un arreglo con los identificadores de entidades errores a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_errores. (obligatorio)

id_entidad_errores (string) es un nombre o identificador de cada entidad errores instancia de la clase correspondiente. y a su vez este id_entidad_errores es un arreglo asociativo al que se puede insertar una llave de nombre parametros_iniciacion y otra de nombre datos, posteriormente este arreglo id_entidad_errores se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_errores como sea necesario ya que la entidad errores puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad errores (id_entidad_errores), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad errores a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad_errores declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad errores es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad errores en esta clase Errores; estos datos se crean en el momento del ingreso de la entidad errores sin que permita cambios posteriores. (opcional)

Para el resto de las funcionalidades. Es necesario volver a aclarar que si se declaran parámetros en este procedimiento para el fichero control y al menos uno de los restantes es vacío entonces este procedimiento ejecuta (require_once) el control, desecha los demás parámetros pasados a este

procedimiento y espera en forma de retorno(return) desde el fichero control requerido(require_once), un arreglo asociativo con las posibles siguientes estructuras:

Funcionalidad 2 donde el control debe ejecutarse, y este mismo ya incluyó la clase y nos retorna las características de la entidad incluido la o las instancias(id_entidad_errores)(objeto) de la clase, para que se ejecute esta funcionalidad esta estructura tiene que estar completa, si faltase algún elemento no se ejecutará esta funcionalidad, también tiene como condición que el elemento $La_resultado_require['referencia_path_entidad'] no exista, la responsabilidad de incluir la estructura correcta de los id_entidad_errores instanciadas (objeto) o registradas(clase) es responsabilidad del control, también es obligatorio que exista al menos una id_entidad_errores en la estructura de datos que se espera por retorno. La estructura que se espera por retorno es la siguiente:

$La_resultado_require['namespace'] (obligatorio)
$La_resultado_require['clase'] (obligatorio)
$La_resultado_require['path_entidad_clase'] (obligatorio)
$La_resultado_require['tipo_entidad'] (obligatorio)
$La_resultado_require['iniciacion'] (opcional)
$La_resultado_require['instancias'] (obligatorio)
$La_resultado_require['instancias'] ['id_entidad_errores '] (obligatorio)
$La_resultado_require['instancias'] ['id_entidad_errores '] ['objeto'] (opcional)
$La_resultado_require['instancias'] ['id_entidad_errores'] ['registro'] (opcional)
$La_resultado_require['instancias'] ['id_entidad_errores'] [' datos'] (opcional)

'namespace': (string) es el nombre del namespace al que pertenece la entidad errores. (obligatorio)

'clase': (string) es el nombre de la clase a la que pertenece la entidad errores. (obligatorio)

'path_entidad_clase': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad errores al arreglo $laEntidades de esta clase. (obligatorio)

'tipo_entidad': (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

'iniciacion': (string) es la forma como se iniciará cada id_entidad_errores , este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades errores al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades errores le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento

cualquiera de la instancia al que el gestor de entidades errores le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad errores sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad errores del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento ['instancias'][id_entidad_errores]['parametros_iniciacion'] de la clase base, por lo que cada entidad errores puede tener sus valores propios, excepto en caso de que la entidad errores sea de tipo clase (tipo_entidad = 'clase') que con el primer id_ entidad_errores se inicia la clase base. Los elementos ['instancias'][id_entidad_errores]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

'instancias' (array) es un arreglo con los identificadores de entidades errores a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_errores. (obligatorio)

id_entidad_errores (string) es un nombre o identificador de cada entidad errores instancia de la clase correspondiente. y a su vez este id_entidad_errores es un arreglo asociativo al que se puede insertar una llave de nombre objeto, una llave de nombre registro, y otra de nombre datos, las llaves objeto, y registro no deben coexistir ya que la existencia de una determina la no existencia de la otra ,posteriormente este arreglo id_entidad_errores se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_errores como sea necesario ya que la entidad errores puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'objeto': (&obj) es un apuntador al objeto instancia de la clase entidad errores a contener en el contenedor $laEntidades de esta clase. Este elemento es obligatorio si el elemento tipo_entidad tiene como valor el string objeto y estrictamente se considera iniciado.

'registro': (boolean true) es para definir cuando una entidad errores a contener en el contenedor $laEntidades de esta clase, y de tipo_entidad clase (procedimientos estáticos) es iniciada . Este elemento es obligatorio si el elemento tipo_entidad tiene como valor el string clase y estrictamente se considera iniciada.

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad errores en esta clase Errores; estos datos se crean en el momento del ingreso de la entidad errores sin que permita cambios posteriores. (opcional)

Funcionalidad 3 donde el control debe ejecutarse, y este nos retorna(return) los elementos o parámetros para ingresar una clase, y reserva al menos un id_entidad_errores para su posterior instanciación, no debe tener id_entidades_errores instanciadas(objeto) o registradas(clase), y es

obligatorio que exista en los parámetros retornados por el control el elemento $La_resultado_require['referencia_path_entidad']. Entiéndase que esta funcionalidad solo realiza su gestión si no existe ningún id_entidad_errores instanciado este factor es obligatorio. La estructura que se espera por retorno es la siguiente:

$La_resultado_require['namespace'] (obligatorio)
$La_resultado_require['clase'] (obligatorio)
$La_resultado_require['path_entidad_clase'] (obligatorio)
$La_resultado_require['referencia_path_entidad '] (obligatorio)
$La_resultado_require['tipo_entidad'] (obligatorio)
$La_resultado_require['iniciacion'] (opcional)
$La_resultado_require['instancias'] (obligatorio)
$La_resultado_require['instancias'] [id_entidad_errores] (obligatorio)
$La_resultado_require['instancias'] id_entidad_errores] ['parametros_iniciacion'] (opcional)
$La_resultado_require['instancias'] [id_entidad_errores] [' datos'] (opcional)

'namespace': (string) es el nombre del namespace al que pertenece la entidad errores. (obligatorio)

'clase': (string) es el nombre de la clase a la que pertenece la entidad errores. (obligatorio)

'path_entidad_clase': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad errores al arreglo $laEntidades de esta clase. (obligatorio)

'referencia_path_entidad': (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

'tipo_entidad': (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

'iniciacion': (string) es la forma como se iniciará cada id_entidad_errores, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades errores al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades errores le pasa los

parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades errores le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad errores sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad errores del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento ['instancias'][id_entidad_errores]['parametros_iniciacion'] de la clase base, por lo que cada entidad errores puede tener sus valores propios, excepto en caso de que la entidad errores sea de tipo clase (tipo_entidad = 'clase') que con el primer id_ entidad_errores se inicia la clase base. Los elementos ['instancias'][id_entidad_errores]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

'instancias' (array) es un arreglo con los identificadores de entidades errores a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_errores. (obligatorio)

id_entidad_errores (string) es un nombre o identificador de cada entidad errores instancia de la clase correspondiente. y a su vez este id_entidad_errores es un arreglo asociativo al que se puede insertar una llave de nombre parametros_iniciacion y otra de nombre datos, posteriormente este arreglo id_entidad_errores se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_errores como sea necesario ya que la entidad errores puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad errores (id_entidad_errores), si existe este parámetro, es arreglo y no es vacio entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad errores a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad_errores declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad errores es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad errores en esta clase Errores; estos datos se crean en el momento del ingreso de la entidad errores sin que permita cambios posteriores. (opcional)

La funcionalidad que el control ejecutado devuelva los datos de la clase y sus elementos['instancias'][id_entidad_errores] no instanciados, es por seguridad ya que existen otras formas para crear controles que retornen estructuras con sus id_entidad_errores ya instanciados(objeto) o registrados(clase); Esta característica está abierta a cambios futuro en dependencia de la opinión mayoritaria de los usuarios del sistema Sockeletom.

antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de la entidad errores, si no existe el fichero el procedimiento no ingresa la entidad errores y retorna null. Ingresar una entidad errores no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de esta entidad errores.

este procedimiento retorna true si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

public static function gestionIngresoEntidad ($Ls_namespace , $Ls_clase , $Ls_path_entidad_clase , $Ls_referencia_path_entidad , $Ls_tipo_entidad , $La_instancias , $Ls_iniciacion = null)
es el procedimiento encargado de gestionar el ingreso de una entidad errores al contenedor $laEntidades de esta clase, antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de la entidad errores, si no existe el fichero el procedimiento no ingresa la entidad errores y retorna null. Ingresar una entidad errores no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de esta entidad errores. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad errores. (obligatorio)

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad errores. (obligatorio)

$Ls_path_entidad_clase: (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad errores al arreglo $laEntidades de esta clase. (obligatorio)

$Ls_referencia_path_entidad: (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio)

$Ls_tipo_entidad: (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio)

$Ls_iniciacion: (string) es la forma como se iniciará cada id_entidad_errores, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades errores al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades errores le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades errores le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad errores sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad errores del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento $La_instancias[id_entidad_errores]['parametros_iniciacion'] de la clase base, por lo que cada entidad errores puede tener sus valores propios, excepto en caso de que la entidad errores sea de tipo clase (tipo_entidad = 'clase') que con el primer id_ entidad_errores se inicia la clase base. Los elementos $La_instancias[id_entidad_errores]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional)

$La_instancias (array) es un arreglo con los identificadores de entidades errores a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_errores. (obligatorio)

id_entidad_errores (string) es un nombre o identificador de cada entidad errores instancia de la clase correspondiente. y a su vez este id_entidad_errores es un arreglo asociativo al que se puede insertar una llave de nombre parametros_iniciacion y otra de nombre datos, posteriormente este arreglo id_entidad_errores se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_errores como sea necesario ya que la entidad errores puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1) (opcional ++)

'parametros_iniciacion': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad errores (id_entidad_errores), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad errores a gestionar sea de tipo clase

($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad_errores declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad errores es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad errores en esta clase Errores; estos datos se crean en el momento del ingreso de la entidad errores sin que permita cambios posteriores. (opcional)

public static function gestionIngresosEntidades ($La_propiedades_errores)
es el procedimiento encargado de gestionar el ingreso de varias entidades errores al contenedor $laEntidades de esta clase, este procedimiento desglosa un arreglo con datos de varias entidades errores y realiza la gestión de ingreso según sus características como lo hacen individualmente los procedimientos self::gestionControlIngresoEntidad y self::gestionIngresoEntidad, es importante aclarar que cada entidad errores será procesada estrictamente con los procedimientos self::gestionControlIngresoEntidad o self::gestionIngresoEntidad por lo que los datos de las entidades errores tienen que cumplir con las condicionantes del procedimiento respectivo para que la gestión sea exitosa ,antes de ingresar los datos, el procedimiento chequea que exista el fichero clase de cada entidad errores, si no existe el fichero el procedimiento no ingresa la entidad errores y continua su gestión. Ingresar una entidad errores no implica en el instante del ingreso la instanciación de la clase, las entidades que sean de tipo objeto son instanciadas en el momento del primer uso o pedido de esta entidad errores. Los parámetros se explican a continuación:

$La_propiedades_errores: es un arreglo en el que los elementos llave de este arreglo son los namespaces de las entidades errores a gestionar su ingreso en esta clase, la llave namespace representa a su vez un arreglo con las clases y algunas características de estas clases que finalmente son las entidades errores declaradas, la llave clase representa a su vez un arreglo con algunas características necesarias para la correcta gestión de las entidades errores, un ejemplo de esta estructura se expone a continuación:

$La_propiedades_errores
namespace_entidad errores1:
clase_entidad errores1:
path_entidad_clase:
referencia_path_entidad:
path_control:
referencia_path_control:
tipo_entidad:
iniciación:
instancias:
id_entidad errores1:

```
                                                parametros_iniciacion
                                                datos:
                                        id_entidad errores2
                                                parametros_iniciacion
                                        id_entidad errores3
                                        id_entidad errores4:
                                                datos:

        namespace_entidad errores2:
                clase_entidad errores1:
                        path_entidad_clase:
                        referencia_path_entidad:
                        path_control:
                        referencia_path_control:
                        tipo_entidad:
                        instancias:
                                id_entidad errores1:
                                        datos:
                                id_entidad errores2
                                id_entidad errores3
                                id_entidad errores4
                clase_entidad errores2:
                        path_entidad_clase:
                        referencia_path_entidad:
                        path_control:
                        referencia_path_control:
                        tipo_entidad:
                        iniciación:
                        instancias:
                                id_entidad errores1:
                                        datos:
                                id_entidad errores2:
                                        parametros_iniciacion
                                        datos:
                                id_entidad errores3:
                                        datos:
                                id_entidad errores4:
                                        datos:
```

los elementos de este arreglo se explican a continuación:

namespace: (string) es el nombre del namespace al que pertenece la entidad errores. (obligatorio en dependencia del procedimiento gestor)

clase: (string) es el nombre de la clase a la que pertenece la entidad errores. (obligatorio en dependencia del procedimiento gestor)

' path_entidad_clase ': (string) contiene el path del fichero que alberga el código de la clase a la que pertenece esta entidad, el fichero que representa este valor ya es verificado que existe cuando se ingresa en el contenedor $laEntidades de esta clase, se recomienda que el path sea absoluto, desde el hdd; si el fichero no existe no se ingresa la entidad errores al arreglo $laEntidades de esta clase. (obligatorio en dependencia del procedimiento gestor)

' referencia_path_entidad ': (string) sus posibles valores son *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el path_entidad_clase relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el path_entidad_clase relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el path_entidad_clase de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio en dependencia del procedimiento gestor)

' path_control ': es el path de un fichero que controlará condiciones y/o el inicio o inclusión de una entidad errores, el gestor de errores espera un retorno de este control con un arreglo asociativo donde se encuentren los datos namespace_entidad_errores, clase_entidad_errores, id_entidad _errores, path_entidad_clase, referencia_path_entidad , tipo_entidad , instancias con sus datos, o combinaciones de estos, este arreglo será procesado por esta clase Emod\Nucleo\Errores que es quien gestiona y administra los Errores en el sistema Sockeletom. (obligatorio en dependencia del procedimiento gestor)

' referencia_path_control ': (string) es la forma que se hace referencia a $Ls_path_control_errores, tiene tres posibles valores, *relativo*, *relativo_esquelemod* y *absoluto*. Relativo es cuando el sistema debe tomar el $Ls_path_control_errores relativo al valor de la propiedad $lsPathDirRaizEntidades declarado en esta misma clase, y tomado de la configuración del sistema. Relativo_esquelemod es un caso especial de relativo y es cuando el sistema debe tomar el $Ls_path_control_errores relativo al valor del path del directorio esquelemod $EEoNucleo->pathDirEsquelemod(). Absoluto es cuando el sistema debe tomar el $Ls_path_control_errores de forma absoluta, desde la raíz del hdd, para esto puede auxiliarse del procedimiento $EEoNucleo->pathDirEsquelemod() si es necesario. (obligatorio en dependencia del procedimiento gestor)

' tipo_entidad ': (string) sus posibles valores son 'clase' u 'objeto', contiene el tipo de entidad, es decir si es de tipo clase (procedimientos estáticos), u objeto (instancia de una clase). (obligatorio en dependencia del procedimiento gestor)

' iniciacion ': (string) es la forma como se iniciará cada id_entidad_errores, este parámetro solo es necesario si se necesita iniciar una entidad con un grupo de parámetros determinados. Los posibles valores de iniciación son 'construct ' o ' nombre de procedimiento'. Cuando se da el valor 'construct ' el gestor de entidades errores al crear un objeto instancia de la clase base que se gestiona, lo crea a través de un constructor al que el gestor de entidades errores le pasa los parámetros con los que iniciará la instancia. El otro posible valor es el nombre de un procedimiento cualquiera de la instancia al que el gestor de entidades errores le pasa los parámetros con los que iniciará la instancia. En caso de que la entidad errores sea de tipo clase (tipo_entidad = 'clase') es obvio que no se debe utilizar el valor 'construct ' sino ' nombre de procedimiento', y se ejecutará el procedimiento con sus parámetros solo cuando se gestione por primera vez la primera entidad errores del arreglo instancias de la clase base que se gestione, ya que con este está iniciada la clase de procedimiento estático de iniciación. Los valores de los parámetros que utilizará la forma de iniciación serán colocados en un arreglo no asociativo en el elemento ['instancias'][id_entidad_errores]['parametros_iniciacion'] de la clase base, por lo que cada entidad errores puede tener sus valores propios, excepto en caso de que la entidad errores sea de tipo clase (tipo_entidad = 'clase') que con el primer id_ entidad_errores se inicia la clase base. Los elementos ['instancias'][id_entidad_errores]['parametros_iniciacion'] se explican a continuación en esta ayuda. (opcional en dependencia del procedimiento gestor)

' instancias ': (array) es un arreglo con los identificadores de entidades errores a reservar desde este procedimiento, debe tener como mínimo un identificador ya que posteriormente a ser registrado admite crear nuevas entidades con sus identificadores, este arreglo asociativo tiene como llaves los id_entidad_errores. (obligatorio en dependencia del procedimiento gestor)

id_entidad_errores (string) es un nombre o identificador de cada entidad errores instancia de la clase correspondiente. y a su vez este id_entidad_errores es un arreglo asociativo al que se puede insertar una llave de nombre parametros_iniciacion y otra de nombre datos, posteriormente este arreglo id_entidad_errores se le incorporan otros elementos con llave asociativa para el trabajo de esta clase. Pueden existir tantos id_entidad_errores como sea necesario ya que la entidad errores puede ser una instancia de una clase con valores muy específicos o personalizados en sus propiedades. (obligatorio 1 en dependencia del procedimiento gestor) (opcional ++)

'parametros_iniciacion ': (array) no asociativo cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad errores (id_entidad_errores), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad errores a gestionar sea de tipo clase ($Ls_tipo_entidad = 'clase') entonces este parámetro solo debe existir en el primer id_entidad_errores declarado en el arreglo instancias de la clase base, porque solo cuando sea llamado por primera vez esta entidad errores es que se buscará el elemento 'parametros_iniciacion' en su estructura. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional en dependencia del procedimiento gestor)

'datos': (array) es un arreglo que permite la inserción en él, de datos que considere conveniente el proceso que hace el ingreso de la entidad errores en esta clase Errores; estos datos se crean en el momento del ingreso de la entidad errores sin que permita cambios posteriores. (opcional en dependencia del procedimiento gestor)

se aconseja por parte del grupo de desarrollo del núcleo Sockeletom que los namespaces se declaren de forma absoluta, comenzando con '\ '

este procedimiento retorna true si la gestión de al menos una entidad errores fue llevada a cabo con éxito, de lo contrario retorna null.

public static function existenciaEntidad($Ls_namespace , $Ls_clase , $Ls_id_error = null)
es el procedimiento encargado de gestionar si existe una entidad errores determinada en el contenedor $laEntidades de esta clase, si existe una entidad errores a disposición de pedido y uso, esta entidad errores no tiene obligatoriamente que estar iniciada, es decir no tiene obligatoriamente que existir la instancia ya creada.

Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad errores. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad errores. (obligatorio).

$Ls_id_error: (string) es el nombre o id de la entidad errores hija de una clase ya ingresada en $this->laEntidades, este parámetro puede ser vacío, de ser vacío se entiende que se busca la existencia de una clase(\namespace\clase) ya ingresada en $this->laEntidades, clase base para entidades errores instancias(objeto) o clase (clase de procedimientos estáticos).

Si $Ls_id_error tiene valor y existe como entidad errores hija de la \namespace\clase base en la estructura de datos de $this->laEntidades entonces este procedimiento retorna el tipo_entidad (‘objeto’ o ‘clase’) de la clase base. Si no existe como entidad errores hija el procedimiento retorna null.

Si $Ls_id_error no tiene valor, o tiene valor vacio, existe la entidad errores \namespace\clase base en la estructura de datos de $this->laEntidades entonces este procedimiento retorna el tipo_entidad (‘objeto’ o ‘clase’) de la clase base. Si no existe como la entidad errores \namespace\clase base el procedimiento retorna null.

public static function ingresarEntidad($Ls_namespace , $Ls_clase , $Ls_id_error , $La_parametros_iniciacion = null)
es el procedimiento encargado de gestionar el ingreso de entidades errores hija de la \namespace\clase base en la estructura de datos de $this->laEntidades. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad errores, es obligatorio que exista ya en $this->laEntidades. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad errores, es obligatorio que exista ya en $this->laEntidades. (obligatorio).

$Ls_id_error: (string) es el nombre o id de una entidad errores a ingresar en la estructura instancias de la clase base \namespace\clase en la estructura de datos de $this->laEntidades(obligatorio).

$La_parametros_iniciacion: (array) cuyos elementos se tomarán como parámetros del procedimiento de iniciación de la clase base en la instanciación de la entidad errores ($Ls_id_error), si existe este parámetro, es arreglo y no es vacío entonces el procedimiento pasará los parámetros al procedimiento en la instanciación, los elementos del arreglo serán organizados como parámetros con el mismo orden que tienen en el arreglo, el elemento 0 será el parámetro 1. En caso de de que la entidad base de la entidad errores a gestionar sea de tipo clase (tipo_entidad = 'clase') no tiene efecto este parámetro ya que se gestionó en el primer elemento $Ls_id_error que se ingreso en la clase base. Estos parámetros serán eliminados después de ser iniciada la instancia u objeto (opcional)

public static function entidad($Ls_namespace , $Ls_clase , $Ls_id_error , $Ls_tipo_referencia = 'referencia')
es el procedimiento encargado de gestionar la entidad errores hija de la \namespace\clase base en la estructura de datos de $this->laEntidades identificada por $Ls_id_error, es decir si la entidad es de tipo ‘objeto’ retronará la referencia a un objeto instancia de la entidad errores, si la entidad es de tipo clase retornará un string compuesto por el namespace y nombre de la clase base. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad errores. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad errores. (obligatorio).

$Ls_id_error: (string) es el nombre o id de la entidad errores hija de una clase ya ingresada en $this->laEntidades, el string en $Ls_id_error tiene obligatoriamente que existir como elemento en la estructura instancias de la clase base en $this->laEntidades . (obligatorio).

$Ls_tipo_referencia: (string) es el tipo de referencia a objeto que devuelve el procedimiento en caso de que el tipo de entidad sea ‘objeto‘, este parámetro tiene dos valores posibles ‘referencia‘ y ‘clon‘, referencia devuelve un puntero a la entidad guardada en el arreglo $this->laEntidades, y clon devuelve un clon de este objeto. Si el tipo de entidad es ‘clase‘ no se considera este parámetro. Por defecto este parámetro tiene como valor ‘referencia‘.

Este procedimiento devuelve (retorna) una referencia o clon a objeto si el tipo_entidad de la clase base es (‘objeto’); devuelve (retorna) un string de la forma \namespace\clase si el tipo_entidad de la clase base es (‘clase’); de lo contrario retorna null.

La instanciación de los objetos ocurre en el momento que es pedido por primera vez una instancia a través de este procedimiento.

En el caso de la instanciación o iniciación de entidades de tipo ‘objeto’, e iniciación de entidades de tipo ‘clase’, estas entidades pueden emitir una excepción desde su constructor o procedimiento de iniciación si existe algún problema de instanciación o iniciación, ya que este procedimiento tiene un bloque try y catch para manejar las excepciones, el bloque catch espera un objeto de la clase \Exception y emite el mensaje de este, el código es de la siguiente forma:

```
catch( \Exception $e)
  {
     echo '<p>Excepción capturada: '.$e->getMessage(). "<p>" ;
     return null;
   }
```

Por lo que el lanzamiento de la excepción sería:

```
throw new \Exception('mensaje.');
```

Esta funcionalidad es muy útil para interrumpir la instanciación de una clase si existe algún error en esta instanciación, y así no se tienen objetos con problemas o sin uso por desperfectos, en este contenedor de instancias.

Lo anterior se aplica a entidades de tipo ‘objeto’ que lancen su excepción desde el constructor, o entidades de tipo ‘clase’ que lancen su excepción desde el procedimiento de iniciación; Quedan excluidos las entidades de tipo ‘objeto’ que lancen su excepción desde un procedimiento de iniciación(no __construct) ya que estos ya fueron instanciados existen como instancias en el contenedor, estos casos solo tienen la opción de emitir un mensaje.

public static function datosEntidad ($Ls_namespace , $Ls_clase , $Ls_id_errores)
es el procedimiento encargado de gestionar y retornar los datos, que de existir, tiene una entidad errores que haya ingresado en esta clase, estos datos son los correspondientes al elemento ‘datos’

que puede traer consigo una entidad errores en el momento de su ingreso al contenedor $laEntidades de esta clase. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad errores. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad errores. (obligatorio).

$Ls_id_error: (string) es el nombre o id de la entidad errores hija de una clase ya ingresada en $this->laEntidades, el string en $Ls_id_error tiene obligatoriamente que existir como elemento en la estructura instancias de la clase base en $this->laEntidades . (obligatorio).

este procedimiento retorna el arreglo 'datos' perteneciente a la entidad errores propietaria si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

public static function idEntidades ($Ls_namespace , $Ls_clase)
es el procedimiento encargado de gestionar y retornar los nombres o identificadores de entidades errores que hayan ingresado en el namespace y clase correspondientes a los parámetros de este procedimiento, estos datos son los correspondientes al elemento 'instancias' que debe tener consigo cada clase base errores en el contenedor $laEntidades de esta clase. Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad errores. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad errores. (obligatorio).

si la gestión fue llevada a cabo con éxito este procedimiento retorna las llaves del array 'instancias' contenidos en el namespace y clase, clase base errores en el contenedor $laEntidades de esta clase, de lo contrario retorna null.

public static function eliminarEntidad ($Ls_namespace , $Ls_clase , $Ls_id_errores = null)
es el procedimiento encargado de eliminar una entidad errores clase base, o una entidad errores de la estructura 'instancias' de una clase base, existentes ambas en el contenedor $laEntidades de esta clase. Este procedimiento está sujeto al valor de la propiedad self::$lbPropietarioEntidad perteneciente a la sección de seguridad de esta clase Errores, si esta propiedad tiene valor true solo puede eliminar la entidad errores el proceso que figura como su propietario, de lo contrario puede eliminarse desde cualquier proceso, el proceso núcleo siempre tiene derechos de eliminar cualquier entidad errores.

Existen dos posibilidades de uso de este procedimiento:

La primera es eliminar una entidad errores clase base, es decir una namespace\clase registrada como clase base de entidades errores, para ello se pasan los valores correspondientes al namespace y clase de la entidad a eliminar, dejando vacío el parámetro $Ls_id_errores, recordar que esta operación está sujeto a la sección de seguridad de esta clase, donde se define quién puede eliminar una entidad determinada.

La segunda es eliminar una entidad errores de la estructura 'instancias' de una clase base, para ello debe existir la entidad errores en la estructura $laEntidades[$Ls_namespace][$Ls_clase] ['instancias'] de esta clase, el parámetro $Ls_id_errores no puede ser vacío, recordar que esta operación está sujeto a la sección de seguridad de esta clase, donde se define quien puede eliminar una entidad determinada.

Los parámetros se explican a continuación:

$Ls_namespace: (string) es el nombre del namespace al que pertenece la entidad errores. (obligatorio).

$Ls_clase: (string) es el nombre de la clase a la que pertenece la entidad errores. (obligatorio).

$Ls_id_error: (string) es el nombre o id de la entidad errores hija de una clase ya ingresada en $this->laEntidades, el string en $Ls_id_error tiene obligatoriamente que existir como elemento en la estructura instancias de la clase base en $this->laEntidades . (obligatorio).

este procedimiento retorna true si la gestión fue llevada a cabo con éxito, de lo contrario retorna null.

public static function implantarGestorErrores($La_gestor_errores , $Ls_id_gestor_errores , $Ls_tipos_error = null)
es el procedimiento encargado de implantar un gestor de errores implementando la función set_error_handler del lenguaje, lo anterior ocurre si existen los parámetros obligatorios y si el nuevo gestor a implantar no coincide con el gestor actual (de turno), no permitiendo que se dupliquen gestores de errores consecutivos en la pila self::$laPilaGestoresErrores que esta clase tiene para la gestión de gestores de errores. Actualiza la pila de gestores de error junto con los valores self::$laGestorErroresEjecucion y self::$lsIdGestorErroresEjecucion

Para implantar el gestor de errores este procedimiento utiliza la función set_error_handler del lenguaje.

Este procedimiento trabaja únicamente con procedimientos pertenecientes a objetos, como gestores de error.

Los parámetros se explican a continuación:

$La_gestor_errores: (array) $La_gestor_errores[0] es el apuntador a un objeto y $La_gestor_errores[1] el procedimiento en ese objeto($La_gestor_errores[0]) que hará de gestor de errores. (obligatorio).

$Ls_id_gestor_errores: (string) contendrá un identificador para el gestor de errores a implantar, de modo que se pueda identificar de forma rápida y muy simple cada gestor de errores (obligatorio).

$Ls_tipos_error : Se puede usar para enmascarar la provocación de la función *error_handler* al igual que la configuración error_reporting ini controla los errores que se muestran. Sin esta máscara establecida *error_handler* será llamada para cada error sin tener en cuenta la configuración de error_reporting. (opcional)

Este procedimiento retorna el gestor de errores anterior directamente desde set_error_handler si la gestión de este procedimiento es satisfactoria, de lo contrario retorna null.

public static function implantarGestorErroresAnterior()
es el procedimiento encargado de implantar un gestor de errores que figura como el gestor de errores implantado anteriormente al actual(en ejecución) (de turno), esta gestión se hace con la función restore_error_handler del lenguaje, por lo que si no utiliza uniformemente en los script clientes del sistema Sockeletom, los procedimiento de su tratamiento de errores entonces puede existir divergencias entre los valores de la pila (LIFO) self::$laPilaGestoresErrores y los valores erales de restore_error_handler. Este procedimiento no recurre a la pila (LIFO) self:: $laPilaGestoresErrores para implantar el gestor de errores anterior, solo recurre a la pila (LIFO) self::$laPilaGestoresErrores para actualizarla

Este procedimiento retorna el resultado de la implementación de la función restore_error_handler del lenguaje si la gestión de este procedimiento es satisfactoria, de lo contrario retorna null.

public static function implantarGestorErroresPila($Ls_posicion_gestor_pila)
es el procedimiento encargado de implantar un gestor de errores que figura, se encuentra, en la pila (LIFO) self::$laPilaGestoresErrores que esta clase tiene para la gestión de gestores de errores. Actualiza la pila de gestores de error junto con los valores self::$laGestorErroresEjecucion y self:: $lsIdGestorErroresEjecucion.

Los parámetros se explican a continuación:

$Ls_posicion_gestor_pila: (integer) es la posición que ocupa en self::$laPilaGestoresErrores el gestor a implantar, este entero es la llave(índice) del arreglo self::$laPilaGestoresErrores teniendo como primer elemento 0. (obligatorio).

Este procedimiento retorna el gestor de errores anterior directamente desde set_error_handler si la gestión de este procedimiento es satisfactoria, de lo contrario retorna null.

public static function gestorErroresEjecucion()
es el procedimiento encargado de devolver (return) los valores correspondientes al gestor de errores actual (en ejecución), específicamente retorna un arreglo con la siguiente estructura array(self:: $lsIdGestorErroresEjecucion , self::$laGestorErroresEjecucion), es decir un arreglo no asociativo donde el valor del índice 0 es el identificador del gestor de errores en ejecución y el valor del índice 1 es un array que contendrá en el índice 0 un objeto y en el índice 1 un string con el nombre del procedimiento perteneciente al objeto (elemento 0) que será el gestor de errores en ejecución, actual(de turno). Si la gestión del procedimiento no es satisfactoria retorna null.

clase_eerrores

Este fichero contiene la clase EErrores. Esta clase está compuesta por propiedades, y procedimientos para la gestión de Errores. Una instancia de esta clase define el manejo de cada error específico, así como la fuente, las propiedades del error, y su formato, también el destino errorlog junto a las propiedades y formato del errorlog; es quién ya iniciado y configurado nos permite con solo poner un identificador de error, hacer el tratamiento del error en sus acciones de implantar un manejador de errores personalizado, buscar las características del error en la fuente de estos, formatear el mensaje, mostrar el mensaje, detener o no el script en dependencia del tipo de error, formatear el mensaje de error log y ponerlo en su destino, implantar el manejador de errores anterior al actual(personalizado),etc., estas y otras son las operaciones básicas que nos permite automatizar una instancia de esta clase.

A continuación mostramos una lista de las propiedades y procedimientos que esta clase pone a disposición de los códigos clientes del Sockeletom:

Namespace: Emod\Nucleo\Errores
Nombre de la clase: EErrores
Herencia: use \Emod\Nucleo\DependenciasEntidadesEmod ;

Propiedades:

protected $EEoNucleo = null ;
puntero al objeto EEoNucleo ;

private $iniciacion = null ;
boolean, propiedad para chequear la iniciación de esta clase.

private $liTiposError = null ;

tipo de error, valor referente a los tipos de errores E_USER_ERROR , E_USER_WARNING , y E_USER_NOTICE establecidos en los niveles de errores de PHP, si no comprende uno de los tipos de errores anteriores se asume el string'Tipo de error desconocido' .

Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

private $GEDEGEDefecto = null ;
string o puntero a objeto, valor por defecto de la entidad que hace de Gestor de Estructuras de Datos para Entidades Gestoras de Errores(GEDEGE), si el GEDEGE es una clase de procedimientos estáticos(tipo_clase) esta propiedad contiene un valor de tipo string con el \namespace\clase del GEDEGE , si el GEDEGE es una clase base para construir instancias esta propiedad contiene un valor de tipo referencia al objeto instancia de la clase base del GEDEGE.

Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

private $lsTipoEntidadGEDEGEDefecto = null ;
string, valor del tipo de la entidad que hace de Gestor de Estructuras de Datos para Entidades Gestoras de Errores(GEDEGE) por defecto, si el GEDEGE es una clase de procedimientos estáticos(tipo_clase) esta propiedad contiene el valor 'clase', si el GEDEGE es una clase base para construir instancias esta propiedad contiene el valor 'objeto'.

Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

private $lsIdGestorErrores = 'gestorErroresEmod' ;
string, valor identificador de la función gestor de errores que se cargara con set_error_handler como gestor de errores por defecto del sistema Sockeletom, este identificador es obligatorio su existencia.

Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

private $lsFuncionGestorErrores = null ;
string, valor nombre de la función gestor de errores que se cargara con set_error_handler como gestor de errores por defecto del sistema Esquelemod, este nombre es obligatorio su existencia. Es dentro de esta clase ($this) el procedimiento escogido para implantarse como gestor de errores con la función del lenguaje set_error_handler.

Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

private $laMatrizElementosError = null ;

array, donde se definen los elementos que serán manejados por esta clase para el tratamiento de errores, un ejemplo de elementos puede ser identificador del error, tipo del error, mensaje a mostrar etc. Estos elementos servirán como mascara, patrón para comparar y filtrar o asociar valores de datos gestionados en los procedimientos de esta clase, funcionando como campos o variables con que regir datos que se gestionen en esta clase. Este arreglo consta de dos llaves asociativas que explicamos a continuación:

$laMatrizElementosError['La_miembros_error'] define los campos o elementos que se gestionaran para un mensaje u otro destino de errores un ejemplo puede ser array('id' , 'tipo' , 'tratamiento' , 'alcance' , 'mensaje').

$laMatrizElementosError['La_miembros_errorlog'] define los campos o elementos que se gestionaran para un contenedor de registros de errores, a esta propiedad siempre se le adicionan los elementos de la propiedad $La_miembros_error, un ejemplo puede ser array('date' , 'aplicacion' , 'modulo' , 'opcion')+ ('id' , 'tipo' , 'tratamiento' , 'alcance' , 'mensaje').

Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

private $laFuenteDatosError = null ;
array, donde se definen los datos para el acceso a la fuente de datos de los errores, estos datos hacen referencia a la fuente donde se gestionaran los datos error y los datos errorlog, la estructura interna de este arreglo depende de la forma que los deba leer el GEDEGE asignado a la instancia de esta clase, ya que $laFuenteDatosError será pasado como argumento a los procedimientos del GEDEGE en su gestión; un ejemplo puede ser:

$laFuenteDatosError['path_fich_errorlog'] (esto es en caso de ser un fichero)
$laFuenteDatosError['path_fich_error'](string)

$laFuenteDatosError['servidor_bd'](string)(esto es en caso de ser una Bd)
$laFuenteDatosError['usuario_bd'](string)
$laFuenteDatosError['password_bd'](string)

Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

private $laFormatoFiltrado = null ;
array, donde se definen los datos para el filtrado de paquetes de datos en términos de el funcionamiento del filtrado y la estructura de los paquetes de datos, estos datos hacen referencia a elementos necesarios en el filtrado y/o formato de los datos en la fuente donde estos se encuentren, éstos datos hacen referencia a los datos error y los datos errorlog, la estructura interna de este arreglo depende de la forma que los deba leer el GEDEGE asignado a la instancia de esta clase, ya

que $laFormatoFiltrado será pasado como argumento a los procedimientos del GEDEGE en su gestión; un ejemplo puede ser:

$laFormatoFiltrado['formato']['separador'](string) es el separador de datos en una línea de texto en un txt
$laFormatoFiltrado['filtrado']['marca_ausente'](boolean) si un campo de datos tiene valor vacío, marca el campo con el propio nombre del campo entre paréntesis como valor del campo.
$laFormatoFiltrado['filtrado']['llave_unica'](string) define una llave como condición para que cuando esta llave y su valor sean encontrados en la línea del txt, se detenga la búsqueda por las líneas restantes del fichero, es decir se detenga el proceso de filtrado y se pase a al termino de la gestión de un procedimiento determinado.

Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

private $laFormatoMensajeError = null ;
array, donde se definen los elementos que conformarán el mensaje de error, es el grupo de elementos que se concatenaran para formar el mensaje de error emitido por $this->error(), estos elementos conformarán el mensaje en el mismo orden en que aparecen los elementos, cada elemento de este arreglo será tenido en cuenta como llave del arreglo asociativo fuente para los datos que conformarán el mensaje de error. Un ejemplo puede ser array('id' , 'tipo' , 'mensaje')

Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

private $laFormatoMensajeErrorlog = null ;
array, donde se definen los elementos que conformarán el contenido del registro de errorlog, es el grupo de elementos que se concatenaran para formar el contenido del registro de errorlog emitido por $this->error(), estos elementos conformarán el registro en el mismo orden en que aparecen los elementos, cada elemento de este arreglo será tenido en cuenta como llave del arreglo asociativo fuente para los datos que conformarán el registro de errorlog. Un ejemplo puede ser array('id' , 'tipo' , 'mensaje' , 'tipo' , 'tiempo' , 'fichero' , 'línea', 'proceso').

Aclaramos que esta propiedad es independiente de $laFormatoMensajeError.

Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

Procedimientos:

__construct ($La_gedege_emod , $La_funcion_gestor_errores , $La_matriz_elementos_error , $La_formato_mensaje , $La_fuente_datos_error , $La_formato_filtrado , $Li_tipos_error = null)
Constructor, los parámetros se explican a continuación:

$La_gedege_emod (array) datos de la entidad que hace de Gestor de Estructuras de Datos para Entidades Gestoras de Errores(GEDEGE), $La_gedege_emod puede tener dos estructuras:

primera estructura posible:

$La_gedege_emod['namespace'],$La_gedege_emod['clase'],$La_gedege_emod['id_entidad'],donde

$La_gedege_emod['namespace'] (string) es el nombre del namespace al que pertenece la entidad GEDEGE. (obligatorio)

$La_gedege_emod['clase'] (string) es el nombre de la clase a la que pertenece la entidad GEDEGE. (obligatorio)

$La_gedege_emod['id_entidad'] (string) es un nombre o identificador de la entidad GEDEGE instancia de la clase correspondiente. (obligatorio)

segunda estructura posible:

$La_gedege_emod[namespace][clase]['instancias'][id_entidad] donde

namespace (string) es el nombre del namespace al que pertenece la entidad GEDEGE. (obligatorio)

clase (string) es el nombre de la clase a la que pertenece la entidad GEDEGE. (obligatorio)

id_entidad (string) es un nombre o identificador de la entidad GEDEGE instancia de la clase correspondiente, y como el elemento 'instancias' es un arreglo que puede contener múltiples id_entidades , este procedimiento tomará como id_entidad para esta funcionalidad la primera id_entidad del arreglo, es decir la id_entidad que se encuentra en el índice 0 de este arreglo 'instancias'. (obligatorio)

Es obligatorio que la entidad GEDEGE a gestionar ya exista ingresada su clase base en la clase errores.

$La_funcion_gestor_errores (array) datos de la función gestor de errores que se cargara con set_error_handler como gestor de errores por defecto del sistema Sockeletom, su estructura es la siguiente:

$La_funcion_gestor_errores['identificador'] string, valor identificador de la función gestor de errores que se cargara con set_error_handler como gestor de errores por defecto del sistema Sockeletom, este identificador es obligatorio su existencia.
Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

$La_funcion_gestor_errores['nombre_funcion'] string, valor nombre de la función gestor de errores que se cargara con set_error_handler como gestor de errores por defecto del sistema Sockeletom, este nombre es obligatorio su existencia. Es dentro de esta clase ($this) el procedimiento escogido para implantarse como gestor de errores con la función del lenguaje set_error_handler.
Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

$La_matriz_elementos_error (array) donde se definen los elementos que serán manejados por esta clase para el tratamiento de errores, un ejemplo de elementos puede ser identificador del error, tipo del error, mensaje a mostrar etc. Estos elementos servirán como mascara, patrón para comparar y filtrar o asociar valores de datos gestionados en los procedimientos de esta clase, funcionando como campos o variables con que regir datos que se gestionen en esta clase. Este arreglo consta de dos llaves asociativas que explicamos a continuación:

$La_matriz_elementos_error ['miembros_error'] (array) define los campos o elementos que se gestionaran para un mensaje u otro destino de errores un ejemplo puede ser array('id' , 'tipo' , 'tratamiento' , 'alcance' , 'mensaje')(obligatorio).

$La_matriz_elementos_error ['miembros_errorlog'] (array) define los campos o elementos que se gestionaran para un contenedor de registros de errores, a esta propiedad siempre se le adicionan los elementos de la propiedad $La_miembros_error, un ejemplo puede ser array('date' , 'aplicacion' , 'modulo' , 'opcion')+ ('id' , 'tipo' , 'tratamiento' , 'alcance' , 'mensaje')(obligatorio)..

Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

$La_formato_mensaje (array)donde se definen los elementos que conformarán los mensajes de error y errorlog, es el grupo de elementos que se concatenaran para formar el mensaje de error y errorlog emitido por $this->error(). Este arreglo consta de dos llaves asociativas que explicamos a continuación:

$La_formato_mensaje['formato_mensaje_error'] (array), donde se definen los elementos que conformarán el mensaje de error, es el grupo de elementos que se concatenaran para formar el mensaje de error emitido por $this->error(), estos elementos conformarán el mensaje en el mismo orden en que aparecen los elementos, cada elemento de este arreglo será tenido en cuenta como llave del arreglo asociativo fuente para los datos que conformarán el mensaje de error. Un ejemplo puede ser array('id' , 'tipo' , 'mensaje')

Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

$La_formato_mensaje['formato_mensaje_error'] (array), donde se definen los elementos que conformarán el contenido del registro de errorlog, es el grupo de elementos que se concatenaran para formar el contenido del registro de errorlog emitido por $this->error(), estos elementos conformarán el registro en el mismo orden en que aparecen los elementos, cada elemento de este arreglo será tenido en cuenta como llave del arreglo asociativo fuente para los datos que conformarán el registro de errorlog. Un ejemplo puede ser array('id' , 'tipo' , 'mensaje' , 'tipo' , 'tiempo' , 'fichero' , 'línea', 'proceso').

Aclaramos que esta llave es independiente de $La_formato_mensaje['formato_mensaje_error'].

Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

$La_fuente_datos_error (array), donde se definen los datos para el acceso a la fuente de datos de los errores, estos datos hacen referencia a la fuente donde se gestionaran los datos error y los datos errorlog, la estructura interna de este arreglo depende de la forma que los deba leer el GEDEGE asignado a la instancia de esta clase, ya que $La_fuente_datos_error será pasado como argumento a los procedimientos del GEDEGE en su gestión; un ejemplo puede ser:

(esto es en caso de ser un fichero)
$La_fuente_datos_error ['path_fich_errorlog'] (string)
$La_fuente_datos_error ['path_fich_error'](string)

(esto es en caso de ser una Bd)
$La_fuente_datos_error ['servidor_bd'](string)
$La_fuente_datos_error ['usuario_bd'](string)
$La_fuente_datos_error ['password_bd'](string)

Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

$La_formato_filtrado (array) , donde se definen los datos para el filtrado de paquetes de datos en términos de el funcionamiento del filtrado y la estructura de los paquetes de datos, estos datos hacen referencia a elementos necesarios en el filtrado y/o formato de los datos en la fuente donde estos se encuentren, éstos datos hacen referencia a los datos error y los datos errorlog, la estructura interna de este arreglo depende de la forma que los deba leer el GEDEGE asignado a la instancia de esta clase, ya que $La_formato_filtrado será pasado como argumento a los procedimientos del GEDEGE en su gestión; un ejemplo puede ser:

$La_formato_filtrado['formato']['separador'](string) es el separador de datos en una línea de texto en un txt

$La_formato_filtrado['filtrado']['marca_ausente'](boolean) si un campo de datos tiene valor vacío, marca el campo con el propio nombre del campo entre paréntesis como valor del campo.
$La_formato_filtrado['filtrado']['llave_unica'](string) define una llave como condición para que cuando esta llave y su valor sean encontrados en la línea del txt, se detenga la búsqueda por las líneas restantes del fichero, es decir se detenga el proceso de filtrado y se pase a al termino de la gestión de un procedimiento determinado.

Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

$Li_tipos_error (integer), tipo de error, valor referente a los tipos de errores E_USER_ERROR , E_USER_WARNING , y E_USER_NOTICE establecidos en los niveles de errores de PHP, si no comprende uno de los tipos de errores anteriores se asume el string' Tipo de error desconocido' .

Esta propiedad es específicas de cada instancia de esta clase, por eso se deben introducir en la iniciación de la instancia.

Este procedimiento retorna true si su gestión se llevó a cavo con éxito, de lo contrario emite un error y finaliza la ejecución del sistema a través de la función die() del lenguaje.

public function gestorErroresEmod($errno, $errstr)
es, la función que hará de función manejadora de errores implantada desde set_error_handler para la gestión de errores por el usuario, en este caso el usuario sería todo aquel código que invocara el procedimiento error() de esta clase, los parámetros se explican a continuación:

$errno (integer), tipo de error, valor referente a los tipos de errores E_USER_ERROR , E_USER_WARNING , y E_USER_NOTICE establecidos en los niveles de errores de PHP, si no comprende uno de los tipos de errores anteriores se asume el string' Tipo de error desconocido' .

$errstr (string), valor mensaje de error a emitir.

Recordar que puede ser otra función de esta clase la que se escoja como manejadora de errores quedando esta función fuera de servicio, esta decisión se toma en el valor de la propiedad $lsFuncionGestorErrores de esta clase.

protected function implementarProcedimientosEntidades($Ls_nombre_funcion , &$La_parametros = null)
es, el encargado de establecer una relación uno a uno de procedimientos de esta clase con procedimientos del GEDEGE declarado para una instancia de esta clase; esta relación uno a uno construye (fabrica) un puente entre un procedimiento de esta clase y su homólogo en el GEDEGE , puente por el que se transmiten datos del procedimiento local al procedimiento del GEDEGE y se reciben resultados del procedimiento del GEDEGE al procedimiento local, ambos procedimientos

tienen que tener el mismo nombre y los parámetros que se pasan por el procedimiento local se pasan al procedimiento del GEDEGE en el mismo orden que en el local. Los parámetros se explican a continuación:

$Ls_nombre_funcion (string), nombre de la función en el GEDEGE .

$La_parametros(array), arreglo cuyos elementos son los valores de los parámetros que se pasarán al procedimiento en el GEDEGE, estos valores están en el mismo orden que tienen los parámetros en el procedimiento del GEDEGE, teniendo como parámetro1 en el procedimiento del GEDEGE el elemento 0 de este arreglo.

public function filtrarDatosError()

este procedimiento hace uso del procedimiento implementarProcedimientosEntidades de esta clase, por lo que su función y objetivo es única pero sus parámetros dependen del funcionamiento del procedimiento homólogo en el GEDEGE correspondiente a en la propiedad $GEDEGEDefecto de cada instancia de esta clase. Quien fija o decide la función y objetivo es este procedimiento, quien decide como se gestiona y ejecuta es el procediendo homólogo en el GEDEGE

La función y objetivo de este procedimiento es filtrar un arreglo asociativo con pares $Ls_key=>$Ls_valor aportado por un cliente (arreglo cliente a filtrar), el filtrado se hace comparándolo con un arreglo que es matriz de las $Ls_key ($La_matriz_elementos_error['miembros_error']) (arreglo filtro)que ya existe en las propiedades de la instancia(no se tienen que proporcionar como parámetros en este procedimiento), y formando o resultando un arreglo nuevo ($La_datos_error_filtro_result) con los pares $Ls_key=>$Ls_valor cuyos $Ls_key se encontraban en ambos arreglos (el arreglo cliente y el arreglo matriz).

Además también se ponen en el arreglo resultante $La_datos_error_filtro_result del filtrado, los elementos que están en la matriz (arreglo filtro) pero no en el arreglo cliente(arreglo cliente a filtrar), lo anterior se hace poniendo el índice $Ls_key y como valor de este su propio identificador string entre paréntesis('key').Esta funcionalidad depende de La_formato_filtrado['filtrado']['marca_ausente'] elemento que se explica más adelante.

El elemento $Ls_key que exista en el arreglo cliente(arreglo cliente a filtrar) y no exista en la matriz(arreglo filtro), no pasa a formar parte del arreglo resultante $La_datos_error_filtro_result.

Para gestionar correctamente su función y objetivo este procedimiento, al establecer un puente con su homólogo en el GEDEGE y proporcionarle a este los parámetros necesarios, hace uso de listas de argumentos de longitud variable, donde la ubicación de los parámetros están normados de la siguiente forma:

1- se ubican(por parte del usuario) los n parámetros necesarios en los primeros elementos o índices, poniendo entre ellos el parámetro arreglo cliente(arreglo cliente a filtrar) en su lugar correspondiente, llamaremos simbólicamente $La_datos_error_filtro a este parámetro .
2- A continuación de los n parámetros se ubica(por parte del usuario) como último parámetro de este procedimiento el parámetro que llamaremos $La_formato_filtrado donde se definen los datos para el filtrado de paquetes de datos en términos de el funcionamiento del filtrado y la estructura de los paquetes de datos, estos datos hacen referencia a elementos necesarios en el filtrado y/o formato de los datos en la fuente donde estos se encuentren, éstos datos hacen referencia a los datos error y los datos errorlog, la estructura interna de este arreglo depende de la forma que los deba leer el GEDEGE asignado a la instancia de esta clase, ya que $La_formato_filtrado será pasado como argumento al procedimiento del GEDEGE en su gestión; un ejemplo puede ser:

$La_formato_filtrado['formato']['separador'](string) es el separador de datos en una línea de texto
$La_formato_filtrado['filtrado']['marca_ausente'](boolean) si un campo de datos tiene valor vacío, marca el campo con el propio nombre del campo entre paréntesis como valor del campo.

En esta misma clase existe una propiedad $this->laFormatoFiltrado que hace referencia al formato y filtrado de datos de errores, este procedimiento busca la existencia de este parámetro $La_formato_filtrado y de existir adiciona a este los elementos de la propiedad $this->laFormatoFiltrado de esta clase, de no existir este parámetro $La_formato_filtrado el procedimiento lo crea con el valor de $this->laFormatoFiltrado.

3- A continuación del paso 2 este procedimiento adiciona a la lista de argumentos de longitud variable(al final de la lista) la variable $this->laMatrizElementosError, concluyendo con la lista de parámetros y pasándolos al procedimiento homólogo en el GEDEGE.

Después del paso 3 el procedimiento espera el retorno del resultado del procedimiento homólogo en el GEDEGE para emitirlo como resultado de su gestión en forma de retorno.

La explicación exacta de cuantos parámetros tiene cada procedimiento y las características de estos, lo tiene la entidad GEDEGE correspondiente a la instancia de esta clase, ya que este procedimiento toma un grupo de parámetros desconocidos y adiciona a estos otros parámetros conocidos para conformar la lista que pasa o envía al procedimiento homólogo en el GEDEGE, en este procedimiento solo se pueden explicar los parámetros conocidos desde él.

Este procedimiento retorna el resultado(return) del procedimiento homólogo en el GEDEGE .

public function crearDatosError()

este procedimiento hace uso del procedimiento implementarProcedimientosEntidades de esta clase, por lo que su función y objetivo es única pero sus parámetros dependen del funcionamiento del procediendo homólogo en el GEDEGE correspondiente a en la propiedad $GEDEGEDefecto de

cada instancia de esta clase. Quien fija o decide la función y objetivo es este procedimiento, quien decide como se gestiona y ejecuta es el procediendo homólogo en el GEDEGE

La función y objetivo de este procedimiento es crear un nuevo registro de datos error en una fuente de datos errores, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar errores.

Para gestionar correctamente su función y objetivo este procedimiento, al establecer un puente con su homólogo en el GEDEGE y proporcionarle a este los parámetros necesarios, hace uso de listas de argumentos de longitud variable, donde la ubicación de los parámetros están normados de la siguiente forma:

1- Se ubican(por parte del usuario) los n parámetros necesarios en los primeros elementos o índices.
2- A continuación de los n parámetros se ubica(por parte del usuario) como último parámetro de este procedimiento el parámetro que llamaremos $La_formato_filtrado donde se definen los datos para el filtrado de paquetes de datos en términos de el funcionamiento del filtrado y la estructura de los paquetes de datos, estos datos hacen referencia a elementos necesarios en el filtrado y/o formato de los datos en la fuente donde estos se encuentren, éstos datos hacen referencia a los datos error y los datos errorlog, la estructura interna de este arreglo depende de la forma que los deba leer el gedege asignado a la instancia de esta clase, ya que $La_formato_filtrado será pasado como argumento al procedimiento del GEDEGE en su gestión; un ejemplo puede ser:

$La_formato_filtrado['formato']['separador'](string) es el separador de datos en una línea de texto
$La_formato_filtrado['filtrado']['marca_ausente'](boolean) si un campo de datos tiene valor vacío, marca el campo con el propio nombre del campo entre paréntesis como valor del campo.

En esta misma clase existe una propiedad $this->laFormatoFiltrado que hace referencia al formato y filtrado de datos de errores, este procedimiento busca la existencia de este parámetro $La_formato_filtrado y de existir adiciona a este los elementos de la propiedad $this->laFormatoFiltrado de esta clase, de no existir este parámetro $La_formato_filtrado el procedimiento lo crea con el valor de $this->laFormatoFiltrado.

3- A continuación del paso 2 este procedimiento adiciona a la lista de argumentos de longitud variable(al final de la lista) la variable $this->laMatrizElementosError, y $this->laFuenteDatosError, en ese orden, concluyendo con la lista de parámetros y pasándolos al procedimiento homólogo en el GEDEGE.

Después del paso 3 el procedimiento espera el retorno del resultado del procedimiento homólogo en el GEDEGE para emitirlo como resultado de su gestión en forma de retorno.

La explicación exacta de cuantos parámetros tiene cada procedimiento y las características de estos, lo tiene la entidad GEDEGE correspondiente a la instancia de esta clase, ya que este procedimiento toma un grupo de parámetros desconocidos y adiciona a estos otros parámetros conocidos para conformar la lista que pasa o envía al procedimiento homólogo en el GEDEGE, en este procedimiento solo se pueden explicar los parámetros conocidos desde él.

Este procedimiento retorna el resultado(return) del procedimiento homólogo en el GEDEGE .

public function eliminarDatosError()

este procedimiento hace uso del procedimiento implementarProcedimientosEntidades de esta clase, por lo que su función y objetivo es única pero sus parámetros dependen del funcionamiento del procediendo homólogo en el GEDEGE correspondiente a en la propiedad $GEDEGEDefecto de cada instancia de esta clase. Quien fija o decide la función y objetivo es este procedimiento, quien decide como se gestiona y ejecuta es el procediendo homólogo en el GEDEGE

La función y objetivo de este procedimiento es eliminar registros de datos error en una fuente de datos errores, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar errores.

Para gestionar correctamente su función y objetivo este procedimiento, al establecer un puente con su homólogo en el GEDEGE y proporcionarle a este los parámetros necesarios, hace uso de listas de argumentos de longitud variable, donde la ubicación de los parámetros están normados de la siguiente forma:

1- Se ubican(por parte del usuario) los n parámetros necesarios en los primeros elementos o índices.
2- A continuación de los n parámetros se ubica(por parte del usuario) como último parámetro de este procedimiento el parámetro que llamaremos $La_formato_filtrado donde se definen los datos para el filtrado de paquetes de datos en términos de el funcionamiento del filtrado y la estructura de los paquetes de datos, estos datos hacen referencia a elementos necesarios en el filtrado y/o formato de los datos en la fuente donde estos se encuentren, éstos datos hacen referencia a los datos error y los datos errorlog, la estructura interna de este arreglo depende de la forma que los deba leer el GEDEGE asignado a la instancia de esta clase, ya que $La_formato_filtrado será pasado como argumento al procedimiento del GEDEGE en su gestión; un ejemplo puede ser:

$La_formato_filtrado['formato']['separador'](string) es el separador de datos en una línea de texto
$La_formato_filtrado['filtrado']['llave_unica'] (string), define un dato como condición para que cuando sea encontrado en el registro de datos, se detenga la búsqueda por los restantes registros, es decir se detenga el proceso de filtrado y se pase al termino de la gestión de este procedimiento.

En esta misma clase existe una propiedad $this->laFormatoFiltrado que hace referencia al formato y filtrado de datos de errores, este procedimiento busca la existencia de este parámetro $La_formato_filtrado y de existir adiciona a este los elementos de la propiedad $this->laFormatoFiltrado de esta clase, de no existir este parámetro $La_formato_filtrado el procedimiento lo crea con el valor de $this->laFormatoFiltrado.

3- A continuación del paso 2 este procedimiento adiciona a la lista de argumentos de longitud variable(al final de la lista) la variable $this->laMatrizElementosError, y $this->laFuenteDatosError, en ese orden, concluyendo con la lista de parámetros y pasándolos al procedimiento homólogo en el GEDEGE.

Después del paso 3 el procedimiento espera el retorno del resultado del procedimiento homólogo en el GEDEGE para emitirlo como resultado de su gestión en forma de retorno.

La explicación exacta de cuantos parámetros tiene cada procedimiento y las características de estos, lo tiene la entidad GEDEGE correspondiente a la instancia de esta clase, ya que este procedimiento toma un grupo de parámetros desconocidos y adiciona a estos otros parámetros conocidos para conformar la lista que pasa o envía al procedimiento homólogo en el GEDEGE, en este procedimiento solo se pueden explicar los parámetros conocidos desde él.

Este procedimiento retorna el resultado(return) del procedimiento homólogo en el GEDEGE .

public function leerDatosError()

este procedimiento hace uso del procedimiento implementarProcedimientosEntidades de esta clase, por lo que su función y objetivo es única pero sus parámetros dependen del funcionamiento del procediendo homólogo en el GEDEGE correspondiente a en la propiedad $GEDEGEDefecto de cada instancia de esta clase. Quien fija o decide la función y objetivo es este procedimiento, quien decide como se gestiona y ejecuta es el procediendo homólogo en el GEDEGE

La función y objetivo de este procedimiento es leer registros de datos error en una fuente de datos errores, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar errores.

Para gestionar correctamente su función y objetivo este procedimiento, al establecer un puente con su homólogo en el GEDEGE y proporcionarle a este los parámetros necesarios, hace uso de listas de argumentos de longitud variable, donde la ubicación de los parámetros están normados de la siguiente forma:

1- Se ubican (por parte del usuario) los n parámetros necesarios en los primeros elementos o índices.
2- A continuación de los n parámetros se ubica (por parte del usuario) como último parámetro de este procedimiento el parámetro que llamaremos $La_formato_filtrado donde se definen los

datos para el filtrado de paquetes de datos en términos de el funcionamiento del filtrado y la estructura de los paquetes de datos, estos datos hacen referencia a elementos necesarios en el filtrado y/o formato de los datos en la fuente donde estos se encuentren, éstos datos hacen referencia a los datos error y los datos errorlog, la estructura interna de este arreglo depende de la forma que los deba leer el GEDEGE asignado a la instancia de esta clase, ya que $La_formato_filtrado será pasado como argumento al procedimiento del GEDEGE en su gestión; un ejemplo puede ser:

$La_formato_filtrado['formato']['separador'](string) es el separador de datos en una línea de texto $La_formato_filtrado['filtrado']['llave_unica'] (string), define un dato como condición para que cuando sea encontrado en el registro de datos, se detenga la búsqueda por los restantes registros, es decir se detenga el proceso de filtrado y se pase al termino de la gestión de este procedimiento.

En esta misma clase existe una propiedad $this->laFormatoFiltrado que hace referencia al formato y filtrado de datos de errores, este procedimiento busca la existencia de este parámetro $La_formato_filtrado y de existir adiciona a este los elementos de la propiedad $this->laFormatoFiltrado de esta clase, de no existir este parámetro $La_formato_filtrado el procedimiento lo crea con el valor de $this->laFormatoFiltrado.

3- A continuación del paso 2 este procedimiento adiciona a la lista de argumentos de longitud variable(al final de la lista) la variable $this->laMatrizElementosError, y $this->laFuenteDatosError, en ese orden, concluyendo con la lista de parámetros y pasándolos al procedimiento homólogo en el GEDEGE.

Después del paso 3 el procedimiento espera el retorno del resultado del procedimiento homólogo en el GEDEGE para emitirlo como resultado de su gestión en forma de retorno.

La explicación exacta de cuantos parámetros tiene cada procedimiento y las características de estos, lo tiene la entidad GEDEGE correspondiente a la instancia de esta clase, ya que este procedimiento toma un grupo de parámetros desconocidos y adiciona a estos otros parámetros conocidos para conformar la lista que pasa o envía al procedimiento homólogo en el GEDEGE, en este procedimiento solo se pueden explicar los parámetros conocidos desde él.

Este procedimiento retorna el resultado(return) del procedimiento homólogo en el GEDEGE .

public function modificarDatosError()

este procedimiento hace uso del procedimiento implementarProcedimientosEntidades de esta clase, por lo que su función y objetivo es única pero sus parámetros dependen del funcionamiento del procediendo homólogo en el GEDEGE correspondiente a en la propiedad $GEDEGEDefecto de cada instancia de esta clase. Quien fija o decide la función y objetivo es este procedimiento, quien decide como se gestiona y ejecuta es el procediendo homólogo en el GEDEGE

La función y objetivo de este procedimiento es modificar registros de datos error en una fuente de datos errores, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar errores.

Para gestionar correctamente su función y objetivo este procedimiento, al establecer un puente con su homólogo en el GEDEGE y proporcionarle a este los parámetros necesarios, hace uso de listas de argumentos de longitud variable, donde la ubicación de los parámetros están normados de la siguiente forma:

1- Se ubican(por parte del usuario) los n parámetros necesarios en los primeros elementos o índices.
2- A continuación de los n parámetros se ubica(por parte del usuario) como último parámetro de este procedimiento el parámetro que llamaremos $La_formato_filtrado donde se definen los datos para el filtrado de paquetes de datos en términos de el funcionamiento del filtrado y la estructura de los paquetes de datos, estos datos hacen referencia a elementos necesarios en el filtrado y/o formato de los datos en la fuente donde estos se encuentren, éstos datos hacen referencia a los datos error y los datos errorlog, la estructura interna de este arreglo depende de la forma que los deba leer el GEDEGE asignado a la instancia de esta clase, ya que $La_formato_filtrado será pasado como argumento al procedimiento del GEDEGE en su gestión; un ejemplo puede ser:

$La_formato_filtrado['formato']['separador'](string) es el separador de datos en una línea de texto $La_formato_filtrado['filtrado']['llave_unica'] (string), define un dato como condición para que cuando sea encontrado en el registro de datos, se detenga la búsqueda por los restantes registros, es decir se detenga el proceso de filtrado y se pase al termino de la gestión de este procedimiento.

En esta misma clase existe una propiedad $this->laFormatoFiltrado que hace referencia al formato y filtrado de datos de errores, este procedimiento busca la existencia de este parámetro $La_formato_filtrado y de existir adiciona a este los elementos de la propiedad $this->laFormatoFiltrado de esta clase, de no existir este parámetro $La_formato_filtrado el procedimiento lo crea con el valor de $this->laFormatoFiltrado.

3- A continuación del paso 2 este procedimiento adiciona a la lista de argumentos de longitud variable(al final de la lista) la variable $this->laMatrizElementosError, y $this->laFuenteDatosError, en ese orden, concluyendo con la lista de parámetros y pasándolos al procedimiento homólogo en el GEDEGE.

Después del paso 3 el procedimiento espera el retorno del resultado del procedimiento homólogo en el GEDEGE para emitirlo como resultado de su gestión en forma de retorno.

La explicación exacta de cuantos parámetros tiene cada procedimiento y las características de estos, lo tiene la entidad GEDEGE correspondiente a la instancia de esta clase, ya que este procedimiento toma un grupo de parámetros desconocidos y adiciona a estos otros parámetros conocidos para

conformar la lista que pasa o envía al procedimiento homólogo en el GEDEGE, en este procedimiento solo se pueden explicar los parámetros conocidos desde él.

Este procedimiento retorna el resultado(return) del procedimiento homólogo en el GEDEGE .

public function crearFuenteDatosError()

este procedimiento hace uso del procedimiento implementarProcedimientosEntidades de esta clase, por lo que su función y objetivo es única pero sus parámetros dependen del funcionamiento del procediendo homólogo en el GEDEGE correspondiente a en la propiedad $GEDEGEDefecto de cada instancia de esta clase. Quien fija o decide la función y objetivo es este procedimiento, quien decide como se gestiona y ejecuta es el procediendo homólogo en el GEDEGE

La función y objetivo de este procedimiento es crear una fuente de datos errores, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar errores.

Para gestionar correctamente su función y objetivo este procedimiento, al establecer un puente con su homólogo en el GEDEGE y proporcionarle a este los parámetros necesarios, hace uso de listas de argumentos de longitud variable, donde la ubicación de los parámetros se corresponde con el orden en que se ingresan los parámetros a este procedimiento.

Este procedimiento espera el retorno del resultado del procedimiento homólogo en el GEDEGE para emitirlo como resultado de su gestión en forma de retorno.

La explicación exacta de cuantos parámetros tiene cada procedimiento y las características de estos, lo tiene la entidad GEDEGE correspondiente a la instancia de esta clase, ya que este procedimiento toma un grupo de parámetros desconocidos y adiciona a estos otros parámetros conocidos para conformar la lista que pasa o envía al procedimiento homólogo en el GEDEGE, en este procedimiento solo se pueden explicar los parámetros conocidos desde él.

Este procedimiento retorna el resultado(return) del procedimiento homólogo en el GEDEGE .

public function eliminarFuenteDatosError()

este procedimiento hace uso del procedimiento implementarProcedimientosEntidades de esta clase, por lo que su función y objetivo es única pero sus parámetros dependen del funcionamiento del procediendo homólogo en el GEDEGE correspondiente a en la propiedad $GEDEGEDefecto de cada instancia de esta clase. Quien fija o decide la función y objetivo es este procedimiento, quien decide como se gestiona y ejecuta es el procediendo homólogo en el GEDEGE

La función y objetivo de este procedimiento es eliminar una fuente de datos errores, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar errores.

Para gestionar correctamente su función y objetivo este procedimiento, al establecer un puente con su homólogo en el GEDEGE y proporcionarle a este los parámetros necesarios, hace uso de listas de argumentos de longitud variable, donde la ubicación de los parámetros se corresponde con el orden en que se ingresan los parámetros(por parte del usuario) a este procedimiento, luego este procedimiento adiciona al final de la lista de parámetros del usuario los elementos de la propiedad $this->laFuenteDatosError de esta clase.

Este procedimiento espera el retorno del resultado del procedimiento homólogo en el GEDEGE para emitirlo como resultado de su gestión en forma de retorno.

La explicación exacta de cuantos parámetros tiene cada procedimiento y las características de estos, lo tiene la entidad GEDEGE correspondiente a la instancia de esta clase, ya que este procedimiento toma un grupo de parámetros desconocidos y adiciona a estos otros parámetros conocidos para conformar la lista que pasa o envía al procedimiento homólogo en el GEDEGE, en este procedimiento solo se pueden explicar los parámetros conocidos desde él.

Este procedimiento retorna el resultado(return) del procedimiento homólogo en el GEDEGE .

public function filtrarDatosErrorLog()

este procedimiento hace uso del procedimiento implementarProcedimientosEntidades de esta clase, por lo que su función y objetivo es única pero sus parámetros dependen del funcionamiento del procediendo homólogo en el GEDEGE correspondiente a en la propiedad $GEDEGEDefecto de cada instancia de esta clase. Quien fija o decide la función y objetivo es este procedimiento, quien decide como se gestiona y ejecuta es el procediendo homólogo en el GEDEGE

La función y objetivo de este procedimiento es filtrar un arreglo asociativo con pares $Ls_key=>$Ls_valor aportado por un cliente (arreglo cliente a filtrar), el filtrado se hace comparándolo con un arreglo que es matriz de las $Ls_key($La_matriz_elementos_error['miembros_errorlog']) (arreglo filtro)que ya existe en las propiedades de la instancia(no se tienen que proporcionar como parámetros en este procedimiento), y formando o resultando un arreglo nuevo ($La_datos_error_filtro_result) con los pares $Ls_key=>$Ls_valor cuyos $Ls_key se encontraban en ambos arreglos (el arreglo cliente y el arreglo matriz).

Además también se ponen en el arreglo resultante $La_datos_error_filtro_result del filtrado, los elementos que están en la matriz (arreglo filtro) pero no en el arreglo cliente(arreglo cliente a filtrar), lo anterior se hace poniendo el índice $Ls_key y como valor de este su propio identificador string entre paréntesis('key').Esta funcionalidad depende de La_formato_filtrado['filtrado']['marca_ausente'] elemento que se explica más adelante.

El elemento $Ls_key que exista en el arreglo cliente(arreglo cliente a filtrar) y no exista en la matriz(arreglo filtro), no pasa a formar parte del arreglo resultante $La_datos_error_filtro_result.

Para gestionar correctamente su función y objetivo este procedimiento, al establecer un puente con su homólogo en el GEDEGE y proporcionarle a este los parámetros necesarios, hace uso de listas de argumentos de longitud variable, donde la ubicación de los parámetros están normados de la siguiente forma:

1- se ubican(por parte del usuario) los n parámetros necesarios en los primeros elementos o índices, poniendo entre ellos el parámetro arreglo cliente(arreglo cliente a filtrar) en su lugar correspondiente, llamaremos simbólicamente $La_datos_error_filtro a este parámetro .
2- A continuación de los n parámetros se ubica(por parte del usuario) como último parámetro de este procedimiento el parámetro que llamaremos $La_formato_filtrado donde se definen los datos para el filtrado de paquetes de datos en términos de el funcionamiento del filtrado y la estructura de los paquetes de datos, estos datos hacen referencia a elementos necesarios en el filtrado y/o formato de los datos en la fuente donde estos se encuentren, éstos datos hacen referencia a los datos error y los datos errorlog, la estructura interna de este arreglo depende de la forma que los deba leer el GEDEGE asignado a la instancia de esta clase, ya que $La_formato_filtrado será pasado como argumento al procedimiento del GEDEGE en su gestión; un ejemplo puede ser:

$La_formato_filtrado['formato']['separador'](string) es el separador de datos en una línea de texto
$La_formato_filtrado['filtrado']['marca_ausente'](boolean) si un campo de datos tiene valor vacío, marca el campo con el propio nombre del campo entre paréntesis como valor del campo.

En esta misma clase existe una propiedad $this->laFormatoFiltrado que hace referencia al formato y filtrado de datos de errores, este procedimiento busca la existencia de este parámetro $La_formato_filtrado y de existir adiciona a este los elementos de la propiedad $this->laFormatoFiltrado de esta clase, de no existir este parámetro $La_formato_filtrado el procedimiento lo crea con el valor de $this->laFormatoFiltrado.

3- A continuación del paso 2 este procedimiento adiciona a la lista de argumentos de longitud variable(al final de la lista) la variable $this->laMatrizElementosError, concluyendo con la lista de parámetros y pasándolos al procedimiento homólogo en el GEDEGE.

Después del paso 3 el procedimiento espera el retorno del resultado del procedimiento homólogo en el GEDEGE para emitirlo como resultado de su gestión en forma de retorno.

La explicación exacta de cuantos parámetros tiene cada procedimiento y las características de estos, lo tiene la entidad GEDEGE correspondiente a la instancia de esta clase, ya que este procedimiento toma un grupo de parámetros desconocidos y adiciona a estos otros parámetros conocidos para conformar la lista que pasa o envía al procedimiento homólogo en el GEDEGE, en este procedimiento solo se pueden explicar los parámetros conocidos desde él.

Este procedimiento retorna el resultado(return) del procedimiento homólogo en el GEDEGE .

public function crearDatosErrorLog()

este procedimiento hace uso del procedimiento implementarProcedimientosEntidades de esta clase, por lo que su función y objetivo es única pero sus parámetros dependen del funcionamiento del procediendo homólogo en el GEDEGE correspondiente a en la propiedad $GEDEGEDefecto de cada instancia de esta clase. Quien fija o decide la función y objetivo es este procedimiento, quien decide como se gestiona y ejecuta es el procediendo homólogo en el GEDEGE

La función y objetivo de este procedimiento es crear un nuevo registro de datos errorlog en una fuente de datos erroreslog, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar erroreslog.

Para gestionar correctamente su función y objetivo este procedimiento, al establecer un puente con su homólogo en el GEDEGE y proporcionarle a este los parámetros necesarios, hace uso de listas de argumentos de longitud variable, donde la ubicación de los parámetros están normados de la siguiente forma:

1- Se ubican(por parte del usuario) los n parámetros necesarios en los primeros elementos o índices.
2- A continuación de los n parámetros se ubica(por parte del usuario) como último parámetro de este procedimiento el parámetro que llamaremos $La_formato_filtrado donde se definen los datos para el filtrado de paquetes de datos en términos de el funcionamiento del filtrado y la estructura de los paquetes de datos, estos datos hacen referencia a elementos necesarios en el filtrado y/o formato de los datos en la fuente donde estos se encuentren, éstos datos hacen referencia a los datos error y los datos errorlog, la estructura interna de este arreglo depende de la forma que los deba leer el GEDEGE asignado a la instancia de esta clase, ya que $La_formato_filtrado será pasado como argumento al procedimiento del GEDEGE en su gestión; un ejemplo puede ser:

$La_formato_filtrado['formato']['separador'](string) es el separador de datos en una línea de texto
$La_formato_filtrado['filtrado']['marca_ausente'](boolean) si un campo de datos tiene valor vacio, marca el campo con el propio nombre del campo entre paréntesis como valor del campo.

En esta misma clase existe una propiedad $this->laFormatoFiltrado que hace referencia al formato y filtrado de datos de errores, este procedimiento busca la existencia de este parámetro $La_formato_filtrado y de existir adiciona a este los elementos de la propiedad $this->laFormatoFiltrado de esta clase, de no existir este parámetro $La_formato_filtrado el procedimiento lo crea con el valor de $this->laFormatoFiltrado.

3- A continuación del paso 2 este procedimiento adiciona a la lista de argumentos de longitud variable(al final de la lista) la variable $this->laMatrizElementosError, y $this->laFuenteDatosError, en ese orden, concluyendo con la lista de parámetros y pasándolos al procedimiento homólogo en el GEDEGE.

Después del paso 3 el procedimiento espera el retorno del resultado del procedimiento homólogo en el GEDEGE para emitirlo como resultado de su gestión en forma de retorno.

La explicación exacta de cuantos parámetros tiene cada procedimiento y las características de estos, lo tiene la entidad GEDEGE correspondiente a la instancia de esta clase, ya que este procedimiento toma un grupo de parámetros desconocidos y adiciona a estos otros parámetros conocidos para conformar la lista que pasa o envía al procedimiento homólogo en el GEDEGE, en este procedimiento solo se pueden explicar los parámetros conocidos desde él.

Este procedimiento retorna el resultado(return) del procedimiento homólogo en el GEDEGE .

public function eliminarDatosErrorLog()

este procedimiento hace uso del procedimiento implementarProcedimientosEntidades de esta clase, por lo que su función y objetivo es única pero sus parámetros dependen del funcionamiento del procediendo homólogo en el GEDEGE correspondiente a en la propiedad $GEDEGEDefecto de cada instancia de esta clase. Quien fija o decide la función y objetivo es este procedimiento, quien decide como se gestiona y ejecuta es el procediendo homólogo en el GEDEGE

La función y objetivo de este procedimiento es eliminar registros de datos errorlog en una fuente de datos erroreslog, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar erroreslog.

Para gestionar correctamente su función y objetivo este procedimiento, al establecer un puente con su homólogo en el GEDEGE y proporcionarle a este los parámetros necesarios, hace uso de listas de argumentos de longitud variable, donde la ubicación de los parámetros están normados de la siguiente forma:

1- Se ubican(por parte del usuario) los n parámetros necesarios en los primeros elementos o índices.
2- A continuación de los n parámetros se ubica(por parte del usuario) como último parámetro de este procedimiento el parámetro que llamaremos $La_formato_filtrado donde se definen los datos para el filtrado de paquetes de datos en términos de el funcionamiento del filtrado y la estructura de los paquetes de datos, estos datos hacen referencia a elementos necesarios en el filtrado y/o formato de los datos en la fuente donde estos se encuentren, éstos datos hacen referencia a los datos error y los datos errorlog, la estructura interna de este arreglo depende de la forma que los deba leer el GEDEGE asignado a la instancia de esta clase, ya que

$La_formato_filtrado será pasado como argumento al procedimiento del GEDEGE en su gestión; un ejemplo puede ser:

$La_formato_filtrado['formato']['separador'](string) es el separador de datos en una línea de texto $La_formato_filtrado['filtrado']['llave_unica'] (string), define un dato como condición para que cuando sea encontrado en el registro de datos, se detenga la búsqueda por los restantes registros, es decir se detenga el proceso de filtrado y se pase al termino de la gestión de este procedimiento.

En esta misma clase existe una propiedad $this->laFormatoFiltrado que hace referencia al formato y filtrado de datos de errores, este procedimiento busca la existencia de este parámetro $La_formato_filtrado y de existir adiciona a este los elementos de la propiedad $this->laFormatoFiltrado de esta clase, de no existir este parámetro $La_formato_filtrado el procedimiento lo crea con el valor de $this->laFormatoFiltrado.

3- A continuación del paso 2 este procedimiento adiciona a la lista de argumentos de longitud variable(al final de la lista) la variable $this->laMatrizElementosError, y $this->laFuenteDatosError, en ese orden, concluyendo con la lista de parámetros y pasándolos al procedimiento homólogo en el GEDEGE.

Después del paso 3 el procedimiento espera el retorno del resultado del procedimiento homólogo en el GEDEGE para emitirlo como resultado de su gestión en forma de retorno.

La explicación exacta de cuantos parámetros tiene cada procedimiento y las características de estos, lo tiene la entidad GEDEGE correspondiente a la instancia de esta clase, ya que este procedimiento toma un grupo de parámetros desconocidos y adiciona a estos otros parámetros conocidos para conformar la lista que pasa o envía al procedimiento homólogo en el GEDEGE, en este procedimiento solo se pueden explicar los parámetros conocidos desde él.

Este procedimiento retorna el resultado(return) del procedimiento homólogo en el GEDEGE .

public function leerDatosErrorLog()

este procedimiento hace uso del procedimiento implementarProcedimientosEntidades de esta clase, por lo que su función y objetivo es única pero sus parámetros dependen del funcionamiento del procediendo homólogo en el GEDEGE correspondiente a en la propiedad $GEDEGEDefecto de cada instancia de esta clase. Quien fija o decide la función y objetivo es este procedimiento, quien decide como se gestiona y ejecuta es el procediendo homólogo en el GEDEGE

La función y objetivo de este procedimiento es leer registros de datos errorlog en una fuente de datos erroreslog, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar erroreslog.

Para gestionar correctamente su función y objetivo este procedimiento, al establecer un puente con su homólogo en el GEDEGE y proporcionarle a este los parámetros necesarios, hace uso de listas de argumentos de longitud variable, donde la ubicación de los parámetros están normados de la siguiente forma:

1- Se ubican (por parte del usuario) los n parámetros necesarios en los primeros elementos o índices.
2- A continuación de los n parámetros se ubica (por parte del usuario) como último parámetro de este procedimiento el parámetro que llamaremos $La_formato_filtrado donde se definen los datos para el filtrado de paquetes de datos en términos de el funcionamiento del filtrado y la estructura de los paquetes de datos, estos datos hacen referencia a elementos necesarios en el filtrado y/o formato de los datos en la fuente donde estos se encuentren, éstos datos hacen referencia a los datos error y los datos errorlog, la estructura interna de este arreglo depende de la forma que los deba leer el GEDEGE asignado a la instancia de esta clase, ya que $La_formato_filtrado será pasado como argumento al procedimiento del GEDEGE en su gestión; un ejemplo puede ser:

$La_formato_filtrado['formato']['separador'](string) es el separador de datos en una línea de texto $La_formato_filtrado['filtrado']['llave_unica'] (string), define un dato como condición para que cuando sea encontrado en el registro de datos, se detenga la búsqueda por los restantes registros, es decir se detenga el proceso de filtrado y se pase al termino de la gestión de este procedimiento.

En esta misma clase existe una propiedad $this->laFormatoFiltrado que hace referencia al formato y filtrado de datos de errores, este procedimiento busca la existencia de este parámetro $La_formato_filtrado y de existir adiciona a este los elementos de la propiedad $this->laFormatoFiltrado de esta clase, de no existir este parámetro $La_formato_filtrado el procedimiento lo crea con el valor de $this->laFormatoFiltrado.

3- A continuación del paso 2 este procedimiento adiciona a la lista de argumentos de longitud variable(al final de la lista) la variable $this->laMatrizElementosError, y $this->laFuenteDatosError, en ese orden, concluyendo con la lista de parámetros y pasándolos al procedimiento homólogo en el GEDEGE.

Después del paso 3 el procedimiento espera el retorno del resultado del procedimiento homólogo en el GEDEGE para emitirlo como resultado de su gestión en forma de retorno.

Este procedimiento y sus resultados son parte de la gestión de otros procedimientos de esta clase, por lo que el resultado de su gestión tiene que ser normado para la compatibilidad en el intercambio de estructuras de datos entre procedimientos, para ello se norma que este procedimiento debe retornar la siguiente estructura de datos:
$arreglo[0][nombre_dato1][valor]
$arreglo[0][nombre_dato2][valor]
$arreglo[0][nombre_dato3][valor]

$arreglo[0][nombre_dato4][valor]

ejemplo:

$arreglo[0][‘id_error’][‘100’]
$arreglo[0][‘tipo_error’][256]
$arreglo[0][‘mensaje’][‘El recurso no fue encontrado ’]
$arreglo[0][‘nombre_dato’][valor]

La explicación exacta de cuantos parámetros tiene cada procedimiento y las características de estos, lo tiene la entidad GEDEGE correspondiente a la instancia de esta clase, ya que este procedimiento toma un grupo de parámetros desconocidos y adiciona a estos otros parámetros conocidos para conformar la lista que pasa o envía al procedimiento homólogo en el GEDEGE, en este procedimiento solo se pueden explicar los parámetros conocidos desde él.

Este procedimiento retorna el resultado(return) del procedimiento homólogo en el GEDEGE .

public function modificarDatosErrorLog()

este procedimiento hace uso del procedimiento implementarProcedimientosEntidades de esta clase, por lo que su función y objetivo es única pero sus parámetros dependen del funcionamiento del procediendo homólogo en el GEDEGE correspondiente a en la propiedad $GEDEGEDefecto de cada instancia de esta clase. Quien fija o decide la función y objetivo es este procedimiento, quien decide como se gestiona y ejecuta es el procediendo homólogo en el GEDEGE

La función y objetivo de este procedimiento es modificar registros de datos errorlog en una fuente de datos erroreslog, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar erroreslog.

Para gestionar correctamente su función y objetivo este procedimiento, al establecer un puente con su homólogo en el GEDEGE y proporcionarle a este los parámetros necesarios, hace uso de listas de argumentos de longitud variable, donde la ubicación de los parámetros están normados de la siguiente forma:

1- Se ubican(por parte del usuario) los n parámetros necesarios en los primeros elementos o índices.
2- A continuación de los n parámetros se ubica(por parte del usuario) como último parámetro de este procedimiento el parámetro que llamaremos $La_formato_filtrado donde se definen los datos para el filtrado de paquetes de datos en términos de el funcionamiento del filtrado y la estructura de los paquetes de datos, estos datos hacen referencia a elementos necesarios en el filtrado y/o formato de los datos en la fuente donde estos se encuentren, éstos datos hacen referencia a los datos error y los datos errorlog, la estructura interna de este arreglo depende de la forma que los deba leer el GEDEGE asignado a la instancia de esta clase, ya que

$La_formato_filtrado será pasado como argumento al procedimiento del GEDEGE en su gestión; un ejemplo puede ser:

$La_formato_filtrado['formato']['separador'](string) es el separador de datos en una línea de texto $La_formato_filtrado['filtrado']['llave_unica'] (string), define un dato como condición para que cuando sea encontrado en el registro de datos, se detenga la búsqueda por los restantes registros, es decir se detenga el proceso de filtrado y se pase al termino de la gestión de este procedimiento.

En esta misma clase existe una propiedad $this->laFormatoFiltrado que hace referencia al formato y filtrado de datos de errores, este procedimiento busca la existencia de este parámetro $La_formato_filtrado y de existir adiciona a este los elementos de la propiedad $this->laFormatoFiltrado de esta clase, de no existir este parámetro $La_formato_filtrado el procedimiento lo crea con el valor de $this->laFormatoFiltrado.

3- A continuación del paso 2 este procedimiento adiciona a la lista de argumentos de longitud variable(al final de la lista) la variable $this->laMatrizElementosError, y $this->laFuenteDatosError, en ese orden, concluyendo con la lista de parámetros y pasándolos al procedimiento homólogo en el GEDEGE.

Después del paso 3 el procedimiento espera el retorno del resultado del procedimiento homólogo en el GEDEGE para emitirlo como resultado de su gestión en forma de retorno.

La explicación exacta de cuantos parámetros tiene cada procedimiento y las características de estos, lo tiene la entidad GEDEGE correspondiente a la instancia de esta clase, ya que este procedimiento toma un grupo de parámetros desconocidos y adiciona a estos otros parámetros conocidos para conformar la lista que pasa o envía al procedimiento homólogo en el GEDEGE, en este procedimiento solo se pueden explicar los parámetros conocidos desde él.

Este procedimiento retorna el resultado(return) del procedimiento homólogo en el GEDEGE .

public function crearFuenteDatosErrorLog()

este procedimiento hace uso del procedimiento implementarProcedimientosEntidades de esta clase, por lo que su función y objetivo es única pero sus parámetros dependen del funcionamiento del procediendo homólogo en el GEDEGE correspondiente a en la propiedad $GEDEGEDefecto de cada instancia de esta clase. Quien fija o decide la función y objetivo es este procedimiento, quien decide como se gestiona y ejecuta es el procediendo homólogo en el GEDEGE

La función y objetivo de este procedimiento es crear una fuente de datos erroreslog, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar erroreslog.

Para gestionar correctamente su función y objetivo este procedimiento, al establecer un puente con su homólogo en el GEDEGE y proporcionarle a este los parámetros necesarios, hace uso de listas

de argumentos de longitud variable, donde la ubicación de los parámetros se corresponde con el orden en que se ingresan los parámetros a este procedimiento.

Este procedimiento espera el retorno del resultado del procedimiento homólogo en el GEDEGE para emitirlo como resultado de su gestión en forma de retorno.

La explicación exacta de cuantos parámetros tiene cada procedimiento y las características de estos, lo tiene la entidad GEDEGE correspondiente a la instancia de esta clase, ya que este procedimiento toma un grupo de parámetros desconocidos y adiciona a estos otros parámetros conocidos para conformar la lista que pasa o envía al procedimiento homólogo en el GEDEGE, en este procedimiento solo se pueden explicar los parámetros conocidos desde él.

Este procedimiento retorna el resultado(return) del procedimiento homólogo en el GEDEGE .

public function eliminarFuenteDatosErrorLog()

este procedimiento hace uso del procedimiento implementarProcedimientosEntidades de esta clase, por lo que su función y objetivo es única pero sus parámetros dependen del funcionamiento del procediendo homólogo en el GEDEGE correspondiente a en la propiedad $GEDEGEDefecto de cada instancia de esta clase. Quien fija o decide la función y objetivo es este procedimiento, quien decide como se gestiona y ejecuta es el procediendo homólogo en el GEDEGE

La función y objetivo de este procedimiento es eliminar una fuente de datos erroreslog, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar erroreslog.

Para gestionar correctamente su función y objetivo este procedimiento, al establecer un puente con su homólogo en el GEDEGE y proporcionarle a este los parámetros necesarios, hace uso de listas de argumentos de longitud variable, donde la ubicación de los parámetros se corresponde con el orden en que se ingresan los parámetros(por parte del usuario) a este procedimiento, luego este procedimiento adiciona al final de la lista de parámetros del usuario los elementos de la propiedad $this->laFuenteDatosError de esta clase.

Este procedimiento espera el retorno del resultado del procedimiento homólogo en el GEDEGE para emitirlo como resultado de su gestión en forma de retorno.

La explicación exacta de cuantos parámetros tiene cada procedimiento y las características de estos, lo tiene la entidad GEDEGE correspondiente a la instancia de esta clase, ya que este procedimiento toma un grupo de parámetros desconocidos y adiciona a estos otros parámetros conocidos para conformar la lista que pasa o envía al procedimiento homólogo en el GEDEGE, en este procedimiento solo se pueden explicar los parámetros conocidos desde él.

Este procedimiento retorna el resultado(return) del procedimiento homólogo en el GEDEGE .

public function error ($Ls_id_error , $La_formato_mensaje_error = null , $La_formato_mensaje_errorlog = null , $La_formato_filtrado = null)

este procedimiento es el encargado de hacer toda la gestión correspondiente a la emisión de un error y su registro en un errorlog partiendo del filtrado del error en una fuente, formateado del mensaje de error y el mensaje del errorlog, contando para ello con los valores que se guardan en las propiedades de cada instancia de esta clase.
Cada instancia de esta clase define dentro de sus propiedades los datos necesarios para que se lleve a cabo una gestión de un error determinado de forma automática y preconcebida, con solo ingresar en este procedimiento un identificador de error, este procedimiento busca este identificador en una fuente de datos de errores, lo extrae junto con otras partes del error como pueden ser mensaje, tipo de error etc. Para después formatearlo, emitirlo y guardar un registro de este error en un destino errorlog que también cuenta con su propio formato como puede ser id del error, mensaje, tipo de error, fecha, hora, fichero, línea etc. Haciendo más fácil, ordenado y reutilizable el uso y tratamiento de errores.

Este procedimiento hace uso de trigger_error() combinado con un procedimiento como gestor de errores declarado en
$this->lsFuncionGestorErrores, y para la gestión de datos de error en la fuente de datos utiliza $this->leerDatosError, como también para la gestión de datos de errorlog en el destino de datos utiliza
$this->crearDatosErrorLog

Este procedimiento para su gestión depende de la estructura de datos aportados por el procedimiento $this->leerDatosError, para el correcto funcionamiento de este procedimiento es obligatorio que el procedimiento $this->leerDatosError devuelva un arreglo con la siguiente estructura:
arreglo[0][nombre_dato1][valor]
arreglo[0][nombre_dato2][valor]
arreglo[0][nombre_dato3][valor]
arreglo[0][nombre_dato4][valor]

ejemplo:

arreglo[0][‘id_error’][‘100’]
arreglo[0][‘tipo_error’][256]
arreglo[0][‘mensaje’][‘El recurso no fue encontrado ’]
arreglo[0][‘nombre_dato’][valor]

A continuación se explican los parámetros de este procedimiento.

$Ls_id_error(string) identificador del error a gestionar, por ejemplo puede ser el número del error,

que en un registro de bases de datos sería un campo llave ,
id_error(llave)
tipo_error
mensaje

y en una línea de un txt sería un valor en una sección de la línea que a su vez es irrepetible en la misma sección de las demás líneas del txt
id_error:: tipo_error:: mensaje

$La_formato_mensaje_error (array) donde se definen los elementos que conformarán el mensaje de error, es el grupo de elementos que se concatenaran para formar el mensaje de error emitido por este procedimiento, estos elementos conformarán el mensaje en el mismo orden en que aparecen los elementos, cada elemento de este arreglo será tenido en cuenta como llave del arreglo asociativo fuente para los datos que conformarán el mensaje de error. Un ejemplo puede ser array('id' , 'tipo' , 'mensaje').

Este parámetro es para que el código cliente tenga la posibilidad de utilizar un formato personalizado, ya que en caso de tener valor vacío este parámetro entonces este procedimiento trabaja con el formato declarado en
$this->laFormatoMensajeError

Recordar que los datos del error ya vienen filtrados con anterioridad por los elementos de $this->laMatrizElementosError por lo que los elementos que formen parte de este arreglo $La_formato_mensaje_error para ser mostrados tienen que estar en la lista de elementos de $this->laMatrizElementosError, ya que si no coinciden en ambos arreglos no se tendrán en cuenta para formar parte del mensaje de error.

$La_formato_mensaje_errorlog (array) donde se definen los elementos que conformarán el mensaje de errorlog, es el grupo de elementos que se concatenaran para formar el mensaje de errorlog emitido por este procedimiento, estos elementos conformarán el mensaje en el mismo orden en que aparecen los elementos, cada elemento de este arreglo será tenido en cuenta como llave del arreglo asociativo fuente para los datos que conformarán el mensaje de errorlog. Un ejemplo puede ser array('id' , 'tipo' , 'mensaje' , 'tipo' , 'tiempo' , 'fichero' , 'línea', 'proceso').

Este parámetro es para que el código cliente tenga la posibilidad de utilizar un formato personalizado, ya que en caso de tener valor vacío este parámetro entonces este procedimiento trabaja con el formato declarado en
$this->laFormatoMensajeErrorLog

En el caso específico del mensaje que se elabora para errorlog este procedimiento admite de forma opcional en la estructura $La_formato_mensaje_errorlog o $this->laFormatoMensajeErrorLog los

elementos 'tiempo' , 'fichero' , 'línea', 'proceso', 'gedee_proceso', para su incorporación al registro de errorlog, a continuación explicamos cada elemento

'tiempo' recoge el valor de time() en el momento en que se gestiona el error.
'fichero' recoge el valor del path del fichero donde se genera el error.
'línea' recoge el valor de la línea del fichero donde se genera el error.
'proceso' recoge el valor del proceso en ejecución donde se genera el error.
'gedee_proceso' recoge el valor del gedee del proceso en ejecución donde se genera el error.

Recordar que los datos del error ya vienen filtrados con anterioridad por los elementos de $this->laMatrizElementosError por lo que los elementos que formen parte de este arreglo $La_formato_mensaje_errorlog para ser mostrados tienen que estar en la lista de elementos de $this->laMatrizElementosError, ya que si no coinciden en ambos arreglos no se tendrán en cuenta para formar parte del registro de errorlog.

$La_formato_filtrado (array), donde se definen los datos para el filtrado de paquetes de datos en términos de el funcionamiento del filtrado y la estructura de los paquetes de datos, estos datos hacen referencia a elementos necesarios en el filtrado y/o formato de los datos en la fuente donde estos se encuentren, éstos datos hacen referencia a los datos error y los datos errorlog, la estructura interna de este arreglo depende de la forma que los deba leer el GEDEGE asignado a la instancia de esta clase, ya que $laFormatoFiltrado será pasado como argumento a los procedimientos del GEDEGE en su gestión; un ejemplo puede ser:

$La_formato_filtrado ['formato']['separador'](string) es el separador de datos en una línea de texto en un txt
$La_formato_filtrado ['filtrado']['marca_ausente'](boolean) si un campo de datos tiene valor vacio, marca el campo con el propio nombre del campo entre paréntesis como valor del campo.
$La_formato_filtrado ['filtrado']['llave_unica'](string) define una llave como condición para que cuando esta llave y su valor sean encontrados en la línea del txt, se detenga la búsqueda por las líneas restantes del fichero, es decir se detenga el proceso de filtrado y se pase a al termino de la gestión de un procedimiento determinado.

En esta misma clase existe una propiedad $this->laFormatoFiltrado que hace referencia al formato y filtrado de datos de errores, este procedimiento busca la existencia de este parámetro $La_formato_filtrado y de existir adiciona a este los elementos de la propiedad $this->laFormatoFiltrado de esta clase, de no existir este parámetro $La_formato_filtrado el procedimiento lo crea con el valor de $this->laFormatoFiltrado.

Este parámetro es para que el código cliente tenga la posibilidad de utilizar un formato y/o filtrado personalizado, ya que en caso de tener valor vacío este parámetro entonces este procedimiento trabaja con el formato declarado en
$this->laFormatoMensajeError

Este procedimiento depende del retorno del procedimiento $this->leerDatosError, y es obligatorio que $this->leerDatosError retorne un arreglo multidimensional donde el primer nivel de llaves o índices es de tipo numérico automático y en cada uno de estos elementos (solo tiene un nivel porque es de llave única la búsqueda) del primer nivel de índices si existirá un arreglo asociativo(con índices o llaves de tipo string) donde cada uno de sus elementos contendrá como valor un string correspondiente con las secciones o propiedades que contenían los datos error filtrados y leídos, cada arreglo contenido en los índices numéricos representa los datos o propiedades de un error (línea txt , fila tabla , registro tabla-bd) que sus datos (seccion txt , celda tabla , campo tabla-bd) coincidieron con el filtrado.

Justamente después de emitir el mensaje de error este procedimiento retorna el control de errores al manejador anterior a su implementación, es decir cuando este procedimiento es invocado implanta su manejador de errores y luego devuelve el control al manejador de errores anterior.

Si este procedimiento realiza la gestión del error junto a la emisión del mensaje y la ceración del registro de errorlog correctamente retorna el valor true de lo contrario retorna el valor null.

class_gedege_txt_emod

Este fichero contiene la clase GedegeTxtEmod. Esta clase está compuesta por propiedades, y procedimientos para la gestión de datos de errores directamente con una fuente de datos de tipo fichero txt. Es quien realiza la gestión de filtrar, leer, eliminar, modificar etc. datos en un fichero txt fuentes de datos de errores, como también crear y eliminar fichero txt fuentes de datos de errores. Es en esencia un GEDEGE Gestor de Estructuras de Datos para Entidades Gestoras de Errores, y su función es gestionar datos como intermediario entre la entidad que hará el tratamiento del error(gestor de errores) y la fuente de los datos de los errores.

A continuación mostramos una lista de las propiedades y procedimientos que esta clase pone a disposición de los códigos clientes del Sockeletom:

Namespace: Emod\Nucleo\Errores\Gedege
Nombre de la clase: GedegeTxtEmod
Herencia:

Propiedades:

private $iniciacion = null ;
boolean, propiedad para chequear la iniciación de las instancias de esta clase.

private $EDatosFormatoTxt = null ;
string, esta clase trabaja directamente con una clase herramienta que hace su gestión sobre ficheros txt donde se encuentran formateados los datos de errores, y esta propiedad es la que va a recibir el

string namespace/clase de la entidad herramienta que en este caso es EDatosFormatoTxt, una herramienta para trabajar contenidos de ficheros txt y que es una clase con procedimientos estáticos, por ello el string namespace/clase para poder acceder a ella.

Procedimientos:

__construct ()
Constructor, donde se verifica la existencia de la herramienta EDatosFormatoTxt en la clase Emod\Nucleo\Herramientas, herramienta fundamental para la gestión de esta clase, de encontrarse accesible la herramienta EDatosFormatoTxt se asigna una referencia a la propiedad $this->EDatosFormatoTxt y se retorna el valor true, de lo contrario se lanza una excepción con la siguiente estructura:

throw new \Exception('Error de instanciacion,'.__METHOD__.' no se encuentra la herramienta de clase base \Emod\Nucleo\Herramientas\EDatosFormatoTxt.');

public function filtrarDatosError($La_datos_error_filtro , $La_formato_filtrado , $La_matriz_elementos_error , $tipo_error = 2)

La función y objetivo de este procedimiento es filtrar un arreglo asociativo con pares $Ls_key=>$Ls_valor aportado por un cliente (arreglo cliente a filtrar), el filtrado se hace comparándolo con un arreglo que es matriz de las $Ls_key ($La_matriz_elementos_error['miembros_error'] o $La_matriz_elementos_error['miembros_errorlog']) (arreglo filtro), y formando o resultando un arreglo nuevo ($La_datos_error_filtro_result) con los pares $Ls_key=>$Ls_valor cuyos $Ls_key se encontraban en ambos arreglos (el arreglo cliente y el arreglo matriz).

Además también se ponen en el arreglo resultante $La_datos_error_filtro_result del filtrado, los elementos que están en la matriz (arreglo filtro) pero no en el arreglo cliente(arreglo cliente a filtrar), lo anterior se hace poniendo el índice $Ls_key y como valor de este su propio identificador string entre paréntesis('key').Esta funcionalidad depende de La_formato_filtrado['filtrado']['marca_ausente'] elemento que se explica más adelante.

El elemento $Ls_key que exista en el arreglo cliente(arreglo cliente a filtrar) y no exista en la matriz(arreglo filtro), no pasa a formar parte del arreglo resultante $La_datos_error_filtro_result.

Los parámetros se explican a continuación:

$La_datos_error_filtro (array) es un arreglo con los pares $Ls_key=>$Ls_valor a filtrar.

$La_formato_filtrado (array) es un arreglo asociativo en el que este procedimiento buscará la estructura ['filtrado']['marca_ausente'], es decir $La_formato_filtrado['filtrado']['marca_ausente'] , si existe esta estructura y tiene valor true define que, si existe un elemento en $La_matriz_elementos_error y no existe ese elemento como $Ls_key en $La_datos_error_filtro se incorpora al arreglo resultante del filtrado, la llave $Ls_key y como su valor entre paréntesis el propio key quedando de la siguiente forma $Ls_key = (‘Ls_key’).Esta estructura es opcional.

$La_matriz_elementos_error (array) es un arreglo asociativo cuyo contenido se toma como referencia para el filtrado del arreglo $La_datos_error_filtro, este arreglo hace de mascara de los $Ls_key(solo de las llaves $Ls_key) para el filtrado del arreglo $La_datos_error_filtro. Su estructura es la siguiente:

$La_matriz_elementos_error ['miembros_errorlog']
propiedad que define los campos o elementos que se gestionaran para un contenedor de registros de errores, a esta propiedad siempre se le adicionan los valores del elemento ['miembros_error']
un ejemplo puede ser
array('date' , 'aplicacion' , 'modulo' , 'opcion')

$La_matriz_elementos_error ['miembros_error']
propiedad que define los campos o elementos que se gestionaran para un mensaje u otro destino de errores un ejemplo puede ser
array('id' , 'tipo' , 'tratamiento' , 'alcance' , 'mensaje')

$tipo_error (integer), define el arreglo matriz de keys por los que se filtraran los keyz del arreglo $La_datos_error_filtro sus posibles valores son:
1- se tomara como arreglo matriz el definido en $La_matriz_elementos_error['miembros_errorlog'].
2- se tomara como arreglo matriz el definido en $La_matriz_elementos_error['miembros_error']

el orden de los elementos key del arreglo resultante $La_datos_error_filtro_result los define el orden de los elementos del arreglo $La_matriz_elementos_error.

Este procedimiento retorna el arreglo resultante con los elementos $Ls_key=>$Ls_valor cuyos $Ls_key de $La_datos_error_filtro existían como elementos en $La_matriz_elementos_error si su gestión fue satisfactoria, de lo contrario retorna el valor null. Si el arreglo resultante del filtrado es vacío también se retorna el valor null.

public function crearDatosError($La_datos_error_filtro , $La_formato_filtrado , $La_matriz_elementos_error , $La_fuente_datos_error , $tipo_error = 2)

La función y objetivo de este procedimiento es crear un nuevo registro de datos error en una fuente de datos errores, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar errores. En este caso es crear una nueva línea estructurada con datos en un fichero txt que hace de fuente de datos errores.

Este procedimiento utiliza en su gestión el procedimiento $this->filtrarDatosError por lo que coinciden en gran parte los parámetros de este procedimiento con los del procedimiento $this->filtrarDatosError

Los parámetros se explican a continuación:

$La_datos_error_filtro (array) es un arreglo con los pares $Ls_key=>$Ls_valor a filtrar, son los pares que el cliente quiere crear como un nuevo registro de datos de error en la fuente de datos errores. De este parámetro se utilizan los $Ls_key en el filtrado y solo pasan como datos de la estructura de la línea del txt los $Ls_valor.

$La_formato_filtrado (array) es un arreglo asociativo en el que se encuentran datos relacionados con el filtrado y el formato de datos errores, este procedimiento utiliza los siguientes:

1- este procedimiento buscará la estructura ['filtrado']['marca_ausente'], es decir $La_formato_filtrado['filtrado']['marca_ausente'] , si existe esta estructura y tiene valor true define que, si existe un elemento en $La_matriz_elementos_error y no existe ese elemento como $Ls_key en $La_datos_error_filtro se incorpora al arreglo resultante del filtrado, la llave $Ls_key y como su valor entre paréntesis el propio key quedando de la siguiente forma $Ls_key = (‘Ls_key’). Esta estructura es opcional.

2- este procedimiento buscará la estructura ['formato']['separador'], es decir $La_formato_filtrado['formato']['separador'], esta estructura debe contener un string que se utilizará como separador de los elementos datos que conformarán la línea de datos de error que se guardará en el fichero txt , que hace de fuente de datos errores. Esta estructura es obligatoria.

$La_matriz_elementos_error (array) es un arreglo asociativo cuyo contenido se toma como referencia para el filtrado del arreglo $La_datos_error_filtro, este arreglo hace de mascara de los $Ls_key(solo de las llaves $Ls_key) para el filtrado del arreglo $La_datos_error_filtro. Su estructura es la siguiente:

$La_matriz_elementos_error ['miembros_errorlog']
propiedad que define los campos o elementos que se gestionaran para un contenedor de registros de errores, a esta propiedad siempre se le adicionan los valores del elemento ['miembros_error']
un ejemplo puede ser
array('date' , 'aplicacion' , 'modulo' , 'opcion')

$La_matriz_elementos_error ['miembros_error']

propiedad que define los campos o elementos que se gestionaran para un mensaje u otro destino de errores un ejemplo puede ser
array('id' , 'tipo' , 'tratamiento' , 'alcance' , 'mensaje')

$La_fuente_datos_error (array) es quien contendrá los datos necesarios para acceder a la fuente de datos errores, en este caso el fichero txt , que hace de fuente de datos errores. Este procedimiento buscará la estructura ['path_fich_error'], es decir $La_fuente_datos_error['path_fich_error'], esta estructura debe contener un string que se utilizará como path donde se encuentra el fichero a incorporar la nueva línea de datos de error. Esta estructura es obligatoria.

$tipo_error (integer), define el arreglo matriz de keys por los que se filtraran los keyz del arreglo $La_datos_error_filtro sus posibles valores son:
1- se tomara como arreglo matriz el definido en $La_matriz_elementos_error['miembros_errorlog'].
2- se tomara como arreglo matriz el definido en $La_matriz_elementos_error['miembros_error']

Este procedimiento retorna el valor true si se lleva a cabo satisfactoriamente su gestión, de lo contrario retorna el valor null. Si la gestión es satisfactoria y no existe el fichero en $La_fuente_datos_error['path_fich_error'], se intentará crear el fichero txt con la estructura gestionada en el path $La_fuente_datos_error['path_fich_error'].

public function eliminarDatosError($La_datos_error_filtro , $La_formato_filtrado , $La_matriz_elementos_error , $La_fuente_datos_error , $tipo_error = 2)

La función y objetivo de este procedimiento es eliminar registros de datos error en una fuente de datos errores, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar errores. En este caso es eliminar una línea, estructurada con datos en un fichero txt que hace de fuente de datos errores, se elimina si esta línea en su estructura de datos errores tiene datos coincidentes con el criterio de eliminación que viene dado por el arreglo $La_datos_error_filtro.

Los parámetros se explican a continuación:

$La_datos_error_filtro (array) es un arreglo con los pares $Ls_key=>$Ls_valor a filtrar, son los pares que el cliente pone como condición para eliminar líneas de datos del txt, registros de datos de error en la fuente de datos errores. Son comparados los pares de este parámetro con los pares formados con $La_matriz_elementos_error(como key_txt) y el dato en la línea del txt (como valor_txt), y si coinciden todos los pares entonces se elimina la línea del txt.

$La_formato_filtrado (array) es un arreglo asociativo en el que se encuentran datos relacionados con el filtrado y el formato de datos errores, este procedimiento utiliza los siguientes:

1- este procedimiento buscará la estructura ['filtrado']['marca_ausente'], es decir $La_formato_filtrado['filtrado']['marca_ausente'] , si existe esta estructura y tiene valor true define que, si existe un elemento en $La_matriz_elementos_error y no existe ese elemento como $Ls_key en $La_datos_error_filtro se incorpora al arreglo resultante del filtrado, la llave $Ls_key y como su valor entre paréntesis el propio key quedando de la siguiente forma $Ls_key = (‘Ls_key’). Esta estructura es opcional.
2- este procedimiento buscará la estructura ['formato']['separador'], es decir $La_formato_filtrado['formato']['separador'], esta estructura debe contener un string que se utilizará como separador de los elementos datos que conformarán la línea de datos de error que se buscarán en el fichero txt , que hace de fuente de datos errores. Esta estructura es obligatoria.
3- este procedimiento buscará la estructura ['filtrado']['llave_unica'], es decir $La_formato_filtrado ['filtrado']['llave_unica'], esta estructura admite un string igual al de un $Ls_key de $La_datos_error_filtro, de existir este string cuando se hace la comparación de datos para buscar coincidencias, si un par $Ls_key=>$Ls_valor coincide con los datos del txt y a la vez este $Ls_key es también 'llave_unica', entonces se detiene el proceso de filtrado , también se detiene el proceso de búsqueda por las demás líneas del txt y se elimina la línea con la coincidencia. Esta estructura es opcional.

$La_matriz_elementos_error (array) es un arreglo asociativo cuyo contenido se toma como referencia para el filtrado de las líneas del txt, este arreglo hace de complemento $Ls_key(solo de las llaves $Ls_key) para el casamiento con los datos en la línea del txt que figurarán como valor de las $Ls_key. Su estructura es la siguiente:

$La_matriz_elementos_error ['miembros_errorlog']
propiedad que define los campos o elementos que se gestionaran para un contenedor de registros de errores, a esta propiedad siempre se le adicionan los valores del elemento ['miembros_error']
un ejemplo puede ser
array('date' , 'aplicacion' , 'modulo' , 'opcion')

$La_matriz_elementos_error ['miembros_error']
propiedad que define los campos o elementos que se gestionaran para un mensaje u otro destino de errores un ejemplo puede ser
array('id' , 'tipo' , 'tratamiento' , 'alcance' , 'mensaje')

$La_fuente_datos_error (array) es quien contendrá los datos necesarios para acceder a la fuente de datos errores, en este caso el fichero txt , que hace de fuente de datos errores. Este procedimiento buscará la estructura ['path_fich_error'], es decir $La_fuente_datos_error['path_fich_error'], esta estructura debe contener un string que se utilizará como path donde se encuentra el fichero a filtrar y eliminar las líneas de datos de error coincidentes con el criterio de eliminación que viene dado por el arreglo $La_datos_error_filtro. Esta estructura es obligatoria.

$tipo_error (integer), define el arreglo matriz de keys que comprenderá el parámetro $La_matriz_elementos_error, sus posibles valores son:

1- se tomara como arreglo matriz el definido en $La_matriz_elementos_error['miembros_errorlog'].
2- se tomara como arreglo matriz el definido en $La_matriz_elementos_error['miembros_error']

Este procedimiento retorna:
retorna true si encuentra elemento al que aplicar la condición de filtrado y lo realiza satisfactoriamente.
retorna false si encuentra elemento al que aplicar la condición de filtrado y no le es posible realizar los cambios en el fichero.
retorna null si no encuentra elemento al que aplicar la condición de filtrado o se le pasan valores incompatibles a los parámetros del procedimiento.

public function leerDatosError($La_datos_error_filtro , $La_formato_filtrado , $La_matriz_elementos_error , $La_fuente_datos_error , $tipo_error = 2)

La función y objetivo de este procedimiento es leer registros de datos error en una fuente de datos errores, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar errores. En este caso es leer una línea, estructurada con datos en un fichero txt que hace de fuente de datos errores, se lee si esta línea en su estructura de datos errores tiene datos coincidentes con el criterio de lectura que viene dado por el arreglo $La_datos_error_filtro.

Los parámetros se explican a continuación:

$La_datos_error_filtro (array) es un arreglo con los pares $Ls_key=>$Ls_valor a filtrar, son los pares que el cliente pone como condición para leer líneas de datos del txt, registros de datos de error en la fuente de datos errores. Son comparados los pares de este parámetro con los pares formados con $La_matriz_elementos_error(como key_txt) y el dato en la línea del txt (como valor_txt), y si coinciden todos los pares entonces se lee(toma) la línea del txt.

$La_formato_filtrado (array) es un arreglo asociativo en el que se encuentran datos relacionados con el filtrado y el formato de datos errores, este procedimiento utiliza los siguientes:

1- este procedimiento buscará la estructura ['formato']['separador'], es decir $La_formato_filtrado['formato']['separador'], esta estructura debe contener un string que se utilizará como separador de los elementos datos que conformarán la línea de datos de error que se buscarán en el fichero txt , que hace de fuente de datos errores. Esta estructura es obligatoria.
2- este procedimiento buscará la estructura ['filtrado']['llave_unica'], es decir $La_formato_filtrado ['filtrado']['llave_unica'], esta estructura admite un string igual al de un $Ls_key de $La_datos_error_filtro, de existir este string cuando se hace la comparación de datos para buscar coincidencias, si un par $Ls_key=>$Ls_valor coincide con los datos del txt y a la vez

este $Ls_key es también 'llave_unica', entonces se detiene el proceso de filtrado , también se detiene el proceso de búsqueda por las demás líneas del txt y se lee la línea con la coincidencia. Esta estructura es opcional.

$La_matriz_elementos_error (array) es un arreglo asociativo cuyo contenido se toma como referencia para el filtrado de las líneas del txt, este arreglo hace de complemento $Ls_key(solo de las llaves $Ls_key) para el casamiento con los datos en la línea del txt que figurarán como valor de las $Ls_key. Su estructura es la siguiente:

$La_matriz_elementos_error ['miembros_errorlog']
propiedad que define los campos o elementos que se gestionaran para un contenedor de registros de errores, a esta propiedad siempre se le adicionan los valores del elemento ['miembros_error']
un ejemplo puede ser
array('date' , 'aplicacion' , 'modulo' , 'opcion')

$La_matriz_elementos_error ['miembros_error']
propiedad que define los campos o elementos que se gestionaran para un mensaje u otro destino de errores un ejemplo puede ser
array('id' , 'tipo' , 'tratamiento' , 'alcance' , 'mensaje')

$La_fuente_datos_error (array) es quien contendrá los datos necesarios para acceder a la fuente de datos errores, en este caso el fichero txt , que hace de fuente de datos errores. Este procedimiento buscará la estructura ['path_fich_error'], es decir $La_fuente_datos_error['path_fich_error'], esta estructura debe contener un string que se utilizará como path donde se encuentra el fichero a filtrar y leer las líneas de datos de error coincidentes con el criterio de lectura que viene dado por el arreglo $La_datos_error_filtro. Esta estructura es obligatoria.

$tipo_error (integer), define el arreglo matriz de keys que comprenderá el parámetro $La_matriz_elementos_error, sus posibles valores son:
1- se tomara como arreglo matriz el definido en $La_matriz_elementos_error['miembros_errorlog'].
2- se tomara como arreglo matriz el definido en $La_matriz_elementos_error['miembros_error']

Este procedimiento retorna:
retorna arreglo multidimensional donde el primer nivel de llaves o índices es de tipo numérico automático y en cada uno de estos elementos existirá un arreglo asociativo(con índices o llaves de tipo string) donde cada uno de sus elementos contendrá como valor un string correspondiente con las secciones o propiedades que contenían las líneas del txt que coincidieron con el filtrado, cada arreglo contenido en los índices numéricos representa los datos de una línea del txt que coincidieron con el filtrado.

retorna null si no encuentra elemento al que aplicar la condición de filtrado o se le pasan valores incompatibles a los parámetros del procedimiento.

public function modificarDatosError($La_datos_error_filtro , $La_datos_sustitutos , $La_formato_filtrado , $La_matriz_elementos_error , $La_fuente_datos_error , $tipo_error = 2)

La función y objetivo de este procedimiento es modificar registros de datos error en una fuente de datos errores, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar errores. En este caso es modificar una línea, estructurada con datos en un fichero txt que hace de fuente de datos errores, se modifica si esta línea en su estructura de datos errores tiene datos coincidentes con el criterio de modificación que viene dado por el arreglo $La_datos_error_filtro.

Los parámetros se explican a continuación:

$La_datos_error_filtro (array) es un arreglo con los pares $Ls_key=>$Ls_valor a filtrar, son los pares que el cliente pone como condición para modificar líneas de datos del txt, registros de datos de error en la fuente de datos errores. Son comparados los pares de este parámetro con los pares formados con $La_matriz_elementos_error(como key_txt) y el dato en la línea del txt (como valor_txt), y si coinciden todos los pares entonces se modifica la línea del txt con el dato correspondiente (de igual key) en el arreglo $La_datos_sustitutos.
En caso de querer modificar valores en todas las líneas del fichero, este parámetro soporta el valor '*', que le indica al procedimiento que realice la sustitución en todas las líneas del fichero sin hacer filtrado.

$La_datos_sustitutos (array) es un arreglo con los pares $Ls_key=>$Ls_valor a modificar en la línea del txt, no tienen que coincidir con los pares de $La_datos_error_filtro.

$La_formato_filtrado (array) es un arreglo asociativo en el que se encuentran datos relacionados con el filtrado y el formato de datos errores, este procedimiento utiliza los siguientes:

1- este procedimiento buscará la estructura ['formato']['separador'], es decir $La_formato_filtrado['formato']['separador'], esta estructura debe contener un string que se utilizará como separador de los elementos datos que conformarán la línea de datos de error que se buscarán en el fichero txt , que hace de fuente de datos errores. Esta estructura es obligatoria.
2- este procedimiento buscará la estructura ['filtrado']['llave_unica'], es decir $La_formato_filtrado ['filtrado']['llave_unica'], esta estructura admite un string igual al de un $Ls_key de $La_datos_error_filtro, de existir este string cuando se hace la comparación de datos para buscar coincidencias, si un par $Ls_key=>$Ls_valor coincide con los datos del txt y a la vez este $Ls_key es también 'llave_unica', entonces se detiene el proceso de filtrado , también se detiene el proceso de búsqueda por las demás líneas del txt y se modifica la línea con la sustitución. Esta estructura es opcional.

$La_matriz_elementos_error (array) es un arreglo asociativo cuyo contenido se toma como referencia para el filtrado de las líneas del txt, este arreglo hace de complemento $Ls_key(solo de las llaves $Ls_key) para el casamiento con los datos en la línea del txt que figurarán como valor de las $Ls_key. Su estructura es la siguiente:

$La_matriz_elementos_error ['miembros_errorlog']
 propiedad que define los campos o elementos que se gestionaran para un contenedor de registros de errores, a esta propiedad siempre se le adicionan los valores del elemento ['miembros_error']
 un ejemplo puede ser
 array('date' , 'aplicacion' , 'modulo' , 'opcion')

$La_matriz_elementos_error ['miembros_error']
propiedad que define los campos o elementos que se gestionaran para un mensaje u otro destino de errores un ejemplo puede ser
array('id' , 'tipo' , 'tratamiento' , 'alcance' , 'mensaje')

$La_fuente_datos_error (array) es quien contendrá los datos necesarios para acceder a la fuente de datos errores, en este caso el fichero txt , que hace de fuente de datos errores. Este procedimiento buscará la estructura ['path_fich_error'], es decir $La_fuente_datos_error['path_fich_error'], esta estructura debe contener un string que se utilizará como path donde se encuentra el fichero a filtrar y modificar las líneas de datos de error coincidentes con el criterio de modificación que viene dado por el arreglo $La_datos_error_filtro. Esta estructura es obligatoria.

$tipo_error (integer), define el arreglo matriz de keys que comprenderá el parámetro $La_matriz_elementos_error, sus posibles valores son:
1- se tomara como arreglo matriz el definido en $La_matriz_elementos_error['miembros_errorlog'].
2- se tomara como arreglo matriz el definido en $La_matriz_elementos_error['miembros_error']

Este procedimiento retorna:
retorna true si encuentra elemento al que aplicar la condición de filtrado y lo realiza satisfactoriamente.
retorna false si encuentra elemento al que aplicar la condición de filtrado y no le es posible realizar los cambios en el fichero.
retorna null si no encuentra elemento al que aplicar la condición de filtrado o se le pasan valores incompatibles a los parámetros del procedimiento.

```
public function crearFuenteDatosError( $La_fuente_datos_error , $Ls_cadena_inicio = '' )
```

La función y objetivo de este procedimiento es crear una fuente de datos errores, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar errores. En este caso es crear un fichero txt donde cada línea del fichero es un registro de datos estructurados, como fuente de datos de errores.

Los parámetros se explican a continuación:

$La_fuente_datos_error (array) es quien contendrá los datos necesarios para crear la fuente de datos errores, en este caso el fichero txt , que hace de fuente de datos errores. Este procedimiento buscará la estructura ['path_fich_error'], es decir $La_fuente_datos_error['path_fich_error'], esta estructura debe contener un string que se utilizará como path donde se creará el fichero txt.

$Ls_cadena_inicio (string) cadena a escribir en el fichero txt, cadena que conforma el cuerpo del fichero de datos de errores, este procedimiento trabaja con la función file_put_contents de php.

Este procedimiento retorna true si su gestión fue satisfactoria y null si fue insatisfactoria.

public function eliminarFuenteDatosError($La_fuente_datos_error)

La función y objetivo de este procedimiento es eliminar una fuente de datos errores, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar errores. En este caso es eliminar un fichero txt donde cada línea del fichero es un registro de datos estructurados, como fuente de datos de errores.

Los parámetros se explican a continuación:

$La_fuente_datos_error (array) es quien contendrá los datos necesarios para eliminar la fuente de datos errores, en este caso el fichero txt , que hace de fuente de datos errores. Este procedimiento buscará la estructura ['path_fich_error'], es decir $La_fuente_datos_error['path_fich_error'], esta estructura debe contener un string que se utilizará como path del fichero txt a eliminar.

Este procedimiento retorna true si su gestión fue satisfactoria y null si fue insatisfactoria.

public function filtrarDatosErrorlog($La_datos_errorlog_filtro , $La_formato_filtrado , $La_matriz_elementos_errorlog)

La función y objetivo de este procedimiento es filtrar un arreglo asociativo con pares $Ls_key=>$Ls_valor aportado por un cliente (arreglo cliente a filtrar), el filtrado se hace comparándolo con un arreglo que es matriz de las $Ls_key
($La_matriz_elementos_error['miembros_error'] y
$La_matriz_elementos_error['miembros_errorlog']) (arreglo filtro), y formando o resultando un

arreglo nuevo ($La_datos_error_filtro_result) con los pares $Ls_key=>$Ls_valor cuyos $Ls_key se encontraban en ambos arreglos (el arreglo cliente y el arreglo matriz).

Además también se ponen en el arreglo resultante $La_datos_error_filtro_result del filtrado, los elementos que están en la matriz (arreglo filtro) pero no en el arreglo cliente(arreglo cliente a filtrar), lo anterior se hace poniendo el índice $Ls_key y como valor de este su propio identificador string entre paréntesis('key').Esta funcionalidad depende de La_formato_filtrado['filtrado']['marca_ausente'] elemento que se explica más adelante.

El elemento $Ls_key que exista en el arreglo cliente(arreglo cliente a filtrar) y no exista en la matriz(arreglo filtro), no pasa a formar parte del arreglo resultante $La_datos_error_filtro_result.

Los parámetros se explican a continuación:

$La_datos_errorlog_filtro (array) es un arreglo con los pares $Ls_key=>$Ls_valor a filtrar.

$La_formato_filtrado (array) es un arreglo asociativo en el que este procedimiento buscará la estructura ['filtrado']['marca_ausente'], es decir
$La_formato_filtrado['filtrado']['marca_ausente'] , si existe esta estructura y tiene valor true define que, si existe un elemento en $La_matriz_elementos_errorlog y no existe ese elemento como $Ls_key en $La_datos_errorlog_filtro se incorpora al arreglo resultante del filtrado, la llave $Ls_key y como su valor entre paréntesis el propio key quedando de la siguiente forma $Ls_key = (‘Ls_key’).Esta estructura es opcional.

$La_matriz_elementos_errorlog (array) es un arreglo asociativo cuyo contenido se toma como referencia para el filtrado del arreglo $La_datos_errorlog_filtro, este arreglo hace de mascara de los $Ls_key(solo de las llaves $Ls_key) para el filtrado del arreglo $La_datos_errorlog_filtro. Su estructura es la siguiente:

$La_matriz_elementos_errorlog['miembros_errorlog']
propiedad que define los campos o elementos que se gestionaran para un contenedor de registros de errores, a esta propiedad siempre se le adicionan los valores del elemento ['miembros_error']
un ejemplo puede ser
array('date' , 'aplicacion' , 'modulo' , 'opcion')

$La_matriz_elementos_errorlog['miembros_error']
propiedad que define los campos o elementos que se gestionaran para un mensaje u otro destino de errores un ejemplo puede ser
array('id' , 'tipo' , 'tratamiento' , 'alcance' , 'mensaje')

el orden de los elementos key del arreglo resultante $La_datos_error_filtro_result los define el orden de los elementos del arreglo $La_matriz_elementos_errorlog.

Este procedimiento retorna el arreglo resultante con los elementos $Ls_key=>$Ls_valor cuyos $Ls_key de $La_datos_errorlog_filtro existían como elementos en $La_matriz_elementos_errorlog si su gestión fue satisfactoria, de lo contrario retorna el valor null. Si el arreglo resultante del filtrado es vacío también se retorna el valor null.

public function crearDatosErrorLog($La_datos_errorlog_filtro , $La_formato_filtrado , $La_matriz_elementos_errorlog , $La_fuente_datos_errorlog)

La función y objetivo de este procedimiento es crear un nuevo registro de datos errorlog en una fuente de datos erroreslog, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar erroreslog. En este caso es crear una nueva línea estructurada con datos en un fichero txt que hace de fuente de datos erroreslog.

Este procedimiento utiliza en su gestión el procedimiento $this->filtrarDatosErrorlog por lo que coinciden en gran parte los parámetros de este procedimiento con los del procedimiento $this->filtrarDatosErrorlog

Los parámetros se explican a continuación:

$La_datos_errorlog_filtro (array) es un arreglo con los pares $Ls_key=>$Ls_valor a filtrar, son los pares que el cliente quiere crear como un nuevo registro de datos de errorlog en la fuente de datos erroreslog. De este parámetro se utilizan los $Ls_key en el filtrado y solo pasan como datos de la estructura de la línea del txt los $Ls_valor.

$La_formato_filtrado (array) es un arreglo asociativo en el que se encuentran datos relacionados con el filtrado y el formato de datos erroreslog, este procedimiento utiliza los siguientes:

1- este procedimiento buscará la estructura ['filtrado']['marca_ausente'], es decir $La_formato_filtrado['filtrado']['marca_ausente'] , si existe esta estructura y tiene valor true define que, si existe un elemento en $La_matriz_elementos_errorlog y no existe ese elemento como $Ls_key en $La_datos_error_filtro se incorpora al arreglo resultante del filtrado, la llave $Ls_key y como su valor entre paréntesis el propio key quedando de la este procedimiento buscará la estructura ['formato']['separador'], es decir $La_formato_filtrado['formato']['separador'], esta estructura debe contener un string que se utilizará como separador de los elementos datos que conformarán la línea de datos de errorlog que se guardará en el fichero txt , que hace de fuente de datos erroreslog. Esta estructura es obligatoria.
2- este procedimiento buscará la estructura ['formato']['separador'], es decir $La_formato_filtrado['formato']['separador'], esta estructura debe contener un string que se utilizará como separador de los elementos datos que conformarán la línea de datos de errorlog que se guardará en el fichero txt , que hace de fuente de datos errores. Esta estructura es obligatoria.

$La_matriz_elementos_errorlog (array) es un arreglo asociativo cuyo contenido se toma como referencia para el filtrado del arreglo $La_datos_errorlog_filtro, este arreglo hace de mascara de los $Ls_key(solo de las llaves $Ls_key) para el filtrado del arreglo $La_datos_errorlog_filtro. Su estructura es la siguiente:

$La_matriz_elementos_errorlog ['miembros_errorlog']
 propiedad que define los campos o elementos que se gestionaran para un contenedor de registros de erroreslog, a esta propiedad siempre se le adicionan los valores del elemento ['miembros_error']
 un ejemplo puede ser
 array('date' , 'aplicacion' , 'modulo' , 'opcion')

$La_matriz_elementos_errorlog ['miembros_error']
propiedad que define los campos o elementos que se gestionaran para un mensaje u otro destino de errores un ejemplo puede ser
array('id' , 'tipo' , 'tratamiento' , 'alcance' , 'mensaje')

$La_fuente_datos_errorlog (array) es quien contendrá los datos necesarios para acceder a la fuente de datos erroreslog, en este caso el fichero txt , que hace de fuente de datos erroreslog. Este procedimiento buscará la estructura ['path_fich_error'], es decir $La_fuente_datos_errorlog['path_fich_errorlog'], esta estructura debe contener un string que se utilizará como path donde se encuentra el fichero a incorporar la nueva línea de datos de errorlog. Esta estructura es obligatoria.

Este procedimiento retorna el valor true si se lleva a cabo satisfactoriamente su gestión, de lo contrario retorna el valor null. Si la gestión es satisfactoria y no existe el fichero en $La_fuente_datos_errorlog['path_fich_error'], se intentará crear el fichero txt con la estructura gestionada en el path $La_fuente_datos_errorlog['path_fich_error'].

public function eliminarDatosErrorLog($La_datos_errorlog_filtro , $La_formato_filtrado , $La_matriz_elementos_errorlog , $La_fuente_datos_errorlog)

La función y objetivo de este procedimiento es eliminar registros de datos errorlog en una fuente de datos erroreslog, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar erroreslog. En este caso es eliminar una línea, estructurada con datos en un fichero txt que hace de fuente de datos erroreslog, se elimina si esta línea en su estructura de datos erroreslog tiene datos coincidentes con el criterio de eliminación que viene dado por el arreglo $La_datos_errorlog_filtro.

Los parámetros se explican a continuación:

$La_datos_errorlog_filtro (array) es un arreglo con los pares $Ls_key=>$Ls_valor a filtrar, son los pares que el cliente pone como condición para eliminar líneas de datos del txt, registros de datos de errorlog en la fuente de datos erroreslog. Son comparados los pares de este parámetro con los pares

formados con $La_matriz_elementos_errorlog(como key_txt) y el dato en la línea del txt (como valor_txt), y si coinciden todos los pares entonces se elimina la línea del txt.

$La_formato_filtrado (array) es un arreglo asociativo en el que se encuentran datos relacionados con el filtrado y el formato de datos erroreslog, este procedimiento utiliza los siguientes:

1- este procedimiento buscará la estructura ['filtrado']['marca_ausente'], es decir $La_formato_filtrado['filtrado']['marca_ausente'] , si existe esta estructura y tiene valor true define que, si existe un elemento en $La_matriz_elementos_errorlog y no existe ese elemento como $Ls_key en $La_datos_error_filtro se incorpora al arreglo resultante del filtrado, la llave $Ls_key y como su valor entre paréntesis el propio key quedando de la siguiente forma $Ls_key = (‘Ls_key’). Esta estructura es opcional.
2- este procedimiento buscará la estructura ['formato']['separador'], es decir $La_formato_filtrado['formato']['separador'], esta estructura debe contener un string que se utilizará como separador de los elementos datos que conformarán la línea de datos de errorlog que se buscarán en el fichero txt , que hace de fuente de datos erroreslog. Esta estructura es obligatoria.
3- este procedimiento buscará la estructura ['filtrado']['llave_unica'], es decir $La_formato_filtrado ['filtrado']['llave_unica'], esta estructura admite un string igual al de un $Ls_key de $La_datos_errorlog_filtro, de existir este string cuando se hace la comparación de datos para buscar coincidencias, si un par $Ls_key=>$Ls_valor coincide con los datos del txt y a la ves este $Ls_key es también 'llave_unica', entonces se detiene el proceso de filtrado , también se detiene el proceso de búsqueda por las demás líneas del txt y se elimina la línea con la coincidencia. Esta estructura es opcional.

$La_matriz_elementos_errorlog (array) es un arreglo asociativo cuyo contenido se toma como referencia para el filtrado de las líneas del txt, este arreglo hace de complemento $Ls_key(solo de las llaves $Ls_key) para el casamiento con los datos en la línea del txt que figurarán como valor de las $Ls_key. Su estructura es la siguiente:

$La_matriz_elementos_errorlog ['miembros_errorlog']
propiedad que define los campos o elementos que se gestionaran para un contenedor de registros de erroreslog, a esta propiedad siempre se le adicionan los valores del elemento ['miembros_error']
un ejemplo puede ser
array('date' , 'aplicacion' , 'modulo' , 'opcion')

$La_matriz_elementos_errorlog ['miembros_error']
propiedad que define los campos o elementos que se gestionaran para un mensaje u otro destino de errores un ejemplo puede ser
array('id' , 'tipo' , 'tratamiento' , 'alcance' , 'mensaje')

$La_fuente_datos_errorlog (array) es quien contendrá los datos necesarios para acceder a la fuente de datos erroreslog, en este caso el fichero txt , que hace de fuente de datos erroreslog. Este procedimiento buscará la estructura ['path_fich_error'], es decir

$La_fuente_datos_errorlog['path_fich_errorlog'], esta estructura debe contener un string que se utilizará como path donde se encuentra el fichero a filtrar y eliminar las líneas de datos de error coincidentes con el criterio de eliminación que viene dado por el arreglo $La_datos_errorlog_filtro. Esta estructura es obligatoria.

Este procedimiento retorna:
retorna true si encuentra elemento al que aplicar la condición de filtrado y lo realiza satisfactoriamente.
retorna false si encuentra elemento al que aplicar la condición de filtrado y no le es posible realizar los cambios en el fichero.
retorna null si no encuentra elemento al que aplicar la condición de filtrado o se le pasan valores incompatibles a los parámetros del procedimiento.

public function leerDatosErrorLog($La_datos_errorlog_filtro , $La_formato_filtrado , $La_matriz_elementos_errorlog , $La_fuente_datos_errorlog)

La función y objetivo de este procedimiento es leer registros de datos errorlog en una fuente de datos erroreslog, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar erroreslog. En este caso es leer una línea, estructurada con datos en un fichero txt que hace de fuente de datos erroreslog, se lee si esta línea en su estructura de datos erroreslog tiene datos coincidentes con el criterio de lectura que viene dado por el arreglo $La_datos_errorlog_filtro.

Los parámetros se explican a continuación:

$La_datos_errorlog_filtro (array) es un arreglo con los pares $Ls_key=>$Ls_valor a filtrar, son los pares que el cliente pone como condición para leer líneas de datos del txt, registros de datos de errorlog en la fuente de datos erroreslog. Son comparados los pares de este parámetro con los pares formados con $La_matriz_elementos_errorlog(como key_txt) y el dato en la línea del txt (como valor_txt), y si coinciden todos los pares entonces se lee(toma) la línea del txt.

$La_formato_filtrado (array) es un arreglo asociativo en el que se encuentran datos relacionados con el filtrado y el formato de datos erroreslog, este procedimiento utiliza los siguientes:

1- este procedimiento buscará la estructura ['formato']['separador'], es decir $La_formato_filtrado['formato']['separador'], esta estructura debe contener un string que se utilizará como separador de los elementos datos que conformarán la línea de datos de errorlog que se buscarán en el fichero txt , que hace de fuente de datos erroreslog. Esta estructura es obligatoria.
2- este procedimiento buscará la estructura ['filtrado']['llave_unica'], es decir $La_formato_filtrado ['filtrado']['llave_unica'], esta estructura admite un string igual al de un $Ls_key de $La_datos_errorlog_filtro, de existir este string cuando se hace la comparación de datos para buscar coincidencias, si un par $Ls_key=>$Ls_valor coincide con los datos del txt y a la

vez este $Ls_key es también 'llave_unica', entonces se detiene el proceso de filtrado , también se detiene el proceso de búsqueda por las demás líneas del txt y se lee la línea con la coincidencia. Esta estructura es opcional.

$La_matriz_elementos_errorlog (array) es un arreglo asociativo cuyo contenido se toma como referencia para el filtrado de las líneas del txt, este arreglo hace de complemento $Ls_key(solo de las llaves $Ls_key) para el casamiento con los datos en la línea del txt que figurarán como valor de las $Ls_key. Su estructura es la siguiente:

$La_matriz_elementos_errorlog ['miembros_errorlog']
 propiedad que define los campos o elementos que se gestionaran para un contenedor de registros de errores, a esta propiedad siempre se le adicionan los valores del elemento ['miembros_error']
 un ejemplo puede ser
 array('date' , 'aplicacion' , 'modulo' , 'opcion')

$La_matriz_elementos_errorlog ['miembros_error']
propiedad que define los campos o elementos que se gestionaran para un mensaje u otro destino de errores un ejemplo puede ser
array('id' , 'tipo' , 'tratamiento' , 'alcance' , 'mensaje')

$La_fuente_datos_errorlog (array) es quien contendrá los datos necesarios para acceder a la fuente de datos erroreslog, en este caso el fichero txt , que hace de fuente de datos erroreslog. Este procedimiento buscará la estructura ['path_fich_error'], es decir $La_fuente_datos_errorlog['path_fich_errorlog'], esta estructura debe contener un string que se utilizará como path donde se encuentra el fichero a filtrar y leer las líneas de datos de errorlog coincidentes con el criterio de lectura que viene dado por el arreglo $La_datos_errorlog_filtro. Esta estructura es obligatoria.

Este procedimiento retorna:
retorna arreglo multidimensional donde el primer nivel de llaves o índices es de tipo numérico automático y en cada uno de estos elementos existirá un arreglo asociativo(con índices o llaves de tipo string) donde cada uno de sus elementos contendrá como valor un string correspondiente con las secciones o propiedades que contenían las líneas del txt que coincidieron con el filtrado, cada arreglo contenido en los índices numéricos representa los datos de una línea del txt que coincidieron con el filtrado.

retorna null si no encuentra elemento al que aplicar la condición de filtrado o se le pasan valores incompatibles a los parámetros del procedimiento.

public function modificarDatosErrorLog($La_datos_errorlog_filtro , $La_datos_sustitutos , $La_formato_filtrado , $La_matriz_elementos_errorlog , $La_fuente_datos_errorlog)

La función y objetivo de este procedimiento es modificar registros de datos errorlog en una fuente de datos erroreslog, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar erroreslog. En este caso es modificar una línea, estructurada con datos en un fichero txt que hace de fuente de datos erroreslog, se modifica si esta línea en su estructura de datos erroreslog tiene datos coincidentes con el criterio de modificación que viene dado por el arreglo $La_datos_errorlog_filtro.

Los parámetros se explican a continuación:

$La_datos_errorlog_filtro (array) es un arreglo con los pares $Ls_key=>$Ls_valor a filtrar, son los pares que el cliente pone como condición para modificar líneas de datos del txt, registros de datos de errorlog en la fuente de datos erroreslog. Son comparados los pares de este parámetro con los pares formados con $La_matriz_elementos_error(como key_txt) y el dato en la línea del txt (como valor_txt), y si coinciden todos los pares entonces se modifica la línea del txt con el dato correspondiente (de igual key) en el arreglo $La_datos_sustitutos.
En caso de querer modificar valores en todas las líneas del fichero, este parámetro soporta el valor '*', que le indica al procedimiento que realice la sustitución en todas las líneas del fichero sin hacer filtrado.

$La_datos_sustitutos (array) es un arreglo con los pares $Ls_key=>$Ls_valor a modificar en la línea del txt, no tienen que coincidir con los pares de $La_datos_errorlog_filtro.

$La_formato_filtrado (array) es un arreglo asociativo en el que se encuentran datos relacionados con el filtrado y el formato de datos erroreslog, este procedimiento utiliza los siguientes:

3- este procedimiento buscará la estructura ['formato']['separador'], es decir $La_formato_filtrado['formato']['separador'], esta estructura debe contener un string que se utilizará como separador de los elementos datos que conformarán la línea de datos de errorlog que se buscarán en el fichero txt , que hace de fuente de datos erroreslog. Esta estructura es obligatoria.
4- este procedimiento buscará la estructura ['filtrado']['llave_unica'], es decir $La_formato_filtrado ['filtrado']['llave_unica'], esta estructura admite un string igual al de un $Ls_key de $La_datos_errorlog_filtro, de existir este string cuando se hace la comparación de datos para buscar coincidencias, si un par $Ls_key=>$Ls_valor coincide con los datos del txt y a la vez este $Ls_key es también 'llave_unica', entonces se detiene el proceso de filtrado , también se detiene el proceso de búsqueda por las demás líneas del txt y se modifica la línea con la sustitución. Esta estructura es opcional.

$La_matriz_elementos_errorlog (array) es un arreglo asociativo cuyo contenido se toma como referencia para el filtrado de las líneas del txt, este arreglo hace de complemento $Ls_key(solo de las llaves $Ls_key) para el casamiento con los datos en la línea del txt que figurarán como valor de las $Ls_key. Su estructura es la siguiente:

$La_matriz_elementos_errorlog ['miembros_errorlog']
propiedad que define los campos o elementos que se gestionaran para un contenedor de registros de erroreslog, a esta propiedad siempre se le adicionan los valores del elemento ['miembros_error']
un ejemplo puede ser
array('date' , 'aplicacion' , 'modulo' , 'opcion')

$La_matriz_elementos_errorlog ['miembros_error']
propiedad que define los campos o elementos que se gestionaran para un mensaje u otro destino de errores un ejemplo puede ser
array('id' , 'tipo' , 'tratamiento' , 'alcance' , 'mensaje')

$La_fuente_datos_errorlog (array) es quien contendrá los datos necesarios para acceder a la fuente de datos erroreslog, en este caso el fichero txt , que hace de fuente de datos erroreslog. Este procedimiento buscará la estructura ['path_fich_error'], es decir
$La_fuente_datos_errorlog['path_fich_errorlog'], esta estructura debe contener un string que se utilizará como path donde se encuentra el fichero a filtrar y modificar las líneas de datos de errorlog coincidentes con el criterio de modificación que viene dado por el arreglo
$La_datos_errorlog_filtro. Esta estructura es obligatoria.

Este procedimiento retorna:
retorna true si encuentra elemento al que aplicar la condición de filtrado y lo realiza satisfactoriamente.
retorna false si encuentra elemento al que aplicar la condición de filtrado y no le es posible realizar los cambios en el fichero.
retorna null si no encuentra elemento al que aplicar la condición de filtrado o se le pasan valores incompatibles a los parámetros del procedimiento.

public function crearFuenteDatosErrorLog($La_fuente_datos_errorlog , $Ls_cadena_inicio = '')

La función y objetivo de este procedimiento es crear una fuente de datos erroreslog, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar erroreslog. En este caso es crear un fichero txt donde cada línea del fichero es un registro de datos estructurados, como fuente de datos de erroreslog.

Los parámetros se explican a continuación:

$La_fuente_datos_errorlog (array) es quien contendrá los datos necesarios para crear la fuente de datos erroreslog, en este caso el fichero txt , que hace de fuente de datos erroreslog. Este procedimiento buscará la estructura ['path_fich_errorlog'], es decir
$La_fuente_datos_errorlog['path_fich_errorlog'], esta estructura debe contener un string que se utilizará como path donde se creará el fichero txt.

$Ls_cadena_inicio (string) cadena a escribir en el fichero txt, cadena que conforma el cuerpo del fichero de datos de erroreslog, este procedimiento trabaja con la función file_put_contents de php.

Este procedimiento retorna true si su gestión fue satisfactoria y null si fue insatisfactoria.

public function eliminarFuenteDatosErrorLog($La_fuente_datos_errorlog)

La función y objetivo de este procedimiento es eliminar una fuente de datos erroreslog, fuente de datos de la que se nutren otros procedimientos de esta clase para gestionar erroreslog. En este caso es eliminar un fichero txt donde cada línea del fichero es un registro de datos estructurados, como fuente de datos de erroreslog.

Los parámetros se explican a continuación:

$La_fuente_datos_errorlog (array) es quien contendrá los datos necesarios para eliminar la fuente de datos erroreslog, en este caso el fichero txt , que hace de fuente de datos erroreslog. Este procedimiento buscará la estructura ['path_fich_errorlog'], es decir $La_fuente_datos_errorlog['path_fich_errorlog'], esta estructura debe contener un string que se utilizará como path del fichero txt a eliminar.

Este procedimiento retorna true si su gestión fue satisfactoria y null si fue insatisfactoria.

Directorio Modulos

Es el directorio que contendrá todos los códigos clientes que utilizan el sistema Sockeletom como plataforma

Programador Cliente

- Es todo aquel que programa o es responsable de código que se ejecute sobre la plataforma Sockeletom.

Código Cliente

- Es todo aquel código que se ejecute sobre la plataforma Sockeletom, que utilice y se beneficie de las estructuras y herramientas que brinda el sistema Sockeletom para hacer más portables fáciles y rápidos el diseño e implementación de códigos y sus combinaciones. Estos códigos pueden ser scripts, aplicaciones, sistema de aplicaciones etc.