SLAMAX WEB DK

TECNICAL DOCUMENTATION

FOR DEVELOPERS

SUMMARY

USEFUL DEFINITIONS TO READ THE DOCUMENTATION

COMMON TERMINOLOGY

CSS = Cascade Style Sheets, simple css

JAVASCRIPT = Javascript, client side programming language for the web

PHP = Server side programming language for the web

SLAMAX WEBDK TERMINOOGY

VIEWPORT = the current browser windows and his screen output

Runtime Script = JAVASCRIPT code

Styling Script = CSS code

class = PHP language class

SETUP AND INTRODUCTION TO Slamax WebDK

If it's the first time reading this documentation , than you can go the object section of this documentation right now, because you just need the object to understand and make somthing simple just to take confidence within the Slamax WebDk, particullary if it's the first time you are trying to create sometghing with Slamax WebDK, after reading this page, we suggest to procede directly to the object section and focus on the object Template, because it rappresents the start point where to start for making a website or a web application.

Let alone the Kernel and all abstract classes (Browser, etc..), those classes and function are not in particular interest of the devolepore, beacuse they are internally managed by Slamax WebDK's objects. They are documented both for having more knolowedge, or to give the power to the developer of expanding easily Slamax WebDK modules, giving a complete management of the functionality of this library.

Installing Slamax WebDK is very simple! If you are using the free version than you have downloaded our includelib.php file from github and you are requested a Key to the address: info@slamaxsoftware.com, you can than include this file.

If you bought this library than you own a file zip call Slamax_WebDK1_X.zip, you have to uncrompress it and copy it into the lib folder of your server (ex: localhost/mywebsite), by including the file "./lib/include.php"

class **Kernel**

Kernel manages authorizations for basic operation,

it makes shure all calls are correctly and securly done,

manages inputs, sessions, forms and datbases requests.

Kernel has 3 buffers:

String Buffer (contains the text and Html code of the page)

Css Buffer (contains the Css code of the page)

Js Buffer (contains the Javascript code of the page)

function **UseCache**

Enables Cache usage.

Doesn't need any parameter.

```
Kernel::UseCache();
```

function **RenderCachedFiles**

Renders all files save to the Cache.

Doesn't need any parameter.

```
Kernel::RenderCachedFiles();
```

function **FormatCache**

Sets the Cacher with given arguments.

Requires any object of type ChacheInfo as a parameter.

```
Kernel::FormatCache(...CacheInfo $info);
```

function **hasCached**

Returns TRUE if the cache is enable.

Doesn't need any parameter.

Kernel::hasCached()

function **isCaching**

Returns TRUE if is caching right now.

Doesn't need any parameter.

Kernel::isCaching()

function **SetContext**

cambia il contesto per la modalita di rendering degli oggetti.

Changes the context for the current rendering mode for objects.

It requires a _CONTEXT constant.

Kernel::SetContext(int $context)

function **GetContext**

Return the current rendering mode setted.

Doesn't need any parameter.

Kernel::GetContext()

function **CreateBuffer**

Creates a buffer for outputting to the browser as a string.

Requires the buffer name as a string,

Any kind of buffer content (OPTIONAL)

Kernel::CreateBuffer(string $name, $inCont)

function **Refresh**

Refreshes the current page, can be called only once.

Requires an url as a string,

and a delay time in seconds(OPZIONALE)

Kernel::Refresh(string $url, int $seconds)

function **AddHeaders**

Adds HttpHeaders to the current session.

Requires an array that has the header name as the index name,

e as an element the HttpHeader itself.

Kernel::AddHeaders(array $headers)

function **DestroyBuffer**

Destroys an existing buffer.

Requires the name of the buffer to destroy as a string.

Kernel::DestroyBuffer(string $name)

function **AddToBuffer**

Adds content to the buffer as text.

Requires the buffer name as a string,

and the buffer content.

Kernel::AddToBuffer(string $name, $cont)

function **BufferExists**

Returns TRUE if the buffer exist and it's correctly initialized.

Requires the buffer name as a string.

Kernel::BufferExists(string $name)

function **flushBuffers**

Prints to the screen the content of all buffers, and empties them.

Doesn't need any parameter.

Kernel::flushBuffers()

function **InstantiateObject**

Instantiate an object preparing it for the scripting, assigins it a univoque ID for the instance, and it's needed to enable Runtime Script functions.

Requires a SlamaxObject to initialize.

Kernel::InstantiateObject(SlamaxObject $h)

function **GetBrowser**

Returns a Browser object, only once.

Doesn't need any parameter.

Kernel::GetBrowser()

function **setScope**

Replaces the current scope with the given BufferedObject, wich becomes the current.

Requires and object of type BufferedObject to set as a scope.

Kernel::setScope(&$b);

function **GetCurrentScope**

Returns the currently setted scope, it returns NULL if no scope has been set.

Doesn't need any parameter.

```
Kernel::GetCurrentScope();
```

class **HttpStatusManager**

HttpStatusManager it's an abstract class that manages all the operation of Http request and the Http status of the server HOST.

function **getSupportedBrowsers**

Return the names of all Slamax WebDK supported browsers.

Doesn't need any parameter.

HttpStatusManager::getSupportedBrowsers();

function **isHostSecure**

Returns TRUE if passes a check that verifies if Host is safe, and respect https protocols.

Doesn't need any parameter.

HttpStatusManager::isHostSecure()

funzione **GetUserIp**

restituisce lindirizzo IP dell utente attualmente connesso al Server.

non richiede alcun parametro.

HttpStatusManager::GetUserIp();

function **GetError**

Returns the error code generated by the HttpHeader.

Doesn't need any parameter.

HttpStatusManager::GetError();

function **GetAgent**

Return the agent name (Browser name) active in the current session.

Doesn't need any parameter.

HttpStatusManger::getAgent();

Slamax WebDK CORE

The core section has all the functionality that are to the base fo Slamax WebDK, wit them you can customize the library behavior, and create new components by extending the core components itself.

WARNING:

Those functionalities are not absolutely needed to use Slamax WebDK, because they are higly utilized in the library source code. If you really need those function than we sugget to read well this documenttion, beacuse otherwise you are at risk of putting Slamax WebDK in a not valid configuration state.

class **Browser**

It's a class able to record automatically on Slamax WebDK buffer, the CSS and JAVASCRIPT code and the HTML code.

```
$myBrowser = new Browser(string $doctype, BrowserInfo $info);
```

function **init**

Initializes the Browser, it is need to call every other Browser function.

Doesn't need any parameter.

```
$myBrowser->init();
```

function **releaseScripts**

records every SlamaxObject Runtime Script, and saves it tho Javascript Buffer.

Doesn't need any parameter.

```
$myBrowser->releaseScripts();
```

function **Close**

Closes every browser related functionaly and it's necessary to make any screen output.

Doesn't need any parameter.

```
$myBrowser->Close();
```

function **addLib**

Adds a link of a dependecy for the Browser.

Requires the script type for the dependecy as a string,

and the address of the dependency as a string.

```
$myBrowser->addLib(string $type, string $src);
```

function **addCachedStylesheet**

Adds a Css file already cached by the Browser.

Requires the address of the cached file as a string.

```
$myBrowser->addCachedStylesheet(string $src);
```

function **addCachedScript**

Adds a Script file already cached by the Browser.

Requires the address of the cached file as a string.

```
$myBrowser->addCachedScript(string $src);
```

function **addTitleTag**

adds a Tag Title (<title>..</) to the current VIEWPORT.

Requires the title text as a string.

```
Browser::addTitleTag(string $content);
```

function **isValidCookie**

Returns TRUE if the Cookie exists and it's a valid cookie.

Doesn't need any parameter.

```
Browser::isValidCookie(string $cookieName);
```

function **CreateCookie**

Creates a new Cookie to current session.

Requires the Cookie name as a string,

the content of the Cookie,

lifetime in milliseconds as an integer (OPZIONALE),

TRUE to use the Cookie safe protocol (OPZIONALE),

TRUE to allow for non safe cookie protocols (OPZIONALE)

```
Browser::CreateCookie(string $cookieName, $cont, int $exp, string $path, bool $secure, bool $httponly);
```

function **GetCookie**

Return the required Cookie.

Requires the Cookie name as a string.

```
Browser::GetCookie(string $name);
```

function **DestroyCookie**

Destroys the request Coookie.

Requires the name of the Cookie to destroy as a string.

```
Browser::DestroyCookie(string $name);
```

function **isValidSession**

Returns TRUE if the session is existing and valid.

Requires the name of the session as a string.

```
Browser::isValidSession(string $name);
```

function **CreateSession**

Creates a new session for client browser.

Requires the session name as a string,

and the session value.

```
Browser::CreateSession(string $name, $value);
```

function **GetSession**

Return the requested session, if it exist.

Requires the name of the session as a string.

```
Browser::GetSession(string $name);
```

function **DestroySession**

Destroys the required session, destroys all sessions if called with no parameters.

Requires the name of the session to destroy as a string.

```
Browser::DestroySession(string $name);
```

function **ResetBrowserStyle**

Adds the reset style, updates the browser with Slamax SDK default CSS.

Doesn't need any parameter.

```
Browser::ResetBrowserStyle();
```

function **addMetaTags**

Adds metatags.

Requires the content name as a string.

```
Browser::addMetaTags(string $meta)
```

function **addStringOutput**

Adds the given string, as an output for the VIEWPORT.

Requires any kind of string.

```
Browser::addStringOutput(string $s)
```

class **Validator**:

Abstract class that allows for the sanification of strings, as an help forms string sanification based logics more easily.

function **startValidation**

Initiliazes the autovalidator.

Requires the maximum password length as an integer (OPZIONALE, default 16)

and the minimum password length as an integer (OPZIONALE, default 8)

and the maximum text length as an integer (OPZIONALE, default 600)

and the maximum name length as an integer (OPZIONALE, default 40)

Validator::startValidation(int $maxPwdL, int $minPwdL, int $maxText, int $maxTextLength)

function **sanifyString**

Return the given string sanified from special characters and code.

Requires any kind of string.

Validator::sanifyString(string $s)

function **StopValidation**

Disables validation.

Doesn't need any parameter.

Validator::StopValidation()

class **EventListener**:

Activates , manages and listens to object events.

```
$myEventListener = new EventLister(BufferedObject $h);
```

function **isCommonEvent**

Returns TRUE if the event is a generic type.

Requires the name of the event as a string.

```
EventListener::iScommonEvent(string $eventName)
```

function **isActorEvent**

Returns TRUE if the event is of type Actor.

Requires the event name as a string.

```
EventListener::iSactorEvent(string $event)
```

function **isCanvasEvent**

Returns TRUE if the event is of type Canvas.

Requires the event name as a string.

```
EventListener::iScanvasEvent(string $event)
```

function **isValidEvent**

Returns TRUE if the Event is a Slamax WebDK event.

Requires the event name as a string.

```
EventListener::iSvalidEvent(string $event);
```

function **AddEvent**

Adds an event to the listener.

Requires an object of type Event.

```
$myEventListener->AddEvent(Event $event);
```

function **ReleaseEvents**

Calls all events.

Doesn't need any parameter.

```
$myEventListener->ReleaseEvents();
```

function **callEvent**

Call the required event.

Requires the name of the event as string.

```
$myEventListener->CallEvent(string $num);
```

class BufferedObject

Objects capable of creating a temporaty buffer for the HTML code and record the CSS and/or JAVASCRIPT code generated from the object to the respective Slamax WebDK browser buffers.

```
$myBufferedObject = new BufferedObject();
```

function **Get**:

Adds the object to the Buffer, and puts his code to the browser's buffers.

Doesn't need any parameter.

```
$myBufferedOject->Get();
```

class **Rule**

Object that executes the given function, but only if all the condition passes as array in the initialization, are met.

```
$myRule = new Rule(array $arr, callable $fun);
```

function **respected**

Return TRUE if all the codintion given are true;

Doesn't need any parameter.

```
$myRule->respected()
```

function **release**

Force the call to the setted function, even if respected() returns FALSE.

Doesn't need any parameter.

```
$myRule->release();
```

class **Buffer**

A Buffer able the contain code in the form of text.

```
$myBuffer = new Buffer(string $nomeBuffer, string $cont);
```

function **Clear**

Clears complety the Buffer and registers the old Buffer.

Non richiede alcun parametro.

```
$myBuffer->Clear();
```

function **Add**

Adds any content to the buffer.

Requires any content able to be casted to string.

```
$myBuffer->Add($buff);
```

function **isEmpty**

Returns TRUE if the buffer is empty.

Doesn't need any parameter.

```
$myBuffer->isEmpty()
```

class **DataObject**:

An object that can contain any kind of data,

enables his parameters to accesed publicly,

by being a common and uniform point of access.

Requires an array that has for the index the varible name,

and the element is content of that varibale.

```
$myData = new DataObject(array $data);
```

function **toStringFile**:

converts its data into a text file.

Requires an object of type Url,

and the file name as a string.

```
$myData->toStringFile(Url $u, string $name);
```

function **toJsonFile**

Converts its data into a json file.

Requires an object of type Url,

and the file name as a string.

```
$myData->toJsonFile(Url $u, string $name);
```

function **toArray**

Returns all variable converted back into an array.

Doesn't need any parameter.

```
$myData->toArray()
```

class **CacheInfo** extends **DataObject**

DataObject that contains cache information, see DataObject.

Requires the buffer name as a string,

the root address as a string,

the file name as a string,

the type of the CacheFile as a string.

```
$myCacheData = new Cacheinfo(string $buffName, string $root, string $name, string $type);
```

class **FormData** extends **DataObject**

DataObject that contains Form calls informations, see DataObject.

```
$myFormData = new FormData(array $data);
```

class **SqlData** extends **DataObject**

DataObject that contains Sql call information, see DataObject.

```
$mySqlData = new SqlData(string $buffName, string $root, string $name, string $type);
```

class **Controller**

Simple class to controll any object from an external class.

Requires the object to control with dinamic reference.

```
$myController = new Controller($this);
```

class **jsModule** extends **Controller**

Class that behaives like a module for Javascript.

see Controller.

```
$myModule = new jsModule($this);
```

class **JavascriptComponent**

Assigns to and Html object the events for Javascript.

By being assigned to the eventListener, it prevents usage from outside a Javascript Component.

Requires a SlamaxObject reference in the __construct method.

```
$myJsComponent = new JavascriptComponent($this);
```

class SqlHelp

Abstract class that act as helper for other sql related classes.

function **getNewLimit**

Return an object of type SqlLimit, for data pagination.

Requires the element number as an integer,

the search page as an integer,

the maximum value as an integer.

SqlHelp::getNewLimit($els=0, $pcr, $max)

function **getIDFromResult**

Returns the ID parameter inside a SqlData object,

Requires an object of type DataObject, in wich to search.

SqlHelp::getIDFromResult(DataObject $result)

function **getDataFromIDinTable**

Returns an object of type DataObject based on the given ID.

Requires the name of the name as a string,

and the ID of the sql result as an integer.

SqlHelp::getDataFromIDinTable(string $table, int $dataid)

function **getNextFreeIDinTable**

Returns the next avaible ID from current request.

Requires the name of the table as a string.

SqlHelp::getNextFreeIDinTable(string $table)

function **getLastIDinTable**

Returns the last ID inside a table.

Requires the name of the table as a string.

SqlHelp::getLastIDinTable(string $table)

function **isStringNotID**

Returns TRUE if passes a check that verifies the absence of ID related characters.

Requires any string to verify.

SqlHelp::isStringNotID(string $s)

function **escapeString**

Returns a safe version of the given string.

Requires the value to escape as a string.

SqlHelp::escapeString(string $value)

class **JsCompiler**

Abstract class that enables Javascript compilation for all SlamaObjects

function **Assert**

Allows to generate a Javascript assertion,

returns TRUE if the assertion succeded.

Requires a JS_ type constant,

the assertion name as a string,

the assertion symbol as a string (OPTIONAL),

the assertion value as a string (OPTIONAL).

```
JsCompiler::Assert(int $type, string $w, string $s, string $a);
```

class **Url**

Creates an Url object that manages safely http address between classes.

```
$myUrl = new Url(string $path, string $fakePath='', array $params=null);
```

DATABASE MANAGEMENT

Slamax WebDK contains some database managent functionalities, right now it uses MySqli and is compatible all Php-MySql configuration.

(Doesnt not support multiple databases and pool queries)

Those classes allows you complete mnemonic and ripetive operations without the need of rewrite every time your code, it keeps it simple, and managable.

It abstracts some concept as the table, the database and the operations, it allows for a simple setup that comes in three stages:

- Connection to the database: Slamax WebDK is connected to a Sql database

- Chosse the operation: Choose any Sql operation to execute from an existing table inside the connected database.

- Connection closing: This gets done automacally, because it is internally managed by the library.

Kernel database-functions

function **ConnectToDatabase**

Establishes a MySqli connection with a database,

returns TRUE if succeded.

Requires the server addres as a string,

Requires the sql username as a string,

the database password as a string,

the table name as string.

```
Kernel::connectToDatabase(string $server,string $user,string $password,string $name);
```

function **getDatabaseConnection**

Return the currently active sql connection,

throws an Exception if no connection is active.

Doesn't need any parameter.

```
Kernel::getDatabaseConnection();
```

class **SqlParam**

It accepts a A and B condition,

choosing a coniugation simbol.

Require the condition A as a string,

the symbol as a string (OPTIONAL),

and the B condition as a string.

```
$mySqlParam = new SqlParam('A','==','B');
```

class **SqlCond** extends **SqlParam**

SqlParam that creates a condition for the current request.

See SqlParam.

Requires the A condition as a string,

and the B condition as a string.

```
$mySqlCond = new SqlCond('A','B');
```

class **SqlLimit** extends **SqlParam**

SqlParam that creates a limit for the current request,

Vedere SqlParam.

Richiede condizione A come stringa,

e la condizione B come stringa

```
$mySqlLimit = new SqlLimit('A','B');
```

class **SqlOrder** extends **SqlParam**

SqlParam that sort the result data of the sql request,

vedere SqlParam.

Requires the table name as a string,

and a _ORDER type constant.

```
$mySqlOrder = new SqlOrder($name,'ASC');
```

classe **SqlRequest**

class that uses SqlHelp functions e allows safe sql queries exectution .

Richiede il nome della tabella come stringa.

```
$myRequest = new SqlRequest($nomeTab);
```

function **execute**

Executes the current sqlRequest with the setted query.

Doesn't need any parameter.

```
$myRequest->execute() ;
```

function **Select**

Sets the query to execute a Select from the database.

Requires the parameter selector as a string,

and any kind of SqlParam. (OPTIONAL)

```
$myRequest->select($w='*',  ...SqlParams $params);
```

function **countTable**

Sets the query that counts the element int database table,

Requires the parameter selector as a string,

and any kind of SqlParam. (OPTIONAL)

```
$myRequest->countTable(string $w='*', ...SqlParams $params)
```

function **insert**

Sets some element to be insert in the table.

Requires the values to inser as an array,

that has as name the table row name,

and as element the value to insert..

```
$myRequest->insert(array $values);
```

function **update**

Sets the query to update some values in a given table.

Requires the values to inser as an array that has as name the table row name,

and as element the value to insert..

and any object of typeSqlCond (OPTIONAL)

```
$myRequest->update(array $values, SqlCond $con);
```

function **delete**

Sets the query to delete a row inside the database table.

Requires any object of type SqlCond.

```
$myRequest->delete(SqlCond $c=null)
```

function **resetTableIds**

Executes a query that resets the prymary Key counter of the table.

Requires the new id as an integer.

```
$myRequest->resetTableIds($reseteid);
```

FILE MANAGEMENT

Slamax WebDK contains a simple file managent system, multiplatform. It allows to manipulate files, without the need of all tose mnemonic controls that clarifies the behaviour of the manipulation, beeing more simple to use.

Slamax WebDk manages safely, infact, Slamax WebDK can only preload files, for having it going inside your application, and classes and controls, asshures the correct usage of those functionalities.

Kernel file management functions

function **getFileFromDirectory**

Retrieves a file from a directory

Requires the addres of the file as an object of type Url,

the file name as a string,

and the file extension as a string.

```
Kernel::getFileFromDirectory(Url $u, string $nomeFile, string $ext='txt');
```

function **deleteFileInDirectory**

Deletes a file in a directory.

Requires the addres of the file as an object of type Url,

the file name as a string,

and the file extension as a string.

```
Kernel::deleteFileInDirectory(Url $u, string $nomeFile, string $ext='txt');
```

function **moveFileToDirectory**

Moves a file in a directory.

Requires the addres of the file as an object of type Url,

the file name as a string,

and the file extension as a string.

```
Kernel::moveFileToDirectory(Url $destination, Url $u, string $nomeFile, string $ext='txt');
```

function **saveFileInDirectory**

Creates a file in a directory.

Requires the addres of the file as an object of type Url,

the file name as a string,

and the file extension as a string.

```
Kernel::saveFileInDirectory(Url $destination, Url $u, string $nomeFile, string $ext='txt');
```

function **overwriteFileInDirectory**

Overwrits a file in a directory.

Requires the addres of the file as an object of type Url,

the file name as a string,

and the file extension as a string.

```
Kernel::overwriteFileInDirectory(Url $destination, Url $u, string $nomeFile, string $ext='txt');
```

class **IniFile**

creates a file ini object, can be converte ad string and as a SlamaxObject.

```
$myUrl = new IniFile(Url $u, string $nome);
```

function **SetContent**

Overwrites the file ini content with the given content.

Requires the new content to be setted for the file.

```
$myUrl->SetContent($c);
```

function **AddContent**

Adds the content to the ini file.

Requires the content to be add for the file.

```
$myUrl->AddContent($c) ;
```

function **ToObject**

converts this IniFile object into a DataObject.

Doesn't need any parameter.

```
$myUrl->toObject();
```

class **XmlFile**

creates an object that manages an Xml file.

```
$myXml = new XmlFile(Url $u, string $nome);
```

function **SetContent**

Overwrites the xml file content with the given content.

Requires the new content to be setted for the file.

```
$myUrl->SetContent($c);
```

function **AddContent**

Adds the content to the xml file.

Requires the content to be add for the file.

```
$myUrl->AddContent($c) ;
```

function **ToObject**

converts this XmlFile object into a DataObject.

Doesn't need any parameter.

```
$myUrl->toObject();
```

class **JsonFile**

creates an object that manages a Json file.

```
$myJson = new JsonFile(Url $u, string $nome);
```

function **SetContent**

Overwrites the json file content with the given content.

Requires the new content to be setted for the file.

```
$myUrl->SetContent($c);
```

funzione **AddContent**

Adds the content to the json file.

Requires the content to be add for the file.

```
$myUrl->AddContent($c) ;
```

function **ToObject**

converts this JsonFile object into a DataObject.

Doesn't need any parameter.

```
$myUrl->toObject();
```

class **ZipFile**

creates an object that manager a ZipFile.

```
$myZip = new ZipFile(Url $u, string $nome);
```

function **AddFiles**

Adds some files to the archirve.

Requires an array of files to add.

```
$myZip->AddFiles($files);
```

function **Unzip**

It dezips a file.

Requires the name of the file as a string,

and the destination as a string.

```
$myZip->Unzip($filename, $destination);
```

class MediaFile

creates an object that manages Media, audio or video.

```
$myMedia = new MediaFile(Url $u, string $nome);
```

function **ToTexture**

converts the FIle into an object of type Texture2d.

Doesn't need any parameter.

```
$myXml ->toTexture();
```

RENDERING

This section contains almost all rendering related classes and methods of Slamax WebDK, they are usefull to configure and change specific behavior for SlamaxObjects, and they can be required by other function.

The Slamax WebDK rendering phase comes with those steps:

- Load of all external contennt (resources,libraries,external files)

- ENTRYPOINT start for SlamaxWebDK (Template object)

- CSS properties conversion for object from inside the entry point and decoding its CSS to the CssBuffer

- Conversion of JAVASCRIPT Runtime Script of all object from inside the entry point and decoding its JAVASCRIPT to the Buffer

- Conversion of all object from inside the entry point HTML into their Html rappresentation, it write its HTML to the HtmlBuffer

- Prints to the VIEWPORT the opening of the Html with his eventual head

- Prints to the VIEWPORT the CssBuffer

- Prints to the VIEWPORT the HtmlBuffer

- Prints to the VIEWPORT the JavascriptBuffer

- Prints to the VIEWPORT the closure of theHTML

class **Transform2D**

Parent object controlling the transform properties of a Html

unified both for javascript and the Css based on current CONTEXT.

Require an Html parent reference for the __construct method.

```
$myTransform = new Transform2d($this);
```

function **init**

Intialized the object.

```
$myTransform->init($x, $y, $sX, $sy, Anchor $a=null, $r=null)
```

function **ApplyTransform**

Applies all transform properties a default Css,

and it does not have any Runtime effect.

Doesn't need any parameter.

```
$myTransform->ApplyTransform();
```

class **Anchor**

An anchor object that converts top left bottom right coordinate,

mapping them to the x and y axis.

Requires the anchor type a constant of type ANCHOR_,

the x coordinate as an integer or a string,

the y coordinat as an integer or a string.

```
$myAnchor = new Anchor(ANCHOR_TOP_LEFT, $x, $y);
```

function **Get**

Applies coordinate propriets to a CSS rappresentation.

Doesn't need any parameter.

```
$myAnchor->Get();
```

class **Css**

Creates a Css object that allows to quickly apply a style to a Html object.

It can require a selector as a string (OPTIONAL).

```
$myCss = new Css(string $selector);
```

function **addProp**

Adds any CSS type property to this buffer.

Requeires the name for the paramter as a string,

and the parameter as a valid parameter.

```
$myCss-> AddProp(string $n, $p);
```

function **flushCss**

flushes all the buffer content to the global buffer.

It can require a selector overload (OPTIONAL)

```
$myCss->flushCss(string $as);
```

function **Get**

Returns a string rappresentation of the CssObject.

Doesn't need any parameter.

```
$myCss->Get();
```

class **Brush**

Un An object that can paint a SlamaxObject for the viewport.

```
$myBrush = new Brush();
```

function **Get**

Applies its style settings a a background for the SlamaxObject, works in Runtime too.

Requires an SlamaxObject in wich to apply the brush.

```
$myBrush->Get(SlamaxObject $o);
```

class **Color** extends **Brush**

A brush parent object that work as a color, both for static and runtime context.

```
$myColor = new Color(float $r, float $g, float $b, float $a);
```

function **toHex**

Return an hexadecimal version, based on red.

Doesn't need any parameter.

```
$myColor->toHex();
```

function **Get**

Same as parent but return just the CSS rappresentation, it doesn't apply any style.

Doesn't need any parameter.

```
$myColor->Get(SlamaxObject $o);
```

class **Gradient** extends **Color**

A color object but creates a gradient

```
$myGradient = new Gradient(Color $col1, Color $col2);
```

function **Get**

Same as parent.

```
$myGradient->Get(SlamaxObject $o);
```

function **addPoint**

Adds a new point to the created gradient.

Requres the start percentage as a string,

and the color as a Color object.

```
$myGradient->addPoint($perc, Color $col)
```

class **Texture2d**

A brush based objects the return an image based on context.

```
$myTexture = new Texture2d(MediaFile $immagine);
```

function **Get**

Same as parent.

```
$myTexture->Get(SlamaxObject $o=null)
```

class **Renderer**

Hanles javavascript, WebGL, and the chache system, creating a supported renderer.

Requires a BrowserInfo object from the __construct method.

```
$myRenderer = new Renderer(BrowserInfo $info);
```

function **Render**

Renders a Template.

Requires a Template object to render.

```
$myRender->Render(Template $t);
```

function **RenderDone**

To call after rendering, it must be called to close and output buffers.

Doesn't need any parameter.

```
$myRenderer->RenderDone();
```

class **Theme**

Creates a theme object, that stores and handles complex style for templates and objects.

```
$myTheme = new Theme();
```

function **getTexture**

Returns a save Texture resurce saved for the them.

Requires the resource name as a string.

```
$myTheme::getTexture(string $name);
```

function **setTemplateNavigator**

Sets the given object to be the main manu for the Template navigation.

Requires an Html object to setted as a navigator.

```
$myTheme->setTemplateNavigator(Html &$t);
```

function **setTemplateHeader**

Sets the given object to be the Template header.

Requires an Html object to be setted as an header.

```
$myTheme->setTemplateHeader(Html &$t);
```

function **setTemplateContent**

Sets the given object to be the main Template content.

Requires a Html to be setted as the Template content.

```
$myTheme->setTemplateContent(Html &$t);
```

function **setTemplateFooter**

Sets the given object to be the Template footer.

Requires an Html object to be setted as a footer.

```
$myTheme->setTemplateFooter(Html &$t) {  $this->templateFooter = $t; }
```

function **AddStyle**

Adds any Css object as a Theme style.

Requires the styel name as a string,

and the Css object.

```
$myTheme->AddStyle(string $nomeStile, Css $style);
```

function **Apply**

Applies the Theme to the Object or to the template.

Requires a SlamaxObject in wich to apply the style.

```
$myTheme->Apply(SlamaxObject $t)
```

class **Page**

Creates a page to be used as a Template Widget. It support the automatic

title setup and has all Widget reletad events

```
$myPage = new Page();
```

FORM MANAGEMENT

Slamax WebDK offers a simple form management system, that makes easy implenting form logic in the application, with database related operations, file realted, or to interact with the VIEWPORT.

Almost all of form related functions comes with default security systems, because it has a little autovalidation stystem, ma it must be said the every application has it own flaws, so that is always need to add specific validation logic.

Kernel form-functions

function **GetBrowser**

Return an Instance of browser, works only once.

Doesn't need any parameter.

```
Kernel::GetBrowser();
```

function **receivePostDataArray**

Return post data as an ARRAY, if it as valid element inside the given array.

Requires the wanted post data as an Array.

```
Kernel::receivePostDataArray(array $wantedData);
```

function **getWebDKlocation**

Return the actual address of the Slamax WebDK.

Doesn't need any parameter.

```
Kernel::getWebDKlocation();
```

function **setWebDKlocation**

Changes the default WebDK location

Requires the new location as a string.

```
Kernel::setWebDKlocation(string $location);
```

function **receiveGetDataArray**

Return get data as an ARRAY, if it as valid element inside the given array.

Requires the wanted get data as an Array.

Kernel::receiveGetDataArray(array $wantedData);

function **receivePostData**

Returns the valid elements inside the given array from the POST method.

Requires the wanted post data as an Array.

Kernel::receivePostData(array $wantedData);

function **receiveGetData**

Returns the valid elements inside the given array from the GET method.

Requires the wanted get data as an Array.

Kernel::receiveGetData(array $wantedData);

class **Form**

Creates a Form widget that can listen and handle inputs from the user.

```
$myForm = new Form(Widget $parent, string $name, Url $action, int $formType, string $sendType);
```

function **addFormElement**

Adds an element to the Form.

Requeires the element name as a string,

and the Input object to be added.

```
$myForm->addFormElement(string $name, Input $i)
```

function **addFormMultipleElement**

Adds more elements as multiple for the form.

Requeires the element name as a string,

and the Html object conainting all data to be setted.

```
$myForm->addFormMultipleElement(string $name, Html $i)
```

function **Listen**

Listen to all Form inputs.

Doesn't need any parameter.

```
$myForm->Listen();
```

class **Input**

Creates a Widget that handles user Input.

```
$myInput = new Input(string $defaultValue, string $name, bool $hasPlaceHolder, int $type);
```

function **Listen**

Listens to the input and calls its callback function.

Requires a callback function as a callable,

and the listen method as a string.

```
$myInput->Listen(callable $f,string $type='POST');
```

class **SelectInput**

Create a new input of type Select, same as input.

```
$mySelectInput = new SelectInput(string $defaultValue, string $name, bool $hasPlaceHolder, int $type);
```

class **HiddenInput**

Creates a new Input of type Hidden, same as input

```
$myHiddenInput = new HiddenInput(string $defaultValue, string $name, bool $hasPlaceHolder, int $type);
```

class **CheckInput**

Creates a new input of type Check, same as input.

```
$myCheckInput = new CheckInput(string $defaultValue, string $name, bool $hasPlaceHolder, int $type);
```

class **TextInput**

Creates a new Input of type text, same as Input.

```
$myTextInput = new TextInput(string $defaultValue, string $name, bool $hasPlaceHolder, int $type);
```

class **NumberInput**

Creates a new Input of type Number, same as input.

```
$myNumberInput = new NumberInput(string $defaultValue, string $name, bool $hasPlaceHolder, int $type);
```

class **DateInput**

Creates a new Input of type date, same as Input.

```
$myDateInput = new DateInput(string $defaultValue, string $name, bool $hasPlaceHolder, int $type);
```

class **FileInput**

Creates a new Input of type File, same as Input.

```
$myFileInput = new FileInput(string $defaultValue, string $name, bool $hasPlaceHolder, int $type);
```

class **ImageInput**

Creates an Input of type Image, same as input.

```
$myImageInput = new ImageInput(string $defaultValue, string $name, bool $hasPlaceHolder, int $type);
```

class **TextAreaInput**

Creates an Input of type Textarea, same as Input.

```
$myTextAreaInput = new TextAreaInput(string $defaultValue, string $name, bool $hasPlaceHolder, int $type);
```

class **Button**

Creates a Button Widget, in can be use to submit forms.

```
$myButton = new Button(string $name, string $value);
```

OGGETTI

Slamax WebDK objects, are all that you need to begin using it! Slamax Objects can be used even standlaon, but if they are use outside a Template class, it will be ignored any Runtime Script.

The Template it's the most important for all the library, and it is define from the library ENTRY POINT, so that it can configure all settings that enables every functionaly for the Slamax WebDK.

If you are trying not using the Template class, you have to carefully read the documentation to recreate a proper impleamentation, and you have to own the source code, you must be very carefull to prevent wrong Slamax WebDK configuraations that can break functionalities.

class **Template**

Creates a Template object, wich rappresent the ENTRYPOINT of Slamax WebDK, it is suggested to create a new class extendsìing Template and call it like your app.

The Template class handles theme and resources for the application, and the layout.

```
$myTemplate = new Template();
```

function **Get**

Renders the template.

Doesn't need any parameter.

```
$myTemplate-> Get() ;
```

function **AddTheme**

Adds a Theme to the Template.

Requires a Theme object to be added.

```
$myTemplate->AddTheme(Theme $theme) ;
```

function **getTheme**

Return a theme setted in the Template.

Requires the Theme indexd as an integer.

```
$myTemplate->getTheme(int $index);
```

class **Tag**

Creates a new object capable of generation a XML Tag,

to be inserted in a buffered or to be returned as string.

```
$myTag = new Tag(string $name, $cont);
```

function **addContent**

Aggiunge un contenuto stringa al Tag, non accetta classi convertibili in stringa

Richiede il contenuto da aggiunger al tag come string.

```
$myTag->addContent(string $c)
```

class **Html**

Creates a new Tag of type xHtml that has Css capabilities.

```
$myHtml = new Html($cont, string $id, string $class, string $name);
```

function **addLink**

Adds a link to as href paramter for the tag.

Doesn't need any parameter.

```
$myHtml->addLink(Url $link, string $title='')
```

class **SlamaxObject**

Creates a SlamaxObject, an autocompiled multilanguage object that writes JAVASCRIPT, CSS and HTML, and it is the core of Slamax WebDK.

```
$myObject = new SlamaxObject($cont, string $id, bool $isDinamic);
```

function **Show**

Shows this object to the current VIEWPORT.

Doesn't need any parameter.

```
$myObject->Show();
```

function **Hide**

Hides this object from the current VIEWPORT.

Doesn't need any parameter.

```
$myObject->Hide();
```

function **Toggle**

Shows or hides an object in the VIEWPORT based on his current visibility.

Doesn't need any parameter.

```
$myObject->Toggle();
```

function **RemoveBackground**

Resets the background style, removing it.

Doesn't need any parameter.

```
$myObject->RemoveBackground();
```

function **RemoveBorder**

Removes the object border.

Doesn't need any parameter.

```
$myObject->RemoveBorder();
```

function **Center**

Centers the obect into his parent.

Doesn't need any parameter.

```
$myObject->Center();
```

function **SetPadding**

Changes the object padding.

Requires topLeft, topRight, bottomLeft, bottoRight.

```
$myObject->SetPadding($tl, $tr, $bl, $br);
```

function **SetMargin**

Changes the object margins.

Requires marginLeft, marginRight, bottomLeft, bottoRight.

```
$myObject->SetMargin($ml, $mr, $bl, $br);
```

function **SmoothBorders**

Smooths the object borders.

Requires topLeft, topRight, bottomLeft, bottoRight.

```
$myObject->SmoothBorders($ml, $mr, $bl, $br);
```

function **SetTextColor**

Change the object text color.

Requires a Color object.

```
$myObject->SetTextColor (Colore $col);
```

function **AddBackground**

Aggiunge uno sfondo all'oggetto

Requires a Brush object.

```
$myObject->AddBackground(Brush $brush)
```

function **AddBorder**

Adds a border to the object.

Requires the width as a string , a Color object,

and the style flag as CSS_ constant (OPTIONAL)

```
$myObject->AddBorder($gros, Color $col, $stile);
```

function **AddShadow**

Adds a shadow to the object.

Requires the width as a string , a Color object,

and the inner flag as CSS_ constant (OPTIONAL)

```
$myObject->AddShadow($gros , Colore $col, $inner);
```

function **changeFont**

Changes the text Font for the object.

Requires a Font object.

```
$myObject->changeFont(Font $font)
```

function **changeAnchor**

Changes the object anchor.

Requres the new Anchor as a Anchor object.

```
$myObject->changeAnchor(Anchor $a) ;
```

function **changeOpacity**

Changes the object opacity.

Requires the opacity as a float.

```
$myObject->changeOpacity(float $op) ;
```

function **resize**

Change the object size.

Requires the width, the heigth and the volume.

```
$myObject->resize($w, $h, $v);
```

function **move**

Moves the object of the specified units.

Requires the width, the heigth and the volume.

```
$myObject->move($x, $y, $z);
```

function **getSelfReference**

Get it own refernece, as a RUNTIME ACTOR.

Doesn't need any parameter.

```
$myObject->getSelfReference()
```

function **getSelfNode**

Gets a reference of the JAVASCRIPT node for the RUNTIME ACTOR.

Doesn't need any parameter.

```
$myObject->getSelfNode();
```

class **Widget**

Creates a SlamaxObject of type Widget.

```
$myWidget = new Widget();
```

function **applyTheme**

Appllies a the Theme to the Widget.

Requires a Theme object to be added.

```
$myWidget->applyTheme(Theme $t);
```

function **getChildren**

Return a parent children from the given index.

Requires the parent name as an integer.

```
$myWidget->GetChildren($index=0);
```

function **getAllChildrens**

Return all Children widgets.

Doesn't need any parameter.

```
$myWidget->GetAllChildrens();
```

function **RemoveChild**

Removes a child Widget at the given index.

Requires the index of the widget to be removed.

```
$myWidget->RemoveChild($index=0);
```

function **addText**

Adds Text to Widget content.

Requires the text to be added as a string.

```
$myWidget->AddText(string $text);
```

class **Media**

Creates a Media object, to import pictures, audio and video.

Doesn't support streaming rigth now.

```
$myMedia = new Media(MediaFile $f, int $tp);
```

class **Image** extends **Media**

Creates an Image MediaFile.

```
$myImage = new Image(MediaFile $f, string $slogan, int $size);
```

class **Audio** extends **Media**

Creates an Audio MediaFile.

```
$myAudio = new Audio(MediaFile $f, string $slogan, int $size);
```

class **Video** extends **Media**

Creates a Video MediaFile.

```
$myVideo = new Video(MediaFile $f, string $slogan, int $size);
```

class **Logo** extends **Media**

Creates a Logo MediaFile, width a slogan an Image to be setted.

```
$myLogo = new Logo(MediaFile $f, string $slogan, int $size);
```

class **Background**

Creates a Background Widget, able to manage size and position parameters.

```
$myBackground = new Background(Texture2d $image, int $hRepeat, int $vRepeat, float $xOffset, float $yOffset, int $repetitionType);
```

class **ItemList**

Creates content List Widget

```
$myItemList = new ItemList();
```

function **addToList**

Adds the content to the list

Requires any content that can be casted as a string.

```
$myList-> addToList(string $content)
```

class **Window**

Creates a window Widget, it comes with size title, content,

his style and colors

```
$myWIndow = new Window($w, $h, string $title, int $titleH=20, int $x=0, int $y=0, Brush $backCol = null, Brush $titCol = null, Color $titTextCol = null, Color $textCol=null , bool $draggable=false);
```

function **AddToWindow**

Adds any elemnt to the window

Requires any content that can be casted as a string.

```
$myWindow->AddToWindow(string $content)
```

class **TableBox**

Creates a Table Widget to orderly set the content, visually.

```
$myTable = new TableBox(string $titleID, array $names);
```

function **setMaxElement**

Set the Max element to allow a greater number of columns.

Requires the new max element as an integer.

```
$myTable->setMaxElement(int $e)
```

function **addPagination**

Adds a pagination usefull to navigate width complex tables.

Requires the element number as an integer,

the start of the pagination as an integer,

the last pagination page as an integer,

the main page addres as an Url object (OPTIONAL),

requires the current page as an integer (OPTIONAL),

the Css pagination id as a string (OPTIONAL).

```
$myTable->addPagination(int $totElementi=10,int $from=0, int $to=10, Url $url=null, int $corrente=1, string $pgId='pageN')
```

funzione **AddRow**

Adds an array as cell elements, for a new table row.

Requires an array object of cells,

and a boolean check to know if it is a title row (OPTIONAL).

```
$myTable->addRow(array $cells, $isTitle=false);
```

class **Separator**

Creates a new content separator Widget.

```
$mySeparator = new Separator();
```

class **Grid**

Creates a grid uses the table as a the main, and can handle elements width a grid system.

```
$myGrid = new Grid(int $rows, int $cols)
```

function **addToGrid**

Adds an element to the grid, within the given coordinates..

Requires the Y coordinate as an integer,

and the X coordinate as an integer,

and the Widget object to be inserted in the grid.

```
$grid->addToGrid(int $row, int $col, Widget $widget);
```

class **SizeBox**

Creates a new SizeBox Widget, ita can handle size, width, text color, background color.

```
$mySizeBox = new SizeBox(Widget $content=null, $w, $h, Color $col=null, Color $backg=null);
```

class **Curve**

Creates a new Curve object, returns a 2d vector collection, useful for animations.

```
$myCurve = new Curve(CurvePoint $start, CurvePoint $end,$minStren=0.0,$maxStren=5.0);
```

function **Get**

Abilita la curva scelta

Non richiede alcun tipo di parametro.

```
$myCurve->Get()
```

function **addControlPoint**

Aggiunge un Punto di Controllo alla Curva.

Richiede un oggetto di tipo curvePoint come punto di controllo.

```
$myCurve->AddControlPoint(curvePoint $vec);
```

SCRIPTING

Slamax WebDK offer even scripting functionalities thanks to the runtime scripts, that are special function tha automatically compose JAVASCRIPT code by saving it in into a buffer, that allows to write JAVASCRIPT code without the need to switch to a diffrent programming environment or files.

There are also avaible even classes and objects that call automatically other function for more complex functionalities.

class **CurvePoint**

Creates a 2d vector Point for the curve.

```
$myCurvePoint = new CurvePoint($x, $y);
```

function **Get**

Return a new reference for the curve.

It doesn't require any parameter.

```
$myCurvePoint->Get();
```

class **Timeline**

Creates a Timeline object that allows to

easily animate objects of the page.

```
$myTimeline = new Timeline(int $time, Curve $curve, $ref, string $name);
```

function **setCompletedFunction**

Sets the function to be called in parent as a string name.

Requires the name of the function as a string.

```
$myTimeline->setCompletedFunction(string $fun);
```

function **setLoopFunction**

Sets the function to be called in the loop.

Requires the name of the function as a string.

```
$myTimeline->setLoopFunction(string $fun);
```

function **Play**

Activates the Timeline and calls its events.

Requires the startime in seconds as an integer, and the endtime in seconds as integer.

```
$myTimeline->Play($startTime, $endTime)  ;
```

function **Stop**

Stops the timeline execuition.

It doesn't need any parameter.

```
$myTimeline->Stop();
```

FUNZIONI DELLA RUNTIME

Pure JAVASCRIPT calls given by Slamax WebDK.

function **createVariable**

creates a new Runtime variable.

```
createVariable($v, string $name='')
```

function **createArray**

creates a new array Runtime variable.

```
createArray(string $size='0', string $name='')
```

function **exitWithValue**

Exits from the runtime Script, with a value for the caller

```
exitWithValue($ret)
```

function **getElementScroll**

Return the scroll value of an element.

Requires the node Reference of the object.

```
getElementScroll(string $node)
```

function **switchParameter**

Prepare a switchCase for the runtime.

```
switchParameter(&$switch)
```

function **ForLoop**

Simple for loop to be execute during the runtime.

ForLoop($min, $max, callable $body)

function **createTimeline**

Creates a timeline for the animations in the runtime.

createTimeline(&$varRef, $startTime, $endTime, Curve $curve, string $updateFun, string $completeFun, string $name='')

function **addClass**

Adds a value to class attribute, of the given reference.

addClass(JsVar $ref, $name)

function **removeClass**

Removes a class attribute value, from the passed reference

removeClass(JsVar $ref, $name)

function **getPageScroll**

Retunr the scroll value tof the page at runtime.

getPageScroll()

function **setPageScroll**

Set the page scroll to the given value, at runtime.

setPageScroll($value)

function **lerp**

Interpolates the given value, based on the minimum the maximum, and alpha.

lerp($j,float $min,float $max,float $a)

SLAMAX SDK EVENTS

Slamax WebDK works thanks to event driven system, the events are dispatched from widthin the ENTRYPOINT, from the Event Manager of each object based on the type of interaction that can depend on one or more condition.

The events are usefull to separate the logic of the code in easy to remeber section based on actual context.

The events can be Runtime (Client Side) e Server (Server Side). The Runtime events are real JAVASCRIPT function that are called by the Task Looper of Slamax WebDK, by following the initializiation order (eventSetup), and after that it gets called the visualization (eventBeginView), and the evente for the update (eventTick) gets called many times each second but only after the event Setup gets succefuly called.

class **Event**

UAn Event for a slamax Object, it will be called by the EventManager,

if conditions are met.

Requires the event name as a string,

the callback function name as a string,

A reference to the EventListener..

```
$myEvent = new Event(string $event, string $funcName, EventListener $evL);
```

function **Release**

Forces the automatic execution of the callback function.

It doesn't need any parameter.

```
$myEvent->Release();
```

SLAMAX OBJECT RUNTIME EVENTS

By extending any SlamaxObject you can script the client side code of the object/plugin that you wanna create.

All widgets are SlamaxObjects.

A function Looper very specialized receives the Actor Instances that contains any script call, and compose their runtime code.

E' possible to create custom functions and classes with the Runtime Script functions.

Event **Setup**

This event gets called one Time before any other event.

```
class MyObject extends SlamaxObject {

    function eventSetup() {  ...  }

}
```

Event **onBeginView**

this event gets called one time after the setup.

```
class MyObject extends SlamaxObject {

    function eventOnBeginView() {  ... }

}
```

Event **Tick**

This event gets called many time per sceond based on browser framerate

and the machine configuration.

It gives you a delta to make animations time indipendent.

```
class MyObject extends SlamaxObject {
	function eventTick($delta) {  ... }
}
```

Event **Click**

This events called whenwher a clicked has been made to the object,

only if click is enabled.

```
class MyObject extends SlamaxObject {
	function eventClick() {  ...  }
}
```

Event **Hover**

This events gets called every time you hover the object,

but only if hovering is enabled.

```
class MyObject extends SlamaxObject {
	function eventHover() { ... }
}
```

Event **onRelease**

This event get called whenewere a clicked object gets released,

but only if the click is enabled.

```
class MyObject extends SlamaxObject {
    function eventOnRelease() { ... }
}
```

Event **AjaxResponse**

```
class MyObject extends SlamaxObject {
    function eventAjaxResponse(DataObject $data) {  ...  }
}
```

SLAMAX OBJECTS SERVER EVENTS

The server events are generated as a response for specific function when called.

Event **FormSubmit**

This event get called when you caall Listen() on any Form object.

It gives valid data from the POST method, with a DatabObject of type

FormData.

```
class MyObject extends SlamaxObject {
    function eventFormSubmit(FormData $data) {  ...  }
}
```

Event **DbResponse**

This event called whenewher you perform a database related action or manipulation.

It gives you a DataObject of type SqlData that contains the response data.

```
class MyObject extends SlamaxObject {
    function eventDbResponse(SqlData $data) {  ...  }
}
```

SLAMAX OBJECTS SPECIAL FUNCTIONS

The SlamaxObject special function allows you to go for more complex functionality or settings easily, with the confort of the event to manage the success or failure for the call.

Every special function fires an Event.

function **RequireDataFromDb**

Executes a Sql request to the database setted for the connection,

Fires an Event giving a SqlData with response results.

Requires the table in witch to retrieve the data as a string,

requires the data to be getted as an array,

and the selection condition as an object of type SqlCond (OPTIONAL),

and the search limit to paginate the results of an object of type SqlLimit (OPTIONAL).

```
$myObject->RequireDataFromDb($table, $dati,  SqlCond $sCond, SqlLimit $lim)
```

function **SendDataToDb**

Executes a sql request to send new data to the connected database.

Fires an Event giving a SqlData with response results.

Requires the table in witch to send the data as a string,

requires the data to be inserted as an array,

```
$myObject->SendDataToDb(string $table, array $data)
```

function **UpdateDataOnDb**

Executes a Sql to update aready present data into the connect dataase,

Fires an Event giving a SqlData with response results.

Requires the table in witch to send the data as a string,

requires the data to be inserted as a DataObject,

the id of data to update as an integer.

```
$myObject->UpdateDataOnDb(string $table, DataObject $data, int $id)
```

function **DeleteDataOnDb**

Executes a Sql requrest to delete already present data from the connected database,

Fires an Event giving a SqlData with response results.

Requires the table in witch to send the data as a string,

requires id of the element to delete as integer,

and the new max Element for the PRIMARY_KEY incrementer as integer (OPTIONAL).

```
$myObject->DeleteDataOnDb(string $table, int $id, int $maxEl)
```

USEFULL CONSTANTS FOR SLAMAX WEBDK

Slamax WebDk comes with usefull constans for converting values or for matting automaically properties, or to change some settings for some rendering related classses.

Slamax WebDK constante can be used only in theri specicic places that can be:

- As a parameter for a specific class function

- From an assignment / setting of a property of an object.

- As a conjugate between 1 or 2 SlamaxObjects (by casting them into strings)

Those constants must be used respecting their functionality, for example, it's higly discourage using a constant as a setting, ora a Css constant, as conjugate.

Other uses of Slamax WebDK constants are to be considered faulty, because the can give bugs e corrupt your application, so their use are not supported outside those cases.

BROWSER DOCTYPE'S CONSTANTS

Those kind of constants must be defined at the tmeplate initilaziation, by calling the function from the Browser wich changes the doctype.

HTML_5	html 5 doctype
STRICT_4_01	html 4.01 strict doctype
TRANSITIONAL_4_01	html 4.01 strinct doctype
FRAME_SET_4_01	html 4.01 FrameSet doctype
XHTML_1_0	xhtml 1.0 doctype
XHTML_1_0_STRICT	xhtml 1.0 strict doctype
XHTML_1_0_TRANSITIONAL	xhtml 1.0 transitional doctype
XHTML_1_0_FRAMESET	xhtml 1.0 FrameSet doctype
XHTML_1_1	xhtml 1.1 doctype

ANCHOR SETTINGS CONSTANTS

Those constants changes an object anchorage, must be given to class Anchor or to __construct method o with the Set() function.

ANCHOR_TOPLEFT	imposta l'ancora in alto a sinistra
ANCHOR_TOPRIGHT	imposta l'ancora in alto a destra
ANCHOR_BOTTOMLEFT	imposta l'ancora in basso a destra
ANCHOR_BOTTOMRIGHT	imposta l'ancora in basso a destra

SCRIPTING CONSTANTS

Those constants are usefull for Runtime Scripts and CSS properties, use them carefully based on the context.

(j suffix stands for the fact works only as a runtime property)

pX	Converts the last float to a pixel unit
perc	Converts the last float to a percent unit
em	Converts the last float to a em unit
jPX	Same as pX but only Runtime
jPerc	Same as perc but only Runtime

_Force	Forces the propriety to be higher in hierarchy.
_none	Sets to none the property
AUTO	Sets the property to have the default Value
_€	Euro symbol
_Copy	Copyrigth symbol
NL	Makes a carriage return

B_SOLID	Set the property to be a solid brush

D_COLUMN	set the property as a column
D_LEFT	sets the property as left
D_RIGHT	sets the property as rigth
D_TOP	sets the property as top
D_CENTER	sets the property as center
D_BOTTOM	sets the property as bottom

V_BLOCK	sets the property as block
V_COVER	sets the property as cover
V_FLEX	sets the property as flex
V_HIDDEN	sets the property as hidden
V_SHOWEN	sets the property as showen
V_INLINEBLOCK	sets the property as inline-block
V_POINTER	sets the property as pointer
V_SCROLL	sets the property as scroll

P_ABSOLUTE	sets the property as absolute
P_RELATIVE	sets the property as relative
P_FIXED	sets the property as fixed

F_BOLD	sets the property as bold
F_ITALIC	sets the property as italic
F_NORMAL	sets the property as normal
F_UPPERCASE	sets the property as uppercase

DI Slamax WebDK ERRORS

Slamax WebDK occurs because there a mismatched setting of the librart Slamax WebDK or when bugs are present in the current version.

If you find any of those errors when programming with Slamax WebDK, do not wait to contact our support for an eventual clarification, we will respond very please as soon as possible!

Error **UNDEFINEDerror**

Fatal Error, it must never pop out, if pops out it indicates that or a bug is present in this library version, or your library has been fatally corrupted, feel free to contact the support.

Error **SdkInitalizationViolationError**

Fatal Error, it pops out when one or more configuration, is not correctly setted for the Slamax WebDK, see Setup and Intruduction to Slamax WebDK.

Error **SdkSECURITYerror**

Fatal Error, this error pops out when one or more functions has violated the security system whe calling Slamax WebDK function, find eventual bugs present in your application, and/or change password and access data for your administrator profile, it's infact likely to be also an Hacking attack.

Error **SdkCOREerror**

Fatal Error, this error pops out when one or function, calls incorrectly other function of Slamax WebDK, find eventual bugs present in your application.

Error **SdkAPPLICATIONerror**

Fatal error, this is custom settable fatal error, for your application.

SLAMAX WEBDK WARNINGS

A Slamax WebDK warning pops out when it a potential not seen broken logic, or more ambiguos parameters inside the code, but it's not a real error.

Se

If you find this kind of warning, check your code or simply ignore the warning, it will not pop out in the production stage.

Warning **SdkCOREwarning**

Warning, one or more function called can product non consistent result.

It will not popout in a production envinorment.

Warning **SdkAPPLICATIONwarning**

A custom Warning settable by the application.

ESEMPI E TUTORIAL PER Slamax WebDK

If you have alreay bouth a copy of Slamax WebDK, inside the example folder, you will find examples and tutorial simple to follow tutorial, to begin exploring all the potential of this library!

In the nexte page, is riporte an example not present inside the example folder, it is the most minimal code to begin programming your web application with Slamax WebDK.

```
class MyPage extends Page {
	/**  EVENT OnBeginView **/
	function eventOnBeginView() {
		/**	IMPLEMENT HERE RUNTIME LOGIC
		FOR THE CURRENT PAGE	**/
		//prints the text "Hi from Javascript" to the browser console
		Console::log('Hi from Javascript Console!');
	}
	function OnCreate() {
		/**	IMPLEMENT HERE SERVER LOGIC FOR THE CURRENT PAGE  **/
		//prints the text "hello world" to the page
		$this->addContent('Hello World!');
	}
}
class MyTemplate extends Template {
	/**  ENTRY POINT **/
	function OnCreate() {
		/**	IMPLEMENT HERE THE SERVER LOGIC TO CHOOSE THE
			CURRENT PAGE
			ex: if ($_GET['page']==1) { $this->currentPage = new MyPage1(); }   **/
		$this->currentPage=new MyPage();
	}
}
new MyTemplate();
```