Mécanismes

Table des matières

1. Contenu

Ce document à pour but d'expliquer différents mécanismes de fonctionnement implémentés en standard dans **UniversalWeb**.

2. Mécanisme d'authentification

UniversalWeb propose par défaut sur son backend un mécanisme d'authentification basé sur son annuaire interne (table **`_users`** en base de données). Pour cela il fait appel à la classe **`Login`**. Chaque page du backend devant être protégé par ce mécanisme d'authentification, celui-ci est appelé depuis le script **`init.inc.php`** lancé au début de chaque script.

L'objet **`login()`** instancié de la classe **`Login`** est sauvegardé en session dans la variable **`_APP_LOGIN_`**. Noter qu'en cas d'utilisation d'un annuaire externe, cette variable de session doit recevoir non pas un objet **`login()`** mais un objet instancié de la classe particulière à l'annuaire externe (non fourni).

```
//---------------------------------------------------------------------
// Creation et mise en session de l'objet _APP_LOGIN_ qui contient les infos de l'utilisateur logué
// et prise en compte de la langue en cours de l'utilisateur
//---------------------------------------------------------------------
//$_SESSION[_APP_LOGIN_] = null;
if  ((!isset($_SESSION[_APP_LOGIN_])) || ($_SESSION[_APP_LOGIN_] == null)) {
   if (_ANNUAIRE_ == _ANNUAIRE_INTERNE_) $_SESSION[_APP_LOGIN_] = new Login();
   //incrire d'autres éventuels annuaires ici
   //...
   $_SESSION[_APP_LANGUE_ENCOURS_] = $_SESSION[_APP_LOGIN_]->getLanguePref();
}
```

Il suffit ensuite de vérifier si l'utilisateur est connecté (appel de la méthode **`isLogged()`**). S'il l'est, le script se déroule normalement. Dans le cas contraire **`init.inc.php`** redirige vers le script de connexion.

```
//---------------------------------------------------------------------
//test si utilisateur connecté, sinon renvoi phase de connexion
//---------------------------------------------------------------------
if (!$_SESSION[_APP_LOGIN_]->isLogged() && (basename($_SERVER['PHP_SELF']) != _URL_AUTHENTIFICATION_)) {
   header('Location: '._URL_BASE_SITE_._URL_AUTHENTIFICATION_);
}
```

Ce mécanisme pourra être utilisé tel quel dans le cadre du frontend si votre projet le nécessite.

3. Mécanisme de gestion des langues

UniversalWeb est par défaut multilingue, c'est-à-dire que votre application peut être proposée dans plusieurs langues. C'est ce que vous pouvez découvrir dans le backend. Les langues proposées en standard sont le français (par défaut, fr) et l'anglais (en).

Tout tourne autour de l'inclusion du fichier de langue adéquat parmi ceux présents dans le dossier **`libs/langues/`** et par l'appel à la fonction **`getLib()`** qui y est implémenté qui se charge de la traduction d'une définition compacte en langage réel. Chaque nom de fichier de langue doit être ainsi structuré :

langue_[ISO 3166 alpha2].inc.php

C'est dans le fichier **init.inc.php** qu'est décidé quel fichier de langue charger. Pour ceci UniversalWeb utilise deux variables de session :

_APP_LANGUE_ENCOURS_ : renseigne sur la langue actuellement utilisée.
_APP_LANGUE_CHANGEE_ : flag sur la demande récente de changement de langue.

Voici le code actuel.

```
…
//---------------------------------------------------------------------
// Choix de la langue
//---------------------------------------------------------------------
//langue en cours par défaut est le français 'fr'
if (empty($_SESSION[_APP_LANGUE_ENCOURS_])) {
   $_SESSION[_APP_LANGUE_ENCOURS_] = 'fr';
}
if ((isset($_SESSION[_APP_LANGUE_CHANGEE_])) && ($_SESSION[_APP_LANGUE_CHANGEE_] !=
$_SESSION[_APP_LANGUE_ENCOURS_])) {
   //modification de la langue choisie par l'utilisateur dans l'objet _APP_LOGIN_
   $_SESSION[_APP_LOGIN_]->changeLangue($_SESSION[_APP_LANGUE_CHANGEE_]);
   //nouvelle langue en cours prise en compte
   $_SESSION[_APP_LANGUE_ENCOURS_] = $_SESSION[_APP_LANGUE_CHANGEE_];
   //reset de l'info de changement de langue
   unset($_SESSION[_APP_LANGUE_CHANGEE_]);
}
//chargement du fichier de langue adéquate
if ($_SESSION[_APP_LANGUE_ENCOURS_] == 'fr') {
   require_once(_LANGUES_.'langue_fr.inc.php');
}
elseif ($_SESSION[_APP_LANGUE_ENCOURS_] == 'en') {
   require_once(_LANGUES_.'langue_us.inc.php');
}
…
```

init.inc.php

Bien entendu, en cas de davantage de langues, il sera nécessaire d'adapter ce code à vos besoins.

Il suffit alors de déclencher la demande de changement de langue (modification de la variable de session **_APP_LANGUE_CHANGEE_)** depuis un lien particulier, par exemple (cas du backend), un clic sur le drapeau correspondant via appel au script **langue.php**.

```
…
echo '<a href="'._URL_BASE_SITE_._URL_LANGUE_.'?langue=fr">';
   echo '<img class="mb-1"src="'._DRAPEAU_FR_.'" alt="" title="" />';
echo '</a>';
echo ' | ';
echo '<a href="'._URL_BASE_SITE_._URL_LANGUE_.'?langue=en">';
   echo '<img class="mb-1" src="'._DRAPEAU_US_.'" alt="" title="" />';
echo '</a>';
…
```

brique_header.inc.php

```
<?php
//---------------------------------------------------------------------
// CHANGEMENT DE LANGUE
//---------------------------------------------------------------------
// éè : pour enregistrement UTF-8
//---------------------------------------------------------------------
require_once('libs/common.inc.php');

if (isset($_GET['langue'])) {
   $_SESSION[_APP_LANGUE_CHANGEE_] = MySQLDataProtect($_GET['langue']);
}
//raffraichissement de la page
goReferer();
```

langue.php

Chaque fichier de langue est constitué de quelques définitions qu'il est nécessaire de conserver et d'adapter pour chaque langue. Voici ces définitions :

Nom de la constante	Valeur
_LG_	code ISO de la langue
_FR_	code ISO du français
_EN_	code ISO de l'anglais
INT	code spécifique indiquant toutes les langue (ou aucune en particulier)
_FORMAT_DATE_TIME_SQL_	format du datetime pour MySQL
_FORMAT_DATE_SQL_	format date pour MySQL
_FORMAT_DATE_	format de la date dans la langue
_FORMAT_DATE_TIME_	format du datetime dans la langue
_FORMAT_DATE_TIME_NOS_	format du datetime sans les secondes dans la langue
_IMAGES_LANGUE_	chemin spécifiques vers les images propres à la langue

… et de quelques variables globales :

Nom de la variable	Valeur
$_MOIS_EN_CLAIR	Tableau ordonnée de valeurs des mois de l'année (ex : '', 'janvier', 'février', 'mars', 'avril', 'mai', 'juin', 'juillet', 'août', 'septembre', 'octobre', 'novembre', 'décembre') pour la langue française.
$_CIVILITE	Tableau ordonné de dénominations selon le sexe d'un utilisateur (ex : '', 'M.', 'Mme', 'Melle', 'Inconnue') pour la langue française.
$_LIBELLES	Cœur du processus, tableau de mnémoniques et de leur traduction dans la langue considérée. C'est aux fonctions **getLib()** et **getLibUpper()** que revient la tache de la traduction en fonction de la définition passé en paramètre. Ex : **getLib**('SAUVEGARDE_BD') renvoie la chaine suivante en langue française : 'Sauvegarde de la base de données'.

fonction getLib()

Traduit une définition compacte dans la langue spécifié du fichier **langue_[ISO 3166 alpha2].inc.php** chargé.

Description

```
string getLib(string $definition [, string $param1][, string $param2][,
string $param3][, string $param4][, string $param5]);
```

Liste des paramètres

$definition : définition compacte faisant référence à la phrase à afficher. Cette définition correspond à une entrée à choisir dans le tableau **$_LIBELLES** qui contient l'ensemble des définitions linguistiques de l'application.

$param1 : paramètre optionnel n°1.
$param2 : paramètre optionnel n°2.
$param3 : paramètre optionnel n°3.
$param4 : paramètre optionnel n°4.
$param5 : paramètre optionnel n°5.

Valeurs de retour
La chaine de caractère affichée dans la langue du fichier de langue chargé. Si la définition est absente, la fonction affiche **TEXTE ABSENT**.

Exemple

cas d'une définition sans paramètre
```
$chaine = getLib('DERNIER_ACCES');
avec la définition suivante : 'DERNIER_ACCES' => 'Dernier accés'
```

cas d'une définition avec un seul paramètre
```
$chaine = getLib('TODAY_IS', date('Y-m-d'));   //cas d'une définition avec 1 seule paramètre
avec la définition suivante : 'TODAY_IS' => 'Nous sommes le %s'
```

cas d'une définition avec 2 paramètres
```
$chaine = getLib('BIENVENUE_NOM_PRENOM', $nom, $prenom);   //cas d'une définition avec 2 paramètres
avec la définition suivante : 'BIENVENUE_NOM_PRENOM' => 'Bienvenue %s %s'
```

Note
Attention : si le nombre de paramètre diffère du nombre de *placeholders* dans la définition, le texte ne sera pas affiché convenablement.
Si la définition n'existe pas dans le tableau **$_LIBELLES**, la fonction affichera **TEXTE ABSENT**.
Cette méthode est une simple mise en œuvre de la fonction PHP **sprintf** avec toutes les possibilités que cela implique.

4. Mécanisme de debugge

UniversalWeb propose un mécanisme qui permet au développeur d'afficher des informations de debugge sur l'application en cours. Il s'agit en fait d'un buffer qui emmagasine les traces disposées par le développeur au fil de l'application et qui sont restituées en une fois via la brique **brique_debug.inc.php**. Pour information, cette brique est affichée juste avant le footer dans les squelettes frontend et backend proposés par UniversalWeb.

Par exemple l'affichage suivant…

…est le résultat de la demande du développeur via l'appel à la fonction

```
DEBUG_('titre de l\'application', _APP_TITLE_);
```

fonction DEBUG_()

Mise en place d'une information de debugge.

Description
```
DEBUG_(string $libelle [, mixed $variable]);
```

Liste des paramètres
libelle : libellé en clair pour représenter l'information à afficher
variable : variable à afficher. Ce peut être une simple variable (entier, booléen, etc.), un objet ou un tableau. Ce paramètre est optionnel. Dans ce cas, l'information affichée se limitera au libellé.

Valeurs de retour
Aucun. L'information est mise en buffer et sera affiché plus tard via l'appel à la fonction **DEBUG_PRINT_()**;

Exemple
```
DEBUG_('titre de l\'application', _APP_TITLE_);
```

Note
Le buffer discuté est en réalité une variable de session : _APP_ID_.'DEBUG_PHP'.

fonction DEBUG_PRINT_()

Affichage des informations de debugge.

Description
```
DEBUG_PRINT_();
```

Liste des paramètres
Aucun

Valeurs de retour
Aucun. La fonction se limite à l'affichage du buffer contenant les entrées à débugger avant de le vider.

Exemple
```
DEBUG_PRINT_();
```

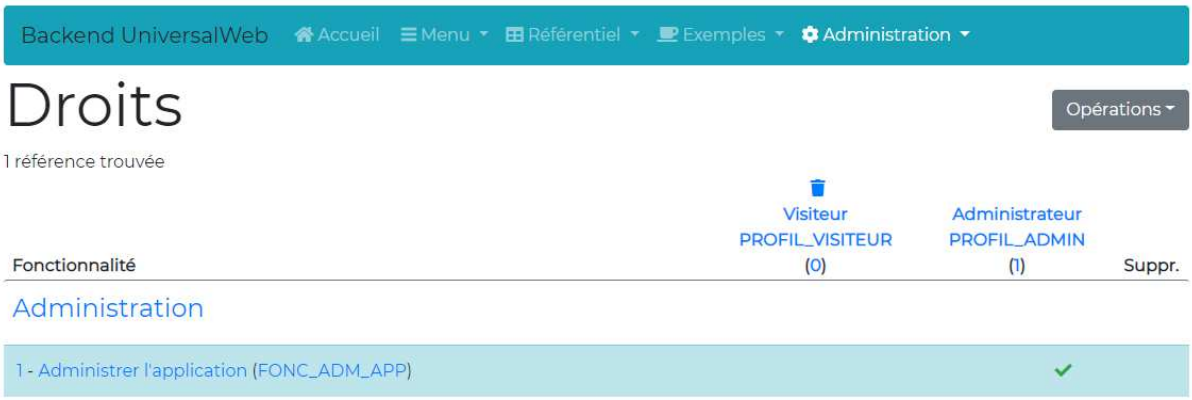
Note
Le buffer est vidé après chaque affichage. Cette fonction est mise en œuvre dans le script **brique_debug.inc.php** qui est proposé sur le frontend et le backend des squelettes UniversalWeb en affichage juste avant les footer respectifs.

Les informations sont seulement affichées aux IP définies dans la définition _IP_DEVELOPPEMENT_ qui doit être renseignée dans le fichier de configuration de l'application (**config.inc.php**). Si votre IP n'apparait pas dans ce tableau , les informations fournies par **DEBUG_PRINT_()** ne seront pas affichées.

5. Mécanisme d'autorisations d'accès

UniversalWeb fourni un mécanisme d'autorisation d'accès aux scripts de votre application basé sur un tableau croisé entre fonctionnalités de l'application et profils utilisateurs. Ces informations sont

contenus dans les tables de la base de données suivantes selon le MCD présenté dans le chapitre « Description de la base de données » (pour rappel **_profils**, **_fonctionnalites**, **_groupes_fonctionnalites**, **_users** et **_droits)**. Cette gestion des droits est illustrée par sa mise en œuvre minimale sur le backend dans le menu « Administration -> Gestion des droits ».



Ici deux profils utilisateur (**PROFIL_ADMIN** et **PROFIL_VISITEUR**) et une fonctionnalité (« Administrer l'application » **FONC_ADM_APP**) appartenant au groupe de fonctionnalités « Administration ». A la rencontre de chaque coordonnée « fonctionnalité / profil » se trouve le **droit** correspondant qui est analysé au début de chaque script. Ainsi, puisque chaque utilisateur possède un profil, il est possible de gérer son accès aux différents scripts (ou parties de scripts) de l'application. Il suffit à l'administrateur de cocher ou décocher le droit concernée pour autoriser ou interdire la fonctionnalités pour le profil choisi.

La modification des fonctionnalités et des profils de cette grille de base fournie avec le backend est **uniquement** réalisable en mode « développement[1] », et ce, sans écrire la moindre ligne de code. Dans ce mode là, ce tableau s'enrichi de nombreuses icones et de liens qui permettent d'ajouter, modifier, supprimer une fonctionnalité (seule la suppression de la fonctionnalité Administrateur est impossible), et d'ajouter, modifier ou supprimer un profil.

Chaque fonctionnalité peut être modifiée à loisir. On peut lire 3 informations distinctes (de gauche à droite) :

1. Identifiant de la fonctionnalité (entier). Les fonctionnalités sont affichées dans l'ordre croissant de cet identifiant.
2. Libellé de la fonctionnalité.
3. Mnémonique de la fonctionnalité. C'est un code qui sera utilisé par vos scripts pour faire référence à la fonctionnalité en question.

Un clic sur une de ces rubriques permet d'en modifier le contenu via une zone de saisie.

De même, chaque profil peut être modifié à loisir via 3 nouvelles informations (de haut en bas) :

1. Libellé du profil
2. Mnémonique du profil. Ce code sera utilisé par vos scripts pour faire référence au profil en question.
3. Identifiant du profil (entier). La encore, les profils sont affichées dans l'ordre croissant de cet identifiant.

[1] Rappel : la détermination du mode d'exécution du backend (comme du frontend) est réalisée en positionnant le paramètre **_RUN_MODE_** dans le fichier de configuration **config.inc.php**.

Gestion de l'accès au script en fonction des droits

La vérification des droits par les scripts (pages) est réalisé par la fonction **`grantAccess()`** comme le montre le bout de code issu de la page d'index du backend. Cette vérification doit être réalisée au début de chaque page nécessitant une vérification de droits. Et pour ce faire il suffit simplement d'ajouter le test après l'inclusion de la librairie **`common.inc.php`**.

```
<?php
//---------------------------------------------------------
// INDEX
//---------------------------------------------------------
// éèàç : pour sauvegarde du fichier en utf-8
//---------------------------------------------------------
require_once('libs/common.inc.php');

//gère l'accès au script
$operation = grantAcces() or die();

$titrePage = _APP_TITLE_;
$scriptSup = '';
$fJquery = '';
…
```

Cette fonction est définie dans le script **`droits.inc.php`**. Elle gère l'accès aux scripts selon les droits accordés. A chaque nouvelle page écrite, le développeur devra enrichir cette fonction qui déterminera in fine l'accès ou le refus d'accès au script appelant. En cas d'oubli, votre script affichera le texte « Droits a préciser… » sur sa zone de messages.

La fonction ne nécessite aucun paramètre d'entrée. En sortie, plusieurs cas de figure à prendre en compte :

- L'accès n'est pas autorisé : la fonction renvoi **`false`**.
- L'accès est autorisé : la fonction renvoie le contenu de la variable **`operation`** passée sur la ligne d'url (**`GET`**) si elle existe ; la chaine de caractères « all » sinon. Cette valeur est stockées dans la variable **`$operation`**.

La fonction est organisée autour d'un **`switch`** qui route les tests en fonction du script appelant. Plusieurs cas de figure :

Accès universel

Tout le monde peut accéder au script quelque soit son profil. Aucun test n'est réalisé . La fonction doit alors simplement renvoyer la valeur de la variable **`$operation`** (envoi de « all » si ce paramètre n'est pas fourni, ce qui est le cas pour la page d'index)**.** Par exemple, tout le monde à le droit d'accéder à la page d'index.

```
//-------------------------------------------------
case _URL_INDEX_ : {
       tout le monde est autorisé, pas de test réalisé.
       return $operation;
       break;
}
```

Accès à la totalité du script pour une fonctionnalité.

Pour autoriser l'accès à un script complet, il suffit, pour le script en question, de passer à la variable **`$droitATester`** le mnémonique de la fonctionnalité choisie. La fonction ira d'elle-même (en réalité en fin de traitement) vérifier que l'utilisateur en cours a le profil suffisant pour accéder à la fonctionnalité choisie pour le script visé. Dans exemple ci-dessous, l'accès au script **`_URL_MAINTENANCE_`** sera testé avec la présence de la fonctionnalité **`FONC_ADM_APP`** (mnémonique) dans le profil de l'utilisateur connecté (c'est-à-dire que l'utilisateur connecté doit avoir le profil « Administrateur » pour accéder au script **`maintenance.php`**).

```
//-----------------------------------------------
case _URL_MAINTENANCE_ : {
       test à réaliser sur la fonctionnalité FONC_ADM_APP
       $droitATester = 'FONC_ADM_APP';
       break;
}
```

Accès à une partie de script pour une fonctionnalité

L'accès à différentes parties d'un même script peut être réservé à des fonctionnalités différentes (un même script peut contenir plusieurs fonctionnalités, c'est par exemple le cas classique d'un formulaire[2] utilisateur qui permet par exemple de consulter, ajouter, modifier ou supprimer un utilisateur). Dans ce cas là, la partie du script à tester doit être mise à disposition dans la variable **`operation`** de l'url (**`GET`**).

Le code passe en revue toutes les fonctionnalités applicable au script jusqu'à trouver celle qui est requise pour accéder à la partie de code à laquelle l'utilisateur veut accéder. La fonction ira d'elle-même (en réalité en fin de traitement) vérifier que l'utilisateur en cours a le profil suffisant pour accéder à la fonctionnalité choisie pour la partie de script visée. Dans exemple ci-dessous, l'accès au script **`_URL_CLIENT_?operation=consulter`** sera testé avec la présence de la fonctionnalité **`FONC_CLIENT_CONSULTER`** dans le profil de l'utilisateur connecté.

```
//-----------------------------------------------
Pour le script _URL_CLIENT_
case _URL_CLIENT_ : {

       droit à tester : la fonctionnalité FONC_CLIENT_CONSULTER
       $droitATester = 'FONC_CLIENT_CONSULTER';

       opérations (ou parties de code) autorisées  pour la fonctionnalité ci-dessus
       $opAutorises  = array('consulter');

       vérification : si l'opération choisie (partie de code contenue dans $operation) fait bien partie des opérations
       autorisées, on quitte le switch (par la commande break) et le test est réalisé en fin de fonction ; sinon on poursuit
       avec la fonctionnalité suivante FONC_CLIENT_AJOUTER …
       if (in_array($operation, $opAutorises)) break;

       $droitATester = 'FONC_CLIENT_AJOUTER';
       $opAutorises  = array('ajouter');
       if (in_array($operation, $opAutorises)) break;

       $droitATester = 'FONC_CLIENT_MODIFIER';
       $opAutorises  = array('modifier');
       if (in_array($operation, $opAutorises)) break;

       $droitATester = 'FONC_CLIENT_SUPPRIMER';
       $opAutorises  = array('supprimer');
       if (in_array($operation, $opAutorises)) break;

       $droitATester = 'FONC_CLIENT_SUPPRIMER';
       $opAutorises  = array('supprimer');
       if (in_array($operation, $opAutorises)) break;

       A la fin si l'opération n'a été trouvée nulle par, le test d'accès est impossible et l'on part en erreur.
       $droitATester = 'erreur';
       break;
}
```

[2] Cette gestion des droits est d'ailleurs systématiquement mise en œuvre par **UniversalWeb** pour tous les formulaires de ce type créés avec la classe **`UniversalForm`**.

Classe Droits

Afin d'obtenir cette transparence d'utilisation, il est nécessaire que votre application soit initialisée avec cette grille de droits construite à partir des informations stockées dans la base de données. C'est le rôle de la classe **`Droits`**. Comme vous le remarquerez, dans le fichier **`init.inc.php`**, on crée un objet contenant tous les droits de l'application que l'on stocke dans la variable de session **`_APP_DROITS_`**. L'avantage du stockage en session est de ne pas relire les droits depuis la base de données à chaque lancement de script ; l'inconvénient de ce stockage (ad vitam de session) est qu'il faut pouvoir relire les droits à chaque modification. A vous de voir quelle est la meilleure option selon vos besoins.

```
//----------------------------------------------------------------------
// Creation de l'objet 'droits' qui gere les droits d'accès à l'application
// stockés dans la base de données
//----------------------------------------------------------------------
//$_SESSION[_APP_DROITS_] = null;
if  ((!isset($_SESSION[_APP_DROITS_])) || ($_SESSION[_APP_DROITS_] == null)) {
    $_SESSION[_APP_DROITS_] = new Droits(true);
}
```

Seule l'instanciation est nécessaire pour permettre la gestion des droits de toute votre application. Les autres méthodes publiques de la classe sont directement mises en œuvre pour la gestion de la grille « Gestion des droits » du backend. Néanmoins, voici leurs utilisations.

`__construct`

Constructeur de la classe Droits.

Description

```
_construct([boolean $notionGroupes=false]);
```

Liste des paramètres

`notionGroupes` : booléen permettant l'affichage de la grille par regroupement des fonctionnalités de l'application en groupes de fonctionnalités.

- **`true`** : affichage des fonctionnalités en groupes
- **`false`** : pas d'affichage par groupes

Valeurs de retour

Aucune

Exemple

```
$lesDroitsAppli = new Droits(true);
```

`countFonctionnalite`

Renvoie le nombre de fonctionnalités développées pour l'application.

Description

```
int countFonctionnalite();
```

Liste des paramètres

Aucun

Valeurs de retour

Nombre de fonctionnalités déclarées dans l'application.

Exemple
```
$lesDroitsAppli = new Droits(true);
$nbFonc = lesDroitsAppli->countFonctionnalite();
```

countProfils

Renvoie le nombre de profils développés pour l'application.

Description
```
int countProfils();
```

Liste des paramètres
Aucun

Valeurs de retour
Nombre de profils déclarés dans l'application.

Exemple
```
$lesDroitsAppli = new Droits(true);
$nbProfils = lesDroitsAppli->countProfils();
```

fonctionnalite ou getArrayFonctionnalite

Renvoie le tableau des fonctionnalités de l'application.

Description
```
array fonctionnalite();
```

Liste des paramètres
Aucun

Valeurs de retour
Tableau descriptif des fonctionnalités de l'application. Par défaut le backend affiche présente le tableau de fonctionnalités suivantes :

```
Array
(
    [FONC_ADM_APP] => Array
        (
            [id_fonctionnalite] => 1
            [id_groupe_fonctionnalite] => 2
            [groupe] => Administration
            [code] => FONC_ADM_APP
            [libelle] => Administrer l'application
        )

)
```

Exemple
```
$lesDroitsAppli = new Droits(true);
$lesFonctionnalites = lesDroitsAppli->fonctionnalite();
```

Note
Les méthodes **`fonctionnalite`** et **`getArrayFonctionnalite`** sont identiques.

profils

Renvoie le tableau des profils de l'application.

Description
```
array profils();
```

Liste des paramètres
Aucun

Valeurs de retour
Tableau descriptif des profils de l'application. Par défaut le backend affiche présente le tableau de profils suivants :

```
Array
(
    [PROFIL_VISITEUR] => Array
        (
            [id_profil] => 0
            [code] => PROFIL_VISITEUR
            [libelle] => Visiteur
        )

    [PROFIL_ADMIN] => Array
        (
            [id_profil] => 1
            [code] => PROFIL_ADMIN
            [libelle] => Administrateur
        )

)
```

Exemple
```
$lesDroitsAppli = new Droits(true);
$lesFonctionnalites = lesDroitsAppli->profils();
```

Note
On remarquera que le profil administrateur a pour identifiant 1 et le code mnémonique permettant d'y faire référence **PROFIL_ADMIN**.

getNotionGroupes

Renseigne si l'objet a pris en compte et mis en œuvre la notion de groupes de fonctionnalités.

Description
```
boolean getNotionGroupes();
```

Liste des paramètres
Aucun

Valeurs de retour
true : groupes de fonctionnalités pris en compte.
false : groupes de fonctionnalités ignorés.

retreiveCodeFonctionnaliteFromId

Retrouve le code mnémonique d'une fonctionnalité à partir de son identifiant.

Description
```
string retreiveCodeFonctionnaliteFromId(int $id);
```

Liste des paramètres
id : identifiant interne de la fonctionnalité.

Valeurs de retour
Code mnémonique représentant la fonctionnalité choisie.

Exemple

```
$lesDroitsAppli = new Droits(true);
echo lesDroitsAppli->retreiveCodeFonctionnaliteFromId(1);   //renvoi ”FONC_ADM_APP”
```

getFonctionnalite

Renvoie la tableau structuré de la fonctionnalité dont le code mnémonique est **$code**.

Description

```
array getFonctionnalite(int $code);
```

Liste des paramètres
code : Code mnémonique de la fonctionnalité ciblée.

Valeurs de retour
Tableau structuré de la fonctionnalité ciblée.

Exemple

```
$lesDroitsAppli = new Droits(true);
DEBUG_TAB_($lesDroitsAppli->getFonctionnalite('FONC_ADM_APP'));

Array
(
    [id_fonctionnalite] => 1
    [id_groupe_fonctionnalite] => 2
    [groupe] => Administration
    [code] => FONC_ADM_APP
    [libelle] => Administrer l'application
)
```

getIdFonctionnalite

Renvoie l’identifiant de la fonctionnalité dont le code mnémonique est **$code**.

Description

```
int getIdFonctionnalite(int $code);
```

Liste des paramètres
code : Code mnémonique de la fonctionnalité ciblée.

Valeurs de retour
Identifiant de la fonctionnalité ciblée.

Exemple

```
$lesDroitsAppli = new Droits(true);
echo $lesDroitsAppli->getIdFonctionnalite('FONC_ADM_APP');
```

Renvoie 1
Ceci revient au même que d’écrire

```
echo $lesDroitsAppli->getFonctionnalite('FONC_ADM_APP')['id_fonctionnalite'];
```

getLibelleFonctionnalite

Renvoie le libellé donné à la fonctionnalité dont le code mnémonique est **$code**.

Description

```
string getLibelleFonctionnalite(int $code);
```

Liste des paramètres

code : Code mnémonique de la fonctionnalité ciblée.

Valeurs de retour

Libellé de la fonctionnalité ciblée.

Exemple

```
$lesDroitsAppli = new Droits(true);
echo $lesDroitsAppli->getLibelleFonctionnalite('FONC_ADM_APP');
```

Renvoie 1
Ceci revient au même que d'écrire

```
echo $lesDroitsAppli->getFonctionnalite('FONC_ADM_APP')['libelle'];
```

getProfil

Renvoie la tableau structuré du profil dont le code mnémonique est **$code**.

Description

```
array getProfil(int $code);
```

Liste des paramètres

code : Code mnémonique du profil ciblé.

Valeurs de retour

Tableau structuré du profil ciblé.

Exemple

```
$lesDroitsAppli = new Droits(true);
DEBUG_TAB_($lesDroitsAppli->getProfil('PROFIL_ADMIN'));

Array
(
    [id_profil] => 1
    [code] => PROFIL_ADMIN
    [libelle] => Administrateur
)
```

getIdProfil

Renvoie l'identifiant du profil dont le code mnémonique est **$code**.

Description

```
int getIdProfil(int $code);
```

Liste des paramètres
code : Code mnémonique du profil ciblé.

Valeurs de retour
Identifiant du profil ciblé.

Exemple

```
$lesDroitsAppli = new Droits(true);
echo $lesDroitsAppli->getIdProfil('PROFIL_ADMIN');
```

Renvoie 1
Ceci revient au même que d'écrire

```
echo $lesDroitsAppli->getProfil('PROFIL_ADMIN')['id_profil'];
```

getLibelleProfil

Renvoie le libellé donné au profil dont le code mnémonique est **$code**.

Description

```
string getLibelleProfil(int $code);
```

Liste des paramètres
libelle : Code mnémonique du profil ciblé.

Valeurs de retour
Libellé du profil ciblé.

Exemple

```
$lesDroitsAppli = new Droits(true);
echo $lesDroitsAppli->getLibelleProfil('PROFIL_ADMIN');
```

Renvoie 1
Ceci revient au même que d'écrire

```
echo $lesDroitsAppli->getProfil('PROFIL_ADMIN')['libelle'];
```

droitExiste

Renseigne si un droit a été déclaré pour un couple fonctionnalité / profil.

Description

```
boolean droitExiste(string $codeFonctionnalite, int $idProfil);
```

Liste des paramètres
codeFonctionnalite : code mnémonique de la fonctionnalité ciblée.
idProfil : identifiant du profil ciblé.

Valeurs de retour
true : le droit du couple **idProfil** / **codeFonctionnalite** existe.
false : le droit du couple **idProfil** / **codeFonctionnalite** n'existe pas ou bien la fonctionnalité n'existe pas.

Exemple
ici on teste l'existence du droit PROFIL_VISITEUR / « Administrer l'application ».

```
$lesDroits = new Droits(true);
if ($res = lesDroits->droitExiste('FONC_ADM_APP', lesDroits->getIdProfil('PROFIL_ADMIN'))) {
  echo 'Le droit '. lesDroits->getLibelleFonctionnalite('FONC_ADM_APP').' a été créé pour le profil '.
    lesDroits->getLibelleProfil('PROFIL_ADMIN');
}
else echo 'Droit non créé';
```

Note
Attention, la méthode de dit pas si l'utilisateur en cours a les droits d'accès à cette fonctionnalité. Pour obtenir cette réponse utiliser la méthode **accesAutorise**.

accesAutorise

Dit si le couple fonctionnalité / profil a droit d'accès.

Description
```
boolean droitAutorise(string $codeFonctionnalite, int $idProfil);
```

Liste des paramètres
codeFonctionnalite : code mnémonique de la fonctionnalité ciblée.
idProfil : identifiant du profil ciblé.

Valeurs de retour
true : le profil **idProfil** a accès à la fonctionnalité **codeFonctionnalite**.
false : le profil **idProfil** n'a pas accès à la fonctionnalité **codeFonctionnalite** ou bien la fonctionnalité n'existe pas.

Exemple
ici on tente de vérifier si le **PROFIL_VISITEUR** a accès à la fonctionnalité « Administrer l'application »

```
$lesDroits = new Droits(true);
if ($lesDroits->accesAutorise('FONC_ADM_APP', $lesDroits->getIdProfil('PROFIL_ADMIN'))) {
  echo 'Le profil '.$lesDroits->getLibelleProfil('PROFIL_ADMIN').' peut accéder à la fonctionnalité
'.$lesDroits->getLibelleFonctionnalite('FONC_ADM_APP');
}
else echo 'Accès non autorisé';
```

Note
Par souci de simplification, le backend de **UniversalWeb** implémente une fonction qui permet d'obtenir la même réponse à l'interrogation sans faire appel directement à l'objet en cours et pour l'utilisateur en cours.

```
function accesAutorise($codeFonctionnalite) {
   if (!isset($codeFonctionnalite)) return false; // retourne false si fonctionnalité non créée
   //test de la fonctionnalité
   return $_SESSION[_APP_DROITS_]->accesAutorise($codeFonctionnalite, $_SESSION[_APP_LOGIN_]-
>getProfil());
}
```

Pour savoir si l'utilisateur en cours a accès à une fonctionnalité, l'appel est alors des plus simple :

```
if (accesAutorise('FONC_ADM_APP')) {
}
```

accesAutoriseByIdFonc

Dit si le couple **$idFonctionnalite** / **$idProfil** est autorisé.

Description

```
boolean accesAutoriseByIdFonc(int $idFonctionnalite, int $idProfil);
```

Liste des paramètres

idFonctionnalite : identifiant de la fonctionnalité ciblée.
idProfil : identifiant du profil ciblé.

Valeurs de retour

true : le profil **idProfil** a accès à la fonctionnalité **idFonctionnalite**.
false : le profil **idProfil** n'a pas accès à la fonctionnalité **idFonctionnalite**.

Exemple

```
$lesDroits = new Droits(true);
if (lesDroits->accesAutoriseByIdFonc(1, 1)) {
   echo 'Accès autorisé';
}
else echo 'Accès non autorisé';
```