Il progetto è realizzato con il contributo finanziario del Programma LIFE della Commissione Europea

Acronimo del progetto *Project Acronym*	**W.I.Z.**
Titolo completo del progetto *Project Full Title*	**WIZ – WaterIZe spatial planning: encompass future drinkwater management conditions to adapt to climate change**
Numero del progetto *Project No*	**LIFE09 ENV/IT/000056**

No. Deliverable D11.4

Componenti WIZ disponibili per ulteriori sviluppi/integrazioni

WIZ components packaged in a way suitable for further development/integration

Mese/*Month* – Anno/*Year* Giugno/June 2012

Partner di progetto/Project Partner

Capofila/Main Contractor

Acque S.p.A.
Sede Legale: Via Garigliano, 1
I - 50053 EMPOLI –IT
Sede operativa: Via A. Bellatalla, 1
I – 56121 Ospedaletto (PI)
http://www.acque.net

Autorità di bacino
Via dei Servi, 15
I - 50122 FIRENZE – IT
http://www.adiba.it

Ingegnerie Toscane S.r.l.
Via di Villamagna, 90
I – 50126 Firenze

Via A. Bellatalla, 1
I – 56121 Ospedaletto (PI)
http://www.acque.net

Instituto Tecnologico de Galicia
PO.CO.MA.CO Sector I Portal 5
ES – 15190 A Coruña - Galicia – ESPAÑA
http://www.itg.es

Informazioni sul documento/Document Information

Project/Progetto

Acronimo del progetto/ Project Acronym	*W.I.Z.*
Titolo completo del progetto/Project Full Title	*WIZ – WaterIZe spatial planning: encompass future drinkwater management conditions to adapt to climate change*
Data di avvio/Project start:	*09/01/10*
Durata del Progetto/Project duration:	*36 mesi*
Contratto no/Grant agreement no.:	*LIFE09 ENV/IT/000056*

Document

No Deliverable/Deliverable No:	*D11.4*
Titolo del Deliverable/Deliverable title:	*Componenti WIZ disponibili per ulteriori sviluppi/integrazioni*
Data contrattuale del Deliverable/Contractual Date of Delivery:	*29/06/12*
Data di consegna del Deliverable/Actual Date of Delivery:	*29/06/12*
Editore(i)/Editor(s):	
Autore(i)/Author(s):	
Revisore(i)/Reviewer(s):	
Partner/Partner(s):	*INGTOS*
No Work package/Work package no.:	*AZIONE #011*
Titolo Work package /Work package title:	*Implementazione del motore di proiezione WIZ*
Leader del Work package/ Work package leader:	*INGTOS*
Distribuzione/Distribution (Public/Reserved):	*Public*
Natura/Nature (Report, …):	*Manual*
Versione-Revisione/ Version-Revision:	*0b*
Bozza-Definitivo /Draft-Final	*Final*
No di pagine (inclusa copertina)/ Total number of pages: (including cover)	*25*
Parole chiave/ Keywords:	*W.I.Z., Deliverable*

Revisioni/Change Log

Motivo della revisione/Reason for change	Argomento della revisione/Issue	Numero della Revisione/Revision	Data della Revisione /Date
–	Versione Iniziale/initial Version	0a	06/06/12
Revisione e Aggiornamento/ Updating	Revisione e Aggiornamento/Updating	0b	25/01/13

Esonero Reponsabilità/Disclaimer

Questo documento contiene descrizioni che riguardano le attività, i risultati e i prodotti del Progetto WIZ. Alcune sue parti potrebbero essere tutelate sotto Diritto di Proprietà Intellettuale (IPR).
Per questo motivo vi chiediamo di contattare il Consorzio WIZ prima di utilizzarlo (e.mail: o.cei@ingegnerietoscane.net).

Se ritenete che questo documento sia in qualsiasi modo lesivo dei diritti di proprietà intellettuale di vostro possesso – come persona o come rappresentante di un organizzazione – informateci tempestivamente.
Gli autori di questo documento hanno preso tutte le misure disponibili possibili per far sì che il suo contenuto sia accurato, consistente e legale. Tuttavia, né il partenariato nel suo insieme, né i singoli partner che direttamente o indirettamente abbiano preso parte alla creazione e alla pubblicazione di questo documento sono responsabili per qualsiasi cosa possa accadere come risultato del suo utilizzo.

Questa pubblicazione è stata realizzata grazie al contributo dell'Unione Europea. Il consorzio WIZ è il solo responsabile del contenuto di questa pubblicazione che non riflette necessariamente il pensiero dell'Unione Europea

WIZ è parzialmente finanziato dll'Unione Europea (Life+ Programme).

This document contains description of the WIZ project findings, work and
products. Certain parts of it might be under partner Intellectual Property Right (IPR) rules so, prior to using its content please contact the consortium head for (e.mail: o.cei@acqueingegneria.net).

In case you believe that this document harms in any way IPR held by you as a person or as a representative of an entity, please do notify us immediately.
The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated the creation and publication of this document hold any sort of responsibility that might occur as a result of using its content.

This publication has been produced with the assistance of the European Union. The content of this publication is the sole responsibility of WIZ

Consortium and can in no way be taken to reflect the views of the European Union.

WIZ is a project partially funded by the European Union

Indice

1 Executive summary

Deliverable **D11.4 – WIZ components packaged in a way suitable for further development/integration** lists the features that enable a developer to extend the potentials of the "WIZ system" by using plugins and code extensions, making the most of the potentials of the Yii framework. Note that the source code is available as Open Source at http://wiz.acque.net/index.php?r=site/page&view=source.

The purpose is to make the system easily developable by any community that may be interested.

So, extra features (*Plugin*) may be added and the existing ones may be changed or generally customised (*extensions*).

Plugin

A plugin is used to add extra features to a system, simply and quickly; a rough knowledge of the framework is enough but a good knowledge of the source code of the portal is essential. A plugin consists of an archive (.tar o .zip), with the name acting as a univocal ID. Through a simple graphical interface, the developer may upload the archives that compose a plugin and enable or disable the preloaded ones. The structure of the archive must necessarily be the same as the one put in as a *Foo plugin* (as detailed in the document), otherwise the system will not be able to properly install it.

The structure of the files composing the archive are then described; they are:

- Version
- Sql
- Model
- Controller
- Views

Extensions

Yii has been designed in such a way that nearly every piece of its code may be extended and customised to adapt it to personal requirements. In this case, an excellent knowledge of both the framework and the source code is essential.

Potential extensions may be:

(1) Application Component

(2) Behaviour

(3) Widget

(4) Filter

(5) Controller

(6) Validator

(7) Helper

(8) Module

A **test website for the developers' community** has been specially developed as a **testing ground to upload and test new plugins**, thus improving and adding extra features to the system.

The website is available at http://dev-wiz.acque.net/ and one must be registered as a *Developer* to be able to contribute to developing the system.

2 Sommario esecutivo

Il **D11.4 - WIZ components packaged in a way suitable for further development/integration (Componenti WIZ disponibili per ulteriori sviluppi/integrazioni)** indica le funzionalità che permettono ad un utente sviluppatore – quindi con conoscenze informatiche avanzate – di poter ampliare le potenzialità del portale stesso tramite l'utilizzo di plugins e le estensioni del codice, utilizzando le potenzialità offerte dal framework Yii .

La stesura del documento è frutto di una azione di confronto tra Ingegnerie Toscane Srl, beneficiario responsabile dell'Azione #11 (Implementazione del motore di proiezione WIZ), e il fornitore esterno incaricato dello sviluppo del sistema, nonché degli altri partner di progetto.

Ricordiamo che il codice sorgente è disponibile open-source all'indirizzo http://wiz.acque.net/index.php?r=site/page&view=source .

3 Scopo

Lo scopo è che il motore WIZ (da contratto con la CE) sia reso disponibile in modo che la comunità interessata possa proseguirne lo sviluppo e miglioramento.

4 Plugins e estensioni

Aggiungere nuove funzionalità è un'operazione molto comune durante il ciclo di vita di un software. Per il portale WIZ sono state previste delle funzionalità per permettere ad un utente sviluppatore, quindi con conoscenze informatiche avanzate, di poter ampliare le potenzialità del portale stesso tramite l'utilizzo di plugins.

Oltre che aggiungere funzionalità è anche possibile modificare o, in generale, personalizzare quelle esistenti; questa operazione non può essere effettuata utilizzando il meccanismo dei plugin ma bisogna 'estendere' il codice, utilizzando le potenzialità offerte dal framework Yii (http://www.yiiframework.com/).

Nel primo caso è richiesta una conoscenza marginale del framework ma una buona conoscenza del codice sorgente del portale; nel secondo, invece, è necessario avere ottime conoscenze sia del framework che del sorgente.

4.1 Plugins

Un plugin permette di aggiungere nuove funzionalità al sistema in maniera semplice e veloce. È costituito da un archivio (.tar o .zip), con il nome che funge da ID univoco. Di seguito è mostrata la struttura per il plugin Foo:

Foo.tar plugin archive file

Foo.version text file containing information about the plugin version

Foo.sql sql file to create new tables (opzionale)

Foo.php model class file

FooController.php controller class file

views/ containing view files

Attraverso una semplice interfaccia grafica l'utente sviluppatore può caricare gli archivi che costituiscono il plugin ed abilitare o disabilitare quelli preventivamente caricati.

La struttura dell'archivio deve necessariamente essere quella illustrata sopra altrimenti il sistema non sarà in grado di installare il plugin correttamente.

Nei paragrafi successivi verranno illustrate le strutture che dovranno avere i vari file che compongono l'archivio.

4.2 Version

Il file version è un file di testo con estensione txt. È suddiviso in sezioni, racchiuse da parentesi quadre. La sezione principale è version, obbligatoria, e contiene il numero di versione del plugin; le altre due sezioni sono opzionali:

- Description: contiene una descrizione del plugin e delle funzionalità offerte
- Changelog: contiene informazioni sulle modifiche apportate nell'ultima versione (ed eventualmente nelle precedenti)

Di seguito è mostrato un esempio di un ipotetico file version per il plugin Foo.

[version]

1.30-r9 Beta

[description]

A short or detailed description of Foo plugin

[changelog]

The changelog

4.2.1 Sql

Nel caso in cui il plugin necessiti di nuove tabelle nel database è necessario fornire un file contenente l'sql per creare tali tabelle. Il portale non entra nel merito del contenuto del file; esegue semplicemente il codice sql per cui è responsabilità dello sviluppatore fornire file sql validi e formalmente corretti, considerando anche il DBMS utilizzato. Di seguito è mostrato un esempio di file sql.

```
CREATE TABLE foo_table_one
(
id integer NOT NULL,
something character varying(255) NOT NULL,
CONSTRAINT id_pkey PRIMARY KEY(id)
)
```

4.2.2 Model

Il modello è una classe che deve estendere CFormModel o CActiveRecord. Viene utilizzato per manipolare e memorizzare i dati. Rappresenta un singolo oggetto che può essere una riga in una tabella o

una form html con dei campi di input. Ogni campo dell'oggetto è rappresentato da un attributo del modello.

Il nome della classe deve essere uguale a quello del file del modello (esclusa l'estensione .php).

```
class Foo extends CActiveRecord
{
    public static function model($className=__CLASS__)
    {
    return parent::model($className);
    }
    public function tableName()
    {
    return 'foo_table_one';
    }
}
```

4.2.3 Controller

Il controller deve estendere la classe CExtController o qualsiasi altro controller già presente nel portale. Il controller esegue le varie action richieste, utilizzando eventualmente i dati del modello e visualizzando le informazioni tramite le view.

```
class FooController extends CExtController
{
    public function actionIndex()
    {
    // ...
    }
    // other actions
}
```

4.2.4 Views

Tutte le view devono essere organizzate dentro la cartella views. Una view altro non è che un file html, con dentro eventualmente del codice in php.

```
<div class="top">
```

```
            <h1>Title</h1>
        <?php echo $content; ?>
    </div>
```

Le view vengono utilizzate, all'interno del controller, dalle varie action per visualizzare dati e informazioni all'utente. L'esempio seguente mostra come invocare la view edit da una qualsiasi action di FooController.

```
$this->render('edit', array(
        'var1'=>$value1,
        'var2'=>$value2,
));
```

Di seguito, invece, viene mostrato come invocare una view già esistente e relativa al componente bar.

```
$this->render('//bar/view', array(
        'var1'=>$value1,
        'var2'=>$value2,
));
```

4.3 Estensioni

Una estensione tipicamente serve per uno scopo ben preciso. Le estensioni possono essere classificate in:

- **application component**
- **behavior**
- **widget**
- **action**
- **filter**
- **controller**
- **validator**
- **helper**
- **module**

L'estensione può essere anche un componente che non rientra in nessuna delle categorie di cui sopra. Yii è studiato in modo tale che quasi ogni pezzo del suo codice può essere esteso e personalizzato per adattarsi alle esigenze individuali.

4.3.1 Application Component

Un application component deve implementare l'interfaccia IApplicationComponent o estendere da CApplicationComponent. Il metodo principale che deve essere implementato è IApplicationComponent::init, grazie al quale il componente esegue le operazioni di inizializzazione. Questo metodo viene invocato subito dopo che il componente è stato creato e setta i valori iniziali delle proprietà (specificate nella configurazione dell'applicazione).

Per default, un application component viene creato ed inizializzato solo quando viene acceduto per la prima volta durante la gestione di una richiesta. Tuttavia, se l'application component deve essere creato subito dopo la creazione di una istanza dell'applicazione, è compito dell'utente inserire l'ID del componente nella proprietà CApplication::preload.

Per utilizzare un application component bisogna innanzitutto modificare il file di configurazione dell'applicazione aggiungendo una nuova property,come mostra di seguito:

```
return array(
    // 'preload'=>array('xyz',...),
    'components'=>array(
        'xyz'=>array(
            'class'=>'ext.xyz.XyzClass',
            'property1'=>'value1',
            'property2'=>'value2',
        ),
        // other component configurations
    ),
);
```

Successivamente si può accedere al componente in qualsiasi punto del codice utilizzando Yii::app()->xyz. Il componente sarà creato non appena verrà acceduto per la prima volta a meno che non sia stato inserito nella proprietà preload.

4.3.2 Behavior

Per creare un behavior bisogna implementare l'interfaccia Ibehavior. Per comodità, Yii fornisce la classe CBehavior che implementa già questa interfaccia e fornisce comodi metodi aggiuntivi. Le classi figlie dovranno principalmente implementare i metodi extra che intendono mettere a disposizione dei componenti ai quali sono collegati.

Quando si sviluppa un behavior per CModel e CActiveRecord, si può anche estendere da CModelBehavior e CactiveRecordBehavior,

rispettivamente. Queste classi offrono funzionalità addizionali specifiche per CModel e CActiveRecord.

Il codice seguente mostra un esempio di un ActiveRecord behavior. Quando questo behavior è collegato ad un oggetto AR, non appena l'oggetto viene salvato chiamando il metodo save(), esso setta automaticamente il valore dell'attributo create_time e update_time con il timestamp corrente.

```
class TimestampBehavior extends CActiveRecordBehavior
{
    public function beforeSave($event)
    {
        if($this->owner->isNewRecord)
            $this->owner->create_time=time();
        else
            $this->owner->update_time=time();
    }
}
```

Un behavior può essere utilizzato in qualsiasi componente. L'utilizzo richiede due step: collegare il behavior al componente ed invocare il behavior. Per esempio:

```
// $name uniquely identifies the behavior in the component
$component->attachBehavior($name,$behavior);
// test() is a method of $behavior
$component->test();
```

Spesso, un behavior viene collegato ad un componente utilizzando il file di configurazione invece che invocare il metodo attachBehavior. Per esempio:

```
return array(
    'components'=>array(
        'db'=>array(
            'class'=>'CDbConnection',
            'behaviors'=>array(
                'xyz'=>array(
                    'class'=>'ext.xyz.XyzBehavior',
                    'property1'=>'value1',
```

```
                                'property2'=>'value2',
                            ),
                        ),
                    ),
                    //....
                ),
        );
```

Questo codice collega il behavior xyz all'application component db. Per le classi CController, CFormModel e CActiveRecord che generalmente vengono estese, i behavior possono essere collegati effettuando l'override del metodo behaviors() . Per esempio:

```
public function behaviors()
{
    return array(
    'xyz'=>array(
        'class'=>'ext.xyz.XyzBehavior',
        'property1'=>'value1',
        'property2'=>'value2',
    ),
);
}
```

4.3.3 Widget

Un widget estende CWidget o una sua classe figlia.

Il metodo più semplice per creare un nuovo widget è quello di estendere un widget esistente ed effettuare l'override dei suoi metodi o modificare i valori di default delle proprietà. Per esempio, se si vuole utilizzare uno stile CSS più carino per CTabView si può configurare la proprietàCTabView::cssfile quando si usa il widget oppure si può estendere CTabView come mostrato sotto in modo da non dover più configurare questa proprietà ad ogni utilizzo.

```
class MyTabView extends CTabView
{
    public function init()
    {
```

```
            if($this->cssFile===null)
            {
                $file=dirname(__FILE__).DIRECTORY_SEPARATOR.'tabview.css';
                $this->cssFile=Yii::app()->getAssetManager()->publish($file);
            }
            parent::init();
        }
    }
```

Nel codice mostrato sopra viene effettuato l'override del metodo CWidget::init ed assegnato a CTabView::cssFile l'URL del nuovo stile CSS, se la proprietà non è già stata settata. Il nuovo stile CSS viene messo nella stessa directory che contiene la classe MyTabView, in modo che il tutto possa anche essere rilasciato come estensione. Poiché il file CSS non è accessibile da Web, è necessario pubblicarlo.

Per creare un nuovo widget da zero, è necessario implementare due metodi: CWidget::init e CWidget::run. Il primo metodo viene invocato quando si utilizza $this->beginWidget per inserire il widget in una vista, mentre il secondo è invocato quando si utilizza $this->endWidget. Se si vuole catturare e processare il contenuto mostrato tra l'invocazione di questi due metodi bisogna bufferizzare l'output in CWidget::init per poi recuperarlo in CWidget::run per eventuali processamenti.

Un widget richiede spesso CSS, Javascritp e altri file nella pagina che utilizza il widget stesso. Questi file vengono generalmente chiamati assets perché stanno insieme al file che contiene il widget e non sono accessibili dagli utenti web. Per rendere questi file accessibili da web è necessario pubblicarli utilizzando CWebApplication::assetManager, come mostrato nel frammento di codice sottostante. Inoltre, se bisogna includere un file CSS o un Javascript nella pagina è necessario registrarlo utilizzando CclientScript.

```
    class MyWidget extends CWidget
    {
        protected function registerClientScript()
        {
            // ...publish CSS or JavaScript file here...
            $cs=Yii::app()->clientScript;
            $cs->registerCssFile($cssFile);
            $cs->registerScriptFile($jsFile);
        }
```

```
}
```

Un widget può anche i propri file per le viste. In questo caso bisogna creare una directory chiama views sotto la directory contenente la classe del widget e posizionare tutte le viste dentro questa directory. In una classe widget, per mostrare una vista, si utilizza $this->render('ViewName').

Data una widget XyzClass appartenente all'estensione xyz, possiamo richiamarla in una view come mostra il codice seguente:

```
// widget that does not need body content
<?php $this->widget('ext.xyz.XyzClass', array(
        'property1'=>'value1',
        'property2'=>'value2')); ?>
// widget that can contain body content
<?php $this->beginWidget('ext.xyz.XyzClass', array(
        'property1'=>'value1',
        'property2'=>'value2')); ?>
...body content of the widget...
<?php $this->endWidget(); ?>
```

4.3.4 Action

Una action estende la classe CAction o una sua classe figlia. Il metodo principale che deve essere implementato è IAction::run.

Le action vengono utilizzate in un controller per rispondere a specifiche richieste. Data una action XyzClass appartenente all'estensione xyz, l'invocazione può essere effettuata facendo l'override del metodo CController::action della classe controller:

```
class TestController extends CController
{
        public function actions()
        {
                return array(
                        'xyz'=>array(
                                'class'=>'ext.xyz.XyzClass',
                                'property1'=>'value1',
                                'property2'=>'value2',
                        ),
```

```
                // other actions
            );
        }
    }
```

La action sarà accessibile tramite il percorso test/xyz.

4.3.5 Filter

Un filter estende la classe CFilter o una sua classe figlia. I metodi principali che devono essere implementati sono CFilter::preFilter e CFilter::postFilter. Il primo viene invocato prima che la action venga eseguita mentre il secondo dopo.

```
class MyFilter extends CFilter
{
    protected function preFilter($filterChain)
    {
        // logic being applied before the action is executed
        return true; // false if the action should not be executed
    }
    protected function postFilter($filterChain)
    {
        // logic being applied after the action is executed
    }
}
```

Il parametro $filterChain è di tipo CFilterChain e contiene informazioni sulla action corrente.

Data un filter XyzClass appartenente all'estensione xyz, esso può essere utilizzato facendo l'override del metodo CController::filters della classe controller:

```
class TestController extends CController
{
    public function filters()
    {
        return array(
            array(
                'ext.xyz.XyzClass',
                'property1'=>'value1',
```

```
                    'property2'=>'value2',
                ),
                // other filters
            );
        }
    }
```

4.3.6 Controller

Un controller distribuito come estensione dovrebbe estendere CExtController, invece che CController. Il motivo principale è che CController assume che i file delle viste risiedano sotto application.views.ControllerID mentre CExtController assume che i file si trovino nella directory views che è una sottodirectory di quella che contiene la classe controller.

Un controller fornisce un insieme di actions che possono essere richiestedall'utente. Al fine di poter utilizzare una estensione controller è necessario configurare la proprietà CWebApplication::controllerMap nel file di configurazione dell'applicazione:

```
return array(
    'controllerMap'=>array(
        'xyz'=>array(
            'class'=>'ext.xyz.XyzClass',
            'property1'=>'value1',
            'property2'=>'value2',
        ),
        // other controllers
    ),
);
```

La action a del controller può essere acceduta tramite il percorso xyz/a.

4.3.7 Validator

Un validator deve estendere da CValidator ed implementare il metodo Cvalidator::validateAttribute

```
class MyValidator extends CValidator
{
    protected function validateAttribute($model,$attribute)
    {
```

```
            $value=$model->$attribute;
            if($value has error)
                $model->addError($attribute,$errorMessage);
        }
    }
```

Un validator è principalmente utilizzato in una classe model (o una qualsiasi che estende CFormModel o CActiveRecord). Data un validator XyzClass appartenente all'estensione xyz, esso può essere utilizzato effettuando l'override del metodo CModel::run della classe model:

```
    class MyModel extends CActiveRecord // or CFormModel
    {
        public function rules()
        {
            return array(
                array(
                    'attr1, attr2',
                    'ext.xyz.XyzClass',
                    'property1'=>'value1',
                    'property2'=>'value2',
                ),
                // other validation rules
            );
        }
    }
```

4.3.8 Helper

È una classe formata solo da metodi statici. Il suo comportamento è simile a quello delle funzioni globali; per invocarle si utilizza il nome della classe come namespace.

```
    class MyHelper
    {
        public static function utility($param)
        {
            // ...code here...
        }
```

```
}

// ......
        MyHelper::utility($value)
// ......
```

4.3.9 Module

Un module estende la classe CWebModule; consiste di models, views and controllers.

Una linea guida generale per lo sviluppo di un module prevede che esso sia self-contained. I file (come i CSS, i Javascript e le immagini) che sono utilizzati da un module devono essere distribuiti insieme al module stesso. Il module deve anche pubblicare questi file in modo da renderli accessibili da web.

Un module è organizzato in una directory, con il nome che funge da ID univoco. Di seguito è mostrata una struttura di directory tipica del module forum:

```
forum/
        ForumModule.php the module class file
        components/ containing reusable user components
          views/ containing view files for widgets
        controllers/ containing controller class files
          DefaultController.php the default controller class file
        extensions/ containing third-party extensions
        models/ containing model class files
        views/ containing controller view and layout files
          layouts/ containing layout view files
          default/ containing view files for DefaultController
            index.php the index view file
```

Per utilizzare un module, bisogna innanzitutto copiare il module nella directory modules dell'applicazione. Successivamente bisogna dichiarare aggiungere l'ID del module nella proprietà modules dell'applicazione. Per esempio, per poter utilizzare il module forum, bisogna utilizzare la seguente configurazione:

```
return array(
```

```
        ......
        'modules'=>array('forum',...),
        ......
);
```

Un module può anche avere delle proprietà che devono essere inizializzate. Per esempio, il module forum può avere la proprietà postPerPage che può essere configurata nel seguente modo:

```
return array(
        ......
        'modules'=>array(
                'forum'=>array(
                        'postPerPage'=>20,
                ),
        ),
        ......
);
```

L'istanza di un module può essere acceduta tramite la proprietà module del controller attivo. Attraverso l'istanza è possibile accedere alle informazioni che sono condivise a livello di module. Per esempio, per accedere alla proprietà postPerPage si può utilizzare:

```
$postPerPage=Yii::app()->controller->module->postPerPage;
// or the following if $this refers to the controller instance
// $postPerPage=$this->module->postPerPage;
```

La action di un controller che si trova dentro un module può essere acceduta tramite il percorso moduleID/controllerID/actionID. Per esempio, ipotizzando che il module forum abbia un controller di nome PostController, è possibile accedere alla action create del controller tramite il percorso forum/post/create. L'url corrispondente sarà http://www.example.com/index.php?r=forum/post/create .

5 Conclusioni

Come già accennato, lo scopo di questo documento è quello di rendere agevole e possibile lo sviluppo e miglioramento del sistema da parte della comunità interessata, attraverso l'aggiunta di funzionalità (Plugin), sia modificare o, in generale, personalizzazione di quelle esistenti (estensioni).

La documentazione predisposta, volutamente sintetica e molto operativa, vuole essere da supporto alla comunità scientifica e ad eventuali altri utilizzatori in questa direzione, in modo da permettere il "ri-uso" e potenziamento del WIZ-Engine.

Un'**area di sperimentazione per la comunità degli sviluppatori** è disponibile all'indirizzo http://dev-wiz.acque.net/ . Dal sito di prova realizzato è possibile registrarsi come Utente Sviluppatore per caricare e testare dei plug-in precedentemente scritti, così migliorare il sistema e aggiungere ulteriori funzionalità.