Wizard Behavior

Multi-form Navigation

for the Yii PHP Framework

Developed for the Yii Community by

PBM Web Development

PBM

Wizard Behavior

Contents

Figures

Tables

Wizard Behavior

Introduction

Handling multi-stage forms, even a second form, introduces a number of issues; data persistence, user navigation, etc. The problem becomes worse if advanced features such as Plot-Branching Navigation and the ability to save and resume data entry are required.

Wizard Behavior simplifies multi-form handling allowing you deal with handling the data and provides advanced features.

Features

- Collects data from multi-step forms, greatly simplifying their handling
- Data saved in the Session
- Plot-branching navigation; choices on which forms to display can be made at runtime
- Next/Previous or Forward Only navigation, the latter is useful for tests, surveys, etc.
- Save and reload a partially completed wizard; allows users to resume a partially completed wizard at a later time without having to re-enter data
- Ability to timeout steps
- Utility methods to get current step, step count, and a menu of steps; use in the view to assist user navigation
- All steps submit to the same Controller::Action for friendly URLs
- Event driven
- Simple installation and configuration

Credits

Wizard Behavior was inspired by the CakePHP Wizard Component: http://github.com/jaredhoyt

Resources

Try It Out

You can try out Wizard Behavior at http://wizard-behavior.pbm-webdev.co.uk

This has 3 wizards that demonstrate the features of Wizard Behavior.

Downloads

You can download Wizard Behavior and the demo from

License

Wizard Behavior is free software. It is released under the terms of the following BSD License.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of PBM Web Development nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Compatibility

	Yii	OS
Tested with	1.1.6	Windows 7
Should work with	All	All

Installation

1. Download User Manager from http://www.yiiframework.com/extension/wizard/
2. Extract the module and place in the required folder; typically the application.extensions folder.

Wizard Behavior

WizardBehavior

Inheritance	WizardBehavior » CBehavior » CComponent
Implements	IBehavior

Public Properties

See CBehavior for inherited properties.

Name	Type	Description
autoAdvance	boolean	Whether to advance to the expected or next step.
cancelButton	string	The name attribute of the button used to cancel the wizard.
cancelledUrl	mixed	Url to be redirected to if the user clicks cancelButton.
currentStep	integer	The 1 based index of the step being processed.
draftSavedUrl	mixed	Url to be redirected to if the user clicks saveDraftButton.
defaultBranch	boolean	Whether to use the default in a branch group if no branch is selected.
events	array	Array of eventName=>eventHandlerName pairs.
finishedUrl	mixed	Url to be redirected to after the wizard has finished.
expiredUrl	mixed	Url to be redirected to if a step expires.
forwardOnly	boolean	Whether navigation is forward only or the user can return to previous steps.
menu	object	The menu object with steps in the object's *items* property.
menuLastItem	string	If not empty, this is added to the menu as the last item.
menuProperties	array	Menu properties.
previousButton	string	The name attribute of the button used to navigate to the previous step.
queryParam	string	Query parameter for the step.
resetButton	string	The name attribute of the button used to reset the wizard and start from the beginning.
saveDraftButton	string	The name attribute of the button used to save draft data.
sessionKey	string	The session key for the wizard.
stepCount	integer	The total number of steps.

Wizard Behavior

Name	Type	Description
steps	array	Array of steps to be processed.
timeout	integer	The step timeout in seconds.

Public Methods

See CBehavior for inherited methods.

Name	Description
attach()	Attaches WizardBehavior to a controller.
branch()	Sets branch directives.
getCurrentStep()	Returns the 1 based index of the step being processed.
getMenu()	Returns the menu object.
getStepCount()	Returns the total number of steps.
getStepLabel()	Returns the label for a step.
process()	Process a step. This method is called form the controller action running the wizard.
read()	Returns stored data for a step or all steps.
reset()	Resets the wizard; removes all wizard session data.
resetWizard()	Calls WizardBehavior::reset() then raises the onReset event.
restore()	Restores wizard data into the session.
save()	Saves data for a step.
setMenu()	Sets the menu object or the menu object properties.

Protected Methods

See CBehavior for inherited methods.

Name	Description
cancelled()	Invoked if the wizard is cancelled.
expired()	Invoked if a step in the wizard expires.
finished()	Invoked if the wizard finishes.
getExpectedStep()	Returns the expected step.
hasCompleted()	Returns a value indicating whether or not the wizard has completed.
hasExpired()	Returns a value indicating whether or not the current step has expired.
hasStarted()	Returns a value indicating whether or not the wizard has started.
invalidStep()	Invoked if an invalid step is detected.
isValidStep()	Returns a value indicating whether or not the requested step is valid.
nextStep()	Moves the wizard to the next step.

parseSteps()	Parses the steps array into a flat array.
previousStep()	Moves the wizard to the previous step.
processStep()	Invoked when a step is to be processed.
redirect()	Redirects the wizard to a step.
saveDraft()	Invoked when draft data is to be saved.
start()	Invoked at the start of the wizard, i.e. before any steps are processed.

Events

Event	Default Event Handler	Description
onCancelled	wizardCancelled	Raised if the “Cancel” button is clicked at any step.
onExpired	wizardExpired	Raised if a step expires.
onFinished	wizardFinished	Raised after all steps have been processed or if the wizard is stopped.
onInvalidStep	wizardInvalidStep	Raised if an invalid step is detected.
onProcessStep	wizardProcessStep	Raised when a step is to be processed.
onReset	wizardReset	Raised if the “Reset” button is clicked at any step.
onSaveDraft	wizardSaveDraft	Raised if the “SaveDraft” button is clicked at any step.
onStart	wizardStart	Raised when the Wizard starts, before any steps are processed.

Wizard Behavior

Property Details

autoAdvance
```
public boolean $autoAdvance;
```

Whether to advance to the expected or next step.

If TRUE the Wizard Behavior will advance to the "*expected step*", i.e. the first step that has not been processed, after a step has been processed; this is useful if a user has returned to previous steps - the Wizard Behavior will advance directly to the step requiring input.

If false the Wizard Behavior will always advance to the next step in the *steps* array.

Defaults to TRUE

cancelButton
```
public string $cancelButton;
```

The name attribute of the button used to cancel the wizard.

Defaults to "cancel"

cancelledUrl
```
public mixed $cancelledUrl;
```

Url to be redirected to if the user clicks cancelButton.

Either string or array. If an array the first parameter is a route to a controller action, others are GET parameters in name=>value pairs.

Defaults to "/"

currentSte
```
public integer getCurrentStep()
```

Read Only

The 1 based index of the step being processed.

draftSavedUrl
```
public mixed $draftSavedUrl;
```

Url to be redirected to if the user clicks saveDraftButton.

Either string or array. If an array the first parameter is a route to a controller action, others are GET parameters in name=>value pairs.

Defaults to "/"

defaultBranch
public Boolean$ defaultBranch;
Whether to use the default in a branch group if no branch is selected.

If TRUE, the first "non-skipped" branch in a branch group will be used if no branch is selected.

Defaults to TRUE

events
```
public array $events;
```

WizardBehavior events and their handlers.

Array of eventName=>eventHandlerName pairs; the event handler names are method names in the behavior's owner.

The default provides the minimum required for a wizard plus the ability to handle invalid steps. If additional features are to be implemented the appropriate event(s) will require event handlers. See Events for details of other events that can be raised.

Defaults to array(
'onStart'=>'wizardStart',
'onProcessStep'=>'wizardProcessStep',
'onFinished'=>'wizardFinished',
'onInvalidStep'=>'wizardInvalidStep'
)

expiredUrl
```
public mixed $expiredUrl;
```

Url to be redirected to if a step expires.

Either string or array. If an array the first parameter is a route to a controller action, others are GET parameters in name=>value pairs.

Defaults to "/"

finishedUrl
```
public mixed $finishedUrl;
```

Url to be redirected to after the wizard has finished.

Either string or array. If an array the first parameter is a route to a controller action, others are GET parameters in name=>value pairs.

Defaults to "/"

forwardOnly
```
public boolean $forwardOnly;
```

Whether navigation is forward only or the user can return to previous steps.

If true the user cannot return to previous steps.

Defaults to FALSE

menu
```
public object getMenu()
public void setMenu(mixed $value)
```

The menu object with steps in the object's *items* property.

If forwardOnly==FALSE, items for previous steps are links.

menuLastItem

```
public string $menuLastItem;
```

If not empty, this is added to the menu as the last item.

Used to add the conclusion, i.e. what happens when the wizard completes - e.g. Register, to a menu.

Defaults to NULL

menuProperties

```
public array $menuProperties;
```

Menu properties.

In addition to the properties of CMenu there is an additional previousItemCssClass that is applied to previous items. If set it is applied to the default CMenu and custom menu objects; all other properties are only used with the default CMenu object, custom menu objects must be set with all properties except the items property set.

Defaults to array(
'id'=>'wzd-menu',
'activeCssClass'=>'wzd-active',
'firstItemCssClass'=>'wzd-first',
'lastItemCssClass'=>'wzd-last',
'previousItemCssClass'=>'wzd-previous'
)

previousButton

```
public string $previousButton;
```

The name attribute of the button used to navigate to the previous step.

Defaults to "previous"

queryParam

```
public string $queryParam;
```

Query parameter for the step.

Defaults to "step"

resetButton

```
public string $resetButton;
```

The name attribute of the button used to reset the wizard and start from the beginning.

Defaults to "reset"

saveDraftButton

```
public string $saveDraftButton;
```

The name attribute of the button used to save draft data.

Defaults to "save_draft"

sessionKey

```
public string $sessionKey;
```

The session key for the wizard.

Defaults to "wizard"

stepCount

```
public integer getStepCount();
```

Read Only

The total number of steps.

The value of this property may change between steps depending on which branches are selected, skipped, or deselected.

steps

```
public array $steps;
```

Array of steps to be processed.

basic example: array('login_info', 'profile', 'confirm');

Steps can be labled: array('Username and Password'=>'login_info', 'User Profile'=>'profile', 'confirm');

The steps array can also contain branch groups that are used to determine the path at runtime.

plot-branched example: array('job_application', array('degree' => array('college', 'degree_type'), 'nodegree' => 'experience'), 'confirm');

The 'branch names' (i.e. 'degree', 'nodegree') are arbitrary; they are used as selectors by the WizardBehavior::branch() method. Branches are steps array, that can also have branch groups, or a single step.

The first "non-skipped" branch in a group (see WizardBehavior::branch()) is used by default if WizardBehavior::defaultBranch==TRUE and a branch has not been selected.

Defaults to array()

timeout

```
public integer $timeout;
```

The step timeout in seconds.

Set empty for no timeout.

If a step does not complete in the timeout period WizardBehavior::expired() is called.

Defaults to NULL

Method Details

attach()

```
public void attach(CController $owner)
```

$owner	CController	The controller to which the Wizard Behavior is to be attached.

Attaches WizardBehavior to a controller.

Attaches the controller's event handlers as declared in the events property.

branch()

```
public void branch(mixed $branchDirectives)
```

$ branchDirectives	Mixed	Branch directives. string: The branch name or a comma delimited list of branch names to select array: an array of branch names to select associative array: "branch name"=>branchDirective pairs branchDirective is one of: • WizardBehavior::BRANCH_SELECT – select the branch • WizardBehavior::BRANCH_SKIP – skip the branch • WizardBehavior::BRANCH_DESELECT – deselect the branch

Selects, skips, or deselects branches.

cancelled()

```
protected void cancelled(string $step)
```

$step	String	Name of the step at which the wizard was cancelled.

Raises the onCancelled event, resets the wizard, and then redirects to WizardBehavior::cancelledURL after the event.

If this method is overridden, call the parent implementation to ensure the event is raised.

If the event handler redirects it should call the WizardBehavior::reset() method before doing so.

expired()

```
protected void expired(string $step)
```

$step	String	Name of the step that expired.

Raises the onExpired event, resets the wizard, and then redirects to WizardBehavior::expiredURL after the event.

If this method is overridden, call the parent implementation to ensure the event is raised.

If the event handler redirects it should call the WizardBehavior::reset() method before doing so.

finished()

```
protected void finished(mixed $step)
```

$step	Mixed	Why the wizard finished. • TRUE – All steps have been processed • FALSE – No steps have been processed • string – The step at which the Wizard was stopped

Raises the onFinished event, resets the wizard, and then redirects to WizardBehavior::finishedURL after the event.

If this method is overridden, call the parent implementation to ensure the event is raised.

If the event handler redirects it should call the WizardBehavior::reset() method before doing so.

getCurrentStep()

```
public integer getCurrentStep()
```

Returns the 1 based index of the step being processed.

getExpectedStep()

```
protected string getExpectedStep()
```

Returns the expected step; the expected step is the first unprocessed step.

Returns NULL is all steps have been processed.

getMenu()

```
public object getMenu()
```

Returns the menu object with steps in the object's *items* property.

If WizardBehavior::forwardOnly==FALSE, previous steps are links.

getStepCount()

```
public integer getStepCount()
```

Returns the total number of steps.

The value returned may change between steps depending on which branches are selected, skipped, or deselected.

getStepLabel()

```
public string getStepLabel(string $step=null)
```

$step	String	Name of the step to return the label for. If NULL the current step is used.

Returns the label for a step.

If the label has not been explicitly given the step is used; underscores, dots, and dashes are replaced with spaces, camelCased words are split with spaces and the first letter is each word is uppercased, e.g. "step_one", "step.one", "step-one", and "stepOne" become "Step One".

hasCompleted()

```
protected boolean hasCompleted()
```

Returns a value indicating if the wizard has completed; TRUE if the wizard has completed, FALSE if not.

hasExpired()

```
protected boolean hasExpired()
```

Returns a value indicating if the current step has expired; TRUE if the current step has expired, FALSE if not.

hasStarted()

```
protected boolean hasStarted()
```

Returns a value indicating if the wizard has started; TRUE if the wizard has started, FALSE if not.

invalidStep()

```
protected void invalidStep(string $step)
```

$step	String	Name of the invalid step.

Raises the onInvalidStep event, redirects to the expected step after the event.

If this method is overridden, call the parent implementation to ensure the event is raised.

If the event handler redirects it should call the WizardBehavior::reset() method before doing so.

isValidStep()

```
protected boolean isValidStep(string $step)
```

$step	String	Name of the step to validate.

The step is validated by:

1. Checking that the step is present in the steps array.
2. That the step is either the expected step, or, if WizardBehavior::forwardOnly==FALSE, before it.

Returns TRUE if the step is valid, FALSE if not.

nextStep()

```
protected void nextStep()
```

Moves the wizard to the next step.

If WizardBehavior::autoAdvance==TRUE the next step is the expected step, i.e. the first unprocessed step.

If WizardBehavior::autoAdvance==FALSE the next step is the next step in the steps array; whether or not it has previously been processed.

onCancelled()

```
public void onCancelled(WizardEvent $event)
```

$event	WizardEvent	The event object.

Raised if the "Cancel" button is clicked at any step.

WizardEvent::step contains the name of the step at which the button was clicked.

onExpired()

```
public void onExpired(WizardEvent $event)
```

$event	WizardEvent	The event object.

Raised if a step expires.

WizardEvent::step contains the step that expired.

onFinished()

```
public void onFinished(WizardEvent $event)
```

$event	WizardEvent	The event object.

Raised after all steps have been processed or if the wizard is stopped.

WizardEvent::step indicates why the Wizard finished:

- TRUE – All steps have been processed
- FALSE – No steps have been processed
- string – The step at which the Wizard was stopped.

WizardEvent::data contains data for all processed steps in the wizard indexed by step name.

onInvalidStep()

```
public void onInvalidStep(WizardEvent $event)
```

$event	WizardEvent	The event object.

Raised if an invalid step is detected.

An invalid step is one that:

- Is not in the steps array, or
- Is after the expected step, or
- Is before the expected step if WizardBehavior::forwardOnly==TRUE

WizardEvent::step contains the name of the invalid step.

onProcessStep()

```
public void onProcessStep(WizardEvent $event)
```

$event	WizardEvent	The event object.

Raised when a step is to be processed.

WizardEvent::step contains the name of the step to be processed.

WizardEvent::data contains current data for the step (i.e. if the user has returned to the step); this can be used to populate model attributes.

The event handler should call the WizardBehavior::save() method to save the data for the step.

onReset()

```
public void onReset(WizardEvent $event)
```

$event	WizardEvent	The event object.

Raised if the "Reset" button is clicked at any step.

WizardEvent::step contains the name of the step at which the button was clicked.

onSaveDraft()

```
public void onSaveDraft(WizardEvent $event)
```

$event	WizardEvent	The event object.

Raised if the "SaveDraft" button is clicked at any step.

WizardEvent::step contains the name of the step at which the button was clicked.

WizardEvent::data contains data to be saved.

onStart()

```
public void onStart(WizardEvent $event)
```

$event	WizardEvent	The event object.

Raised when the Wizard starts, before any steps are processed.

If the event handler sets WizardEvent::handled=FALSE the Wizard will be stopped and WizardBehavior::finished() called.

parseSteps()

```
protected void parseSteps()
```

Parses the steps array into a flat array taking into consideration any branch directives.

previousStep()

```
protected void nextStep()
```

Moves the wizard to the previous step.

Wizard Behavior

process()

```
public void process(string $step=null)
```

$step	String	The name of the step to process.

Process a step. This method is called from the controller action running the wizard.

processStep()

```
protected boolean processStep(string $step)
```

$step	String	The name of the step to process.

Raises the onProcessStep event.

If this method is overridden, call the parent implementation to ensure the event is raised.

The event handler must set the WizardEvent::handled=TRUE for the wizard to move to the next step.

read()

```
public array read(string $step=null)
```

$step	String	Name of the step to read data for. If NULL the data for all steps is returned, indexed by step name.

Returns stored data for a step or all steps.

redirect()

```
protected void redirect(string $step=null, Boolean $terminate=true, integer $statusCode=302)
```

Redirects the wizard to the step. If $step===NULL the expected step is used.

See CController::redirect().

reset()

```
public void reset()
```

Resets the wizard; removes all wizard session data.

resetWizard()

```
public void resetWizard()
```

Calls WizardBehavior::reset() then raises the onReset event.

restore()

```
public boolean restore(array $data)
```

$data	Array	The wizard data to restore into the session.

Restores wizard data into the session.

$data[0] is the step data, $data[1] branch directives, $data[2] the step timeout.

If restoring data stored from an onSaveDraft event, the data will be in the correct format.

Returns TRUE if the data was successfully restored, FALSE if not.

save()

```
public void save(mixed $data, string $step=null)
```

$data	Mixed	The data to store for the step.
$step	String	Name of the step to save data for. If NULL the current step is used.

Saves data for a step.

saveDraft()

```
protected void saveDraft(string $step)
```

$step	String	The step at which the draft was saved.

Raises the onSaveDraft event , resets the wizard then redirects to WizardBehavior::draftSavedURL after the event.

If this method is overridden, call the parent implementation to ensure the event is raised.

If the event handler redirects it should call the WizardBehavior::reset() method before doing so.

setMenu()

```
public void setMenu(mixed $value)
```

Sets the menu object or the menu object properties.

If the value is an object, it must have a property named items.

If the value is an array it is an array of CMenu property name=>value pairs that are merged with WizardBehavior::menuProperties.

start()

```
protected boolean start()
```

Raises the onStart event .

If this method is overridden, call the parent implementation to ensure the event is raised.

The event handler must set WizardEvent::handled=TRUE for the wizard to continue.

If WizardEvent::handled==FALSE the WizardBehavior::finish() is called.

Wizard Behavior

WizardEvent

Inheritance	WizardEvent » CEvent » CComponent

WizardEvent is the event raised by WizardBehavior.

Public Properties

See CEvent for inherited properties.

Name	Type	Description
data	Array	Data for the event.
step	String	The step that caused the event to be raised.

Public Methods

See CEvent for inherited methods.

Name	Description
getData()	Returns the data for the event.
getStep()	Returns the step that caused the event to be raised.

Wizard Behavior

Property Details

data

```
public mixed getData();
```

Read Only

Returns the data for the event; data for the step or all processed steps, or draft data, depending on which event is raised.

step

```
public mixed getStep();
```

Read Only

Returns the step that caused the event to be raised.

Method Details

getData()

```
public mixed getData()
```

Returns the data for the event; data for the step or all processed steps, or draft data, depending on which event is raised.

getStep()

```
public integer getStep()
```

Returns the name of the step that caused the event to be raised.

Using Wizard Behavior

Attaching the Wizard Behavior

Attaching the Wizard Behavior to a controller can be done in two ways:

1. If the configuration of the wizard is fixed, declare the Wizard Behavior in the CController::behaviors() method:

```
class ExampleController extends CController {
        …
  public function behaviors() {
    return array(
      'wizard'=>array(
        'class'=>'path.to.WizardBehavior',
        'steps'=>array('step1','step2',…,'stepN'),
        // other wizard configuration
      )
    );
  }
        …
}
```

Figure 1 –Wizard Configuration in a Controller's behaviors() Method

2. If the configuration is determined at runtime, e.g. if there are multiple actions in the controller that use the Wizard Behavior – each with different steps or the required steps are determined at runtime, the Wizard Behavior can be attached in the CContoller::beforeAction() method:

```
class ExampleController extends CController {
        …
  public function beforeAction($action) {
    switch ($action->id) {
      case 'action1':
        $this->attachBehavior('wizard'),array(
          'class'=>'path.to.WizardBehavior',
          'steps'=>array('a1s1','a1s2','a1s3'),
          // other wizard configuration
        ));
        break;
      case 'action2':
        $this->attachBehavior('wizard'),array(
          'class'=>'path.to.WizardBehavior',
          'steps'=>array('a2s1','a2s2','a2s3'),
          // other wizard configuration
        ));
        break;
      default:
        break;
    }
    return parent::beforeAction($action);
  }
        …
}
```

Figure 2 –Wizard Configuration in a Controller's beforeAction() Method

Running the Wizard

The Wizard Behavior uses a single controller action to process steps and all steps submit their forms to this action. This means the action can be very simple; it only needs to pass the step to be processed to the Wizard Behavior.

```
class ExampleController extends CController {
        …
  public function actionWizard($step=null) {
    $this->process($step);
  }
        …
}
```

Figure 3 – Example controller action

Events

The Wizard Behavior raises various events and your controller must have event handler methods to handle them; if there are events you do not wish to handle you must set WizardBehavior::events appropriately – by default an event handler is declared for all events.

The events and the default names of the event handlers are shown below; the names of the event handlers can be changed in WizardBehavior::events.

The events raised are WizardEvent, this is extended from CEvent.

onStart Event (wizardStart)

This event is raised before any steps are processed. This event must be handled. As a minimum the event handler must set WizardEvent::handled=TRUE (unless it determines that the Wizard should not run).

```
class ExampleController extends CController {
        …
  public function wizardStart($event) {
    $event->handled = true;
  }
        …
}
```

Figure 4 – Example onStarted Event Handler

onProcessStep Event (wizardProcessStep)

This handles the processing of each step. It receives a WizardEvent, which contains the name of the step in its step property; this can be used to determine how to process the step, e.g. what model to use.

The event handler is responsible for rendering and validating the form, and saving the data to be persisted.

The event handler must set WizardEvent::handled=TRUE to advance to the next step.

If each step follows the same pattern the event handler can be used to process all steps. Alternatively, if the processing of steps varies according to the step, the event handler can act as a delegator and call other methods to process each step; examples of both types of onProcessStep event handlers are shown below.

```
class ExampleController extends CController {
        …
  public function wizardProcessStep($event) {
    $modelName = ucfirst($event->step);
    $model = new $modelName('scenario');
    if ($event->data)
      $model->attributes = $event->data;

    $form = new CForm("path.to.get.{$event->step}.form.configuration",
$model);

    if ($form->submitted() && $form->validate()) {
      $event->sender->save($model->attributes);
      $event->handled = true;
    }
    else
      $this->render($event->step, compact('form'));
  }
        …
}
```

Figure 5 –onProcessStep Event Handler Example

This code is not dissimilar to an action used to handle a form.

The handler below delegates event handling to methods named “process<StepName>”.

```
class ExampleController extends CController {
        …
  public function wizardProcessStep($event) {
    $name = 'process'.ucfirst($event->step);
    if (method_exists($this, $name))
      $event->handled = call_user_func(array($this,$name), $event);
    else
      throw new CException(Yii::t('yii','{class} does not have a method
named "{name}".', array('{class}'=>get_class($this), '{name}'=>$name)));
  }
        …
}
```

Figure 6 –onProcessStep Delegating Event Handler Example

onFinished Event (wizardFinished)

This is raised when the Wizard Behavior finishes. The WizardEvent::step property can be used to determine why the Wizard Behavior finished.

WizardEvent::step	Description	WizardEvent::data
TRUE	All steps have been processed	Associative array of data for all steps indexed by step name.
FALSE	No steps have been processed	Empty
<string>	The step at which the Wizard was stopped.	Associative array of data for all processed steps indexed by step name.

Table 1 - Wizard onFinished Event Values

This event handler typically saves the data contained in WizardEvent::data to persistent storage.

If the eventHandler sets WizardEvent::handled=TRUE, the Wizard Behavior will flush Wizard data from the session and redirect to WizardBehavior::finishedUrl. If the event handler redirects it should first call WizardBehavior::reset() to flush the Wizard Behavior session data.

onCancelled Event (wizardCancelled)

This event is raised if the user clicks the **Cancel** button. WizardEvent::step contains the name of the step at which the wizard was cancelled; WizardEvent::data is an associative array of data for all processed steps indexed by step name.

If the eventHandler sets WizardEvent::handled=TRUE, the Wizard Behavior will flush Wizard data from the session and redirect to WizardBehavior::cancelledUrl. If the event handler redirects it should first call WizardBehavior::reset() to flush the Wizard Behavior session data.

onSaveDraft Event (wizardSaveDraft)

This event is raised if the user clicks the **Save Draft** button. WizardEvent::step contains the name of the step at which the user chose to save draft data; WizardEvent::data is an associative array of data for all processed steps indexed by step name.

The event handler should save the data contained in WizardEvent::data to persistent storage such that it can be recovered by the user at a later time. See *Saving and Recovering* Draft Data for details of how to recover data and restore the WizardBehavior session.

If the eventHandler sets WizardEvent::handled=TRUE, the Wizard Behavior will flush Wizard data from the session and redirect to WizardBehavior::draftSavedUrl. If the event handler redirects it should first call WizardBehavior::reset() to flush the Wizard Behavior session data.

onInvalidStep Event (wizardSaveDraft)

This event is raised if an invalid step is detected.

Wizard Behavior will redirect to the expected step after this event.

onReset Event (wizardSaveDraft)

This event is raised if the user clicks the **Reset** button.

WizardBehavior will restart after this event.

The Steps Array

The steps array defines the "path" the Wizard Behavior will take. At its simplest, the steps array is an array of step names.

```
array('step1','step2',…,'stepN')
```

Figure 7 – Simple Steps Array

Step Labels

Labels for steps can be given by setting the key for a step, for example:

```
array('First Step'=>'step_one','step_two',…,'Last Step'=>'stepN')
```

Figure 8 – Simple Steps Array

If a step does not explicitly have a label (e.g. step_two in the above example) the label is generated from the step name by replacing underscores, dots, and dashes with spaces, separating camelCased words with spaces and uppercasing the first letter of each word. For example "step_two", "step.two", "step-two", and "stepTwo" become "Step Two".

Plot-Branching Navigation

Plot-branching navigation (PBN) allows the Wizard Behavior to take different paths depending on user input. For example, you may wish to ask different questions depending on whether the user has a pet dog or a pet cat, and perhaps skip all those questions if they have neither.

The steps array can be used to specify the branch options; two methods are used to determine the branch to be used at runtime.

Branch groups are associative arrays; the keys are branch names and the values are steps arrays which can also contain branch groups.

Note: Branching can be nested as deep as resources allow.

Note: Branch names do not have to be unique among branch groups. This allows a step to influence one or more branches in the wizard.

PBN Examples

```
array('step1','step2','pet',array('dog'=>array('step3','step4'),
'cat'=>array('step4','step5')),'step6')
```

Figure 9 – PBN Example 1

If `WizardBehavior::defaultBranch==TRUE` (the default), the first branch in a branch group is used, so by doing nothing the path taken through the steps array above is:

`step1, step2, pet, step3, step4, step6`
(`step3` and `step4` coming from the default 'dog' branch)

If `WizardBehavior::defaultBranch==FALSE`, all branches are skipped; in this case the path taken through the steps array above is:

`step1, step2, pet, step6`

Using another example:

```
array('step1','step2','pet',array('dog'=>array('step3'),'step4'
'cat'=>array('step5')),'step6')
```

Figure 10 – PBN Example 2

With the steps array, if `WizardBehavior::defaultBranch==TRUE` the default the default path taken is:

`step1, step2, pet, step3, step4, step5, step6`
(note that both the dog and cat branches are included as they are in separate branch groups, each of which use the default branch)

If `WizardBehavior::defaultBranch==FALSE`, all branches are skipped; in this case the path taken through the steps array above is:

`step1, step2, pet, step4, step6`

Selecting, Skipping, and Deselecting Branches

The onProcessStep event handler can tell the Wizard Behavior which branches to select, skip, or deselect; it does this by calling the WizardBehavior::branch() method. The parameter passed to WizardBehavior::branch() can be:

- A string – Selects the branch, e.g. “dog” will select the branch named dog.
- A list; either a comma delimited string or an array – Selects multiple branches.
- An associative array of branch name=>branch directive pairs – Acts on multiple branches according to their branch directives.

A branch directive is one of:

Branch Directive	Description
WizardBehavior::BRANCH_SELECT	Selects the branch.
WizardBehavior::BRANCH_SKIP	Causes the branch to be skipped and the next branch in the branch group to be used as the default.
WizardBehavior::BRANCH_DESELECT	Deselects the branch.

Table 2 - Branch Directives

Important: Branch options are used in the order they are selected. It is therefore necessary to deselect a branch option if it has previously been selected and a later branch in a branch group is to be selected.
For example, using PBN Example 1, if the “dog” branch option had been selected at an earlier point in the wizard and now the “cat” branch option was required, the “dog” branch option must be deselected.

Using the first example, if the users selects ‘cat’ as their pet, the onProcessStep should set the WizardEvent::branch property to any of the following; the last being the preferred as it ensures that all the branches are in the correct state:

- `'cat'`
- `array('cat')`
- `array('cat'=>WizardBehavior::BRANCH_SELECT)`
-
```
array(
  'cat'=>WizardBehavior::BRANCH_SELECT,
  'dog'=>WizardBehavior::BRANCH_DESELECT
)
```

The onProcessStep event handler would be similar to:

```
class ExampleController extends CController {
        …
  public function wizardProcessStep($event) {
    $modelName = ucfirst($event->step);
    $model = new $modelName('scenario');
    if ($event->data)
      $model->attributes = $event->data;

    $form = new CForm('path.to.form.configuration.for.the.step', $model);

    if ($form->submitted() && $form->validate()) {
      if ($event->step==='pet') {
        if ($model->pet==='dog') // Select the "dog" branch
          $event->sender->branch(array(
            'dog'=>WizardBehavior::BRANCH_SELECT,
            'cat'=>WizardBehavior::BRANCH_DESELECT
          ));
        elseif ($model->pet==='cat') // Select the "cat" branch
          $event->sender->branch(array(
            'dog'=>WizardBehavior::BRANCH_DESELECT,
            'cat'=>WizardBehavior::BRANCH_SELECT
          ));
        else ($model->pet==='neither') // Skip both branches
          $event->sender->branch(array(
            'dog'=>WizardBehavior::BRANCH_SKIP,
            'cat'=>WizardBehavior::BRANCH_SKIP
          ));
      }
      $event->sender->save($model->attributes);
      $event->handled = true;
    }
    else
      $this->render($event->step, compact('form'));
  }
        …
}
```

Figure 11 – onProcessEvent Handler Example with PBN

The table below summarises which branch from a branch group is used.

$defaultBranch	Branch or Branches Selected	No Branches Selected
TRUE	The branch first selected	The first non-skipped branch
FALSE	The branch first selected	None

Table 3 - Branch Use Summary

Branch Naming

Branch names within a branch group must be unique; branch names across branch groups do not need to be unique, and this can be used to good effect. Consider the steps array below:

```
array('step1','step2','pet',array('dog'=>array('step3','step4'),
'cat'=>array('step4','step5')),'step6',array('dog'=>array('step7',
'step8'),'cat'=>array('step9','step10')),'step11')
```

Figure 12 – PBN Example 3

This has two branch groups each with a “dog” and “cat” branch. The branch selected in the “pet” step’s onProcessStep event handler will be selected for both branch groups (unless altered at some point). So, if the user selects “cat” at the “pet” step, the path taken will be:

```
step1, step2, pet, step4, step5, step6, step9, step10, step11
```

Saving and Recovering Draft Data

Saving Draft Data

To save draft data provide a "Save" button. This will raise the onSaveDraft event, the event handler should save the data to persistent storage.

Note: If you intend to save the draft data after a step has been successfully completed your onProcessStep event handler should allow step submission using the "Save" button. So the check for form submission and validation in the onProcessStep event handler will look like:

```
if (($form->submitted() || $form->submitted($event->sender-
>saveDraftButton)) && $form->validate()) {
      // Save the step data
}
else {
      // Submission failed
}
```

Recovering Draft Data

To recover draft data you need a controller action to recover draft data from persistent storage and

```
class ExampleController extends CController {
        …
  public function actionRecoverDraft($draftId) {
    $data = Model::model()->recoverDraft($draftId);
    if ($data===false || ! $this->restore($data);)
      // Data recovery failed
    else
      $this->redirect(array('wizardAction'));
  }
        …
}
```

restore it into the Wizard Behavior; it will look something like:

Figure 13 – Recover Data Example

The data is an array with two entries; the first entry is the step data – an associative array with step names as the keys and the step data as values, the second entry is the branch data – an associative array with branch names as the keys and either “branch” or “skip” as the value.

> **Note:** If restoring data previously stored from an onSaveDraft event, the data will be in the correct format.

Form Submit Buttons

In addition to the button used to submit the form data, the Wizard Behavior supports the following submit buttons to provide some features:

Default Button Name	Description
previous	Allows the user to go back to previous steps in the wizard. Data is preserved. Disabled if fowardOnly==TRUE
reset	Allows the user to restart the wizard from the beginning. Data is *not* preserved. Disabled if fowardOnly==TRUE
save_draft	Allows the user to save data entered so far.

Figure 14 – Wizard Behavior Form Buttons

To provide these features, forms in the Wizard need to have the appropriate buttons. The `name` attribute of the buttons can be configured.

Menu

Wizard Behavior can generate a menu of steps that can be used in a view. The menu will contain links to previous steps in the wizard unless `forwardOnly==TRUE`.

The default is to generate a CMenu object; this can be run in the view to render the menu.

```
$this->menu->run();
Or
$this->getMenu()->run()
```

Figure 15 – Wizard Behavior Menu View Code Examples

```
$this->menu = $customMenuObject;
Or
$this->setMenu($customMenuObject);
```

Use `WizardBehavior::setMenu()` to provide a custom menu object:

Figure 16 – Wizard Behavior Set Custom Menu Examples

or to configure the CMenu object on the fly:

```
$this->menu = array(CMenu options);
Or
$this->setMenu(array(CMenu options));
```

Figure 17 – Wizard Behavior Set CMenu Properties Examples

If using a custom menu object it must have an items property that accepts items in the same format as CMenu.

The Wizard Behavior menu supports one additional property – previousItemCssClass – that can be used to provide a class to previous steps in the menu.